Learning Configurations of Operating Environment of Autonomous Vehicles to Maximize their Collisions

Chengjie Lu, Yize Shi, Huihui Zhang, Man Zhang, Tiexin Wang, Tao Yue and Shaukat Ali

Abstract—Autonomous vehicles must operate safely in their dynamic and continuously-changing environment. However, the operating environment of an autonomous vehicle is complicated and full of various types of uncertainties. Additionally, the operating environment has many configurations, including static and dynamic obstacles with which an autonomous vehicle must avoid collisions. Though various approaches targeting environment configuration for autonomous vehicles have shown promising results, their effectiveness in dealing with a continuous-changing environment is limited. Thus, it is essential to learn realistic environment configurations of continuously-changing environment, under which an autonomous vehicle should be tested regarding its ability to avoid collisions. Featured with agents dynamically interacting with the environment, Reinforcement Learning (RL) has shown great potential in dealing with complicated problems requiring adapting to the environment. To this end, we present an RL-based environment configuration learning approach, i.e., *DeepCollision*, which intelligently learns environment configurations that lead an autonomous vehicle to crash. DeepCollision employs Deep Q-Learning as the RL solution, and selects *collision probability* as the safety measure, to construct the reward function. We trained four DeepCollision models and conducted an experiment to compare them with two baselines, i.e., random and greedy. Results show that DeepCollision demonstrated significantly better effectiveness in generating collisions compared with the baselines. We also provide recommendations on configuring DeepCollision with the most suitable time interval based on different road structures.

Index Terms—Autonomous Vehicle Testing, Reinforcement Learning, Environment Configuration.

1 Introduction

Autonomous vehicles must operate safely in their dynamic and continuously-changing environment. However, such an environment is dynamic and complicated with diverse uncertainties [1], [2], e.g., when, where, and such obstacles will appear [3]. In addition, it can have many types of static and dynamic obstacles with which the autonomous vehicle must avoid collisions. Thus, it is vital to learn critical environment configurations to test an autonomous vehicle under various configurations to avoid collisions [4].

Search-Based Testing (SBT) techniques have been widely applied to address operating environment configuration problems for testing autonomous driving systems (ADSs) or autonomous driving assistance systems (ADASs) [5], [6], [7], [8], [9]. Such approaches commonly formulate an environment configuration problem (e.g., searching critical configurations of driving scenarios) as an optimization problem, then adopt or even extend search algorithms (e.g., Non-dominated Sorting Genetic Algorithm II (NSGA-II) [10]) to solve it. However, the input space of the environment of an ADS or ADAS is often large. Consequently, one needs to effectively identify critical test scenarios within available computational cost [9]. Towards this end, some works [7], [9] focus on exploring search spaces of ADSs or ADASs using search algorithms. For example, Ben Abdessalem et al. [7] used NSGA-II for searching scenarios to test an ADAS, together with neural networks to reduce computational overhead, while generated scenarios are limited to one pedestrian and one vehicle at constant speed. Such scenarios seem relatively simple for testing an autonomous vehicle in a real-world context. Though SBT techniques have gained successes in some contexts, the effectiveness of existing SBT techniques in dealing with a continuously-changing operating environment of autonomous vehicles is limited. This is mainly because, in our opinion, such approaches typically do not bring sufficient considerations of interactions of individuals (often encoding test scenarios) with the environment [11], which is however critical, given its inherent complexity and uncertainty [4], [12], e.g., involving obstacles (e.g., pedestrians and other vehicles) with their behaviors (e.g., speed and position) changing constantly and dynamically.

In recent years, reinforcement learning (RL) has been applied in autonomous driving [4], e.g., for motion planning [13] and lane following [14]. The foundation of employing RL is to formulate a problem into Markov Decision Process (MDP). An RL process is adaptive to dynamic and

- *C. Lu, Y. Shi, and T. Wang are with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing, 210016, China. E-mail: {chengjielu, yizeshi, tiexin.wang}@nuaa.edu.cn.*
- *H. Zhang is with the School of Computer Science and Technology, Qilu University of Technology (Shandong Academy of Sciences), Jinan, 250353, China, and Weifang University, Weifang, 261061, China. E-mail: huihui@wfu.edu.cn.*
- *M. Zhang is with Kristiania University College, Oslo, 999026, Norway. E-mail: man.zhang@kristiania.no.*
- *T. Yue (the corresponding author) is with Nanjing University of Aeronautics and Astronautics (Nanjing, 210016, China), and Simula Research Laboratory (Oslo, 999026, Norway). E-mail: taoyue@ieee.org.*
- *S. Ali is with Simula Research Laboratory, Oslo, 999026, Norway. E-mail: shaukat@simula.no.*

continuously-changing environment, and an agent learns to improve its performance in an assigned task by interacting with the environment without the prior knowledge of the environment and the need of labelled training data [4]. Deep Q-Learning (DQN) [15] is an RL algorithm based on Q-Learning [16], which has demonstrated good performance in solving complicated problems with high-dimensional observation space, in the domains of robotics [17], game industry [15] and autonomous driving (e.g., lane following [14] and motion planning [13]).

Motivated by these successful applications of DQN for solving complicated problems with high-dimensional space, we propose DeepCollision, which formulates the problem of environment configuration learning as an MDP and uses the DQN to adaptively and continuously learn environment configurations that could effectively lead an autonomous vehicle to collide with obstacles in its operating environment. To formulate actions of configuring the environment, which together form the DQN's action space, we adopt a list of key configurable environment parameters and implement their invocations as REST API endpoints. Besides, to guide the DQN algorithm towards generating collision-prone scenarios, DeepCollision selects *collision probability*, as the safety measure, to construct the reward function of DQN. Note that DeepCollision has no intention to maximize the number of configurations, i.e., generating as many test scenarios as possible, as that would end up with practicing exhaustive testing, only possible when there is no any practical constraint on computational resources and time budget [18]. Instead, DeepCollision aims to find critical scenarios with fewer configurations within a limited budget.

The environment configuration process of DeepCollision continues throughout a complete driving task (e.g., from its origin to destination) of an autonomous vehicle. More specifically, in each discrete instant while driving in the environment, the RL agent decides an action to configure the environment. We found that the time interval between two adjacent discrete instants is a key factor for effectively and efficiently configuring the environment in our pilot study. Thus, we trained four DeepCollision models, each of them corresponds to one of the four time interval settings: 4s, 6s, 8s and 10s. We performed the evaluation with an industrial scale ADS (i.e., Apollo [19]) and a commonly used autonomous driving simulator (i.e., LGSVL [20]). We experimented with four typical driving roads with various complexities in their road structures, and compared the performance of DeepCollision with two strategies, i.e., random and greedy. The evaluation results show that: 1) DeepCollision outperformed the baselines in terms of generating more effective driving scenarios, e.g., on average, 186% and 126% improvements on generating collisions and potential collision scenarios, respectively; 2) The four DeepCollision models performed differently with respect to various road complexities, thus, we provided recommendations of configuring DeepCollision with the most suitable time interval setting based on different road structures.

To summarize, the contributions of this paper are below:

- With the aim to test ADSs, we propose a novel RL-based approach to learn operating environment configurations of autonomous vehicles, including formalizing environment configuration learning as an MDP and adopting DQN as the RL solution;
- To handle the environment configuration process of an autonomous vehicle, we present a lightweight and extensible *DeepCollision* framework providing 52 REST API endpoints to configure the environment and obtain states of both the autonomous vehicle and its operating environment; and
- We conducted an extensive empirical study with an industrial scale ADS and simulator, and results show that DeepCollision outperforms the baselines. Further, we provide recommendations of configuring DeepCollision with the most suitable time interval setting based on different road structures.

The rest of the paper is organized as follows. Section 2 introduces reinforcement learning, especially the DQN algorithm. We report the related work in Section 3 and present DeepCollision in Section 4. Evaluation is presented in Section 5, followed by discussions in Section 6. We conclude the paper in Section 7.

2 Deep Q-Learning

Reinforcement Learning (RL) is concerned with learning policies for agents interacting with unknown environments [11]. In RL, an agent interacts with its environment through observations (states), actions and rewards. A typical RL problem can be formalized with Markov Decision Process (MDP) as a 5-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma >$, which consists of a set of states $\mathcal{S}$ and actions $\mathcal{A}$, a reward function $\mathcal{R}(s_t, a_t)$, a probability distribution $\mathcal{P}(s_{t+1}|s_t, a_t)$, and a discount factor $\gamma \in [0, 1]$. The set of states $\mathcal{S}$, referred as observation space, specifies what the agent can know about its environment, while the set of actions $\mathcal{A}$, referred as action space, specifies how the agent can act on its environment. The agent's performance is evaluated with the reward function $\mathcal{R}$. The probability distribution $\mathcal{P}$ depicts which state the environment will transit to after an action is executed. The discount factor γ controls how an agent regards future rewards; a small γ value encourages the agent to maximize short-term rewards, whereas high γ values lead the agent to focus more on maximizing long-term rewards. Hence, the return (representing the cumulative reward) can be defined as $R_t = \sum_{t>0} \gamma^{t-1} r_t$, where further rewards are discounted by discount factor γ. An RL agent interacts with its environment at each discrete time step t. First, the agent observes a state $s_t \in \mathcal{S}$, then selects an action $a_t \in \mathcal{A}$. After the execution of a_t, a reward $r_t \sim \mathcal{R}(s_t, a_t)$ is fed back to the agent, along with a newly observed state $s_{t+1} \sim \mathcal{P}(s_{t+1}|s_t, a_t)$.

In RL, a policy $\pi : \mathcal{S} \times \mathcal{A} \rightarrow [0, 1]$ is a probability distribution over actions given states, and $\pi(a_t|s_t)$ represents the probability of choosing action a_t at state s_t. RL aims to find the optimal policy π^*, which can lead to the highest expected cumulative reward, which is achieved via an action-value function estimating the goodness of preforming an action in a certain state. Such a function follows policy π to return the expected cumulative reward of a sequence of actions. In Q-Learning [16], such an action-value function is called Q-value function $Q^\pi(s, a) : (\mathcal{S}, \mathcal{A}) \rightarrow \mathbb{R}$, which is

the discounted expected return of rewards given the state, action, and policy:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}[R_t|s_t, a_t] \quad (1)$$

The Q-value of a given state-action pair (s_t, a_t), i.e., $Q^{\pi}(s_t, a_t)$, is an estimate of the expected future reward obtained from pair (s_t, a_t) with policy π. The optimal action-value function $Q^*(s_t, a_t)$ provides maximum values in all states, based on the Bellman equation:

$$Q^*(s_t, a_t) = \mathbb{E}_{\pi^*}[r_t + \gamma \max_{a_{t+1}} Q^*(s_{t+1}, a_{t+1})] \quad (2)$$

The equation says that the maximum return from s_t and a_t is the sum of immediate reward r_t and γ discounted return obtained with the optimal policy until the end of the episode, i.e., the maximum reward from the next state s_{t+1}. Expectation $\mathbb{E}$ is computed over the distribution of immediate rewards r_t and possible next states s_{t+1}. For discrete and finite number of states and actions, a simple way to represent a Q-value function is to use a table (Q-table) of values for all state-action pairs. Such a table is arbitrarily initialized and updated with data representing the agent's experience at each time step:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(r_t + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)) \quad (3)$$

where, $0 < \alpha \leq 1$ is the learning rate. A larger learning rate leads to a stronger influence of new data on an update.

Many challenging domains such as autonomous driving often have high-dimensional state and/or action spaces. Therefore, it is very challenging to maintain a separate Q-table for each $\mathcal{S} \times \mathcal{A}$ [21]. Instead, a function is often used to approximate Q-values. In particular, DQN [15] approximates and stores Q-values with a deep neural network (referred as Deep-Q-Network) parameterized by weights and biases, together denoted as θ: $Q(s, a; \theta) \approx Q^*(s, a)$. In DQN, the mechanism of ensuring the stability of training is that, it first uses a replay memory to store a certain number of state transitions $< s_t, a_t, r_t, s_{t+1} >$ and randomly samples data for Q-Network training. The aim of this step is to reduce correlations among samples and thereby increase sample efficiency via the reuse of data. Second, to be more stable as compared to Q-Learning, a separate target network $\hat{Q}(s, a; \theta^-)$ parameterized with θ^- is used for generating target Q-values in Q-Learning updates. $\hat{Q}$ is identical to the main network Q except that its parameters θ^- are cloned from θ every C update. In this case, the Q-value function is updated as:

$$\theta \leftarrow \theta + \alpha(y_{target} - Q(s_t, a_t; \theta))\nabla_{\theta} Q(s_t, a_t; \theta) \quad (4)$$

where y_{target} is the target value computed as:

$$y_{target} = r_t + \gamma \max_{a_{t+1}} \hat{Q}(s_{t+1}, a_{t+1}; \theta^-) \quad (5)$$

Last, at each training iteration, a batch of memory is sampled uniformly from the replay memory for the Q-Network training. The training is based on target Q-values from $\hat{Q}$. The loss function of Q-Network in DQN is defined as:

$$\mathcal{L}(\theta) = \mathbb{E}[(y_{target} - Q(s_t, a_t; \theta))^2] \quad (6)$$

3 Related Work

We compare our work with existing search-based ADS testing techniques in Section 3.1, and machine learning (ML)-based ADS testing in Section 3.2.

3.1 Search-based ADS Testing

Search-based approaches are typically applied for black-box testing of ADSs and Advanced Driver Assistance Systems (ADASs) [5], [6], [7], [8], [9], [22], [23], [24]. These methods mostly focus on the automated test generation with optimization objectives such as minimizing distances to obstacles or unsafe area [5], [9], [22], maximizing the speed of a vehicle at the time of collision [9], violating safety requirements (e.g., respecting to a defined speed limit) [8], minimizing time to collision [6], [7], [23], and generating collision scenarios [24], [25] while searching for alternative configurations of ADSs to avoid such collision scenarios [24].

Testing ADSs or ADASs is often complex, which requires solving more than one objectives. Therefore, search-based testing with a single objective may not be effective [26]. Towards this end, works based on multi-objective search algorithms select and/or generate safety-critical scenarios. For instance, NSGAII-SM [7] combines multi-objective search algorithm NSGA-II [10] with surrogate models [27] to test an ADAS with three objectives, minimizing the distance between the subject vehicle and the pedestrian, minimizing the distance between the pedestrian and the acute warning area, and minimizing the time to collision. NSGAII-DT [9] tests vision-based control systems by combining NSGA-II with decision tree classification models to generate distinct and critical test scenarios and identifying conditions that are likely to exhibit system failures. NSGAII-DT uses the speed at the time of collision and the distance to obstacles as the optimization objectives. FITEST [8] applies multi-objective search to detect failures caused by feature interactions. For example, a feature interaction may arise when a braking command issued by emergency braking system is overridden by cruise control's command of maintaining the same speed as that of the front vehicle. To guide the test generation process towards revealing undesired feature interactions that might lead to violations of system safety requirements, FITEST integrates a set of hybrid test objectives designed based on distance functions measuring how far of violating system safety requirements such as no collision with pedestrians, stopping at a stop sign, and respecting to a defined speed limit. Multi-objective search algorithm, i.e., NSGA-II, is used to search for avoidable collision [24]. These collisions can occur with default ADS configurations, but not with alternative ADS configurations.

To create fitness functions for testing ADSs with search, Hauer et al. [22] provided templates to formulate four fitness functions: minimizing the distance between the subject vehicle and obstacles (the safety goal), and ensuring correct positions of scenario obstacles and correct timing of scenario events. The proposed templates were demonstrated with collision and close-to-collision scenarios.

Search-based methods have shown promising results in testing ADSs; however, their effectiveness to deal with continuously-changing environment as is the case for autonomous vehicles is rather limited. This is mainly because typically such methods do not consider interactions of individuals (often encoding test scenarios) with the environment. Such interaction information, e.g., which states an individual passes through during its lifetime and which actions it selects over those states, are ignored for finding optimal solutions in search-based methods. Such information however has been proven to be efficient for decision making in reinforcement learning, especially in an open environment under uncertainty [28], [29]. Further, for solving complicated problems in a continuous environment, the success of a decision-making strategy is the culmination of a series of actions over extended time periods, while search-based methods cannot take into account of time-series decisions (sequentially actions) [11] when performing optimization. Therefore, it is not efficient for dealing with long-term decisions. Therefore, we opt for RL in DeepCollision.

3.2 ML-based ADS Testing

Works such as [30], [31], [32], [33] investigate the application of ML techniques to test ADSs. These approaches mainly focus on generating adversarial examples to test Deep Neural Networks (DNNs) applied in ADSs.

Several ML-based approaches leverage real-world changes in driving conditions such as heavy rain, heavy snow or adverse lighting to synthesize adversarial driving conditions. DeepRoad [30] relies on metamorphic testing techniques for identifying inconsistent behaviors of DNN-based ADSs under various weather conditions. DeepXplore [31] is a white box framework for systematically testing real-world deep learning systems like ADSs, which introduces the neuron coverage for measuring how many rules in a DNN are exercised by a set of inputs to maximize both the neuron coverage and the number of potentially erroneous behaviors without manual labels. DeepTest [32] is a systematic testing tool focusing on generating realistic synthetic images by applying image transformations and mimic different real-world phenomena (e.g., weather condition, object movements). DeepTest also applies the neuron coverage for evaluating DNNs in ADSs and the conducted empirical study shows that changes in the neuron coverage correlate with changes in an autonomous vehicle's behavior. DeepBillboard [33] is a systematic physical-world adversarial testing approach, which focuses on drive-by billboards. DeepBillboard systematically generates adversarial perturbations that can mislead CNN-based steering models under dynamic changing driving conditions.

Due to the continuously and quickly changing driving environment, supervised and unsupervised ML-based techniques tend to be not adaptive in contexts where a prediction model needs to adapt to continuously system status changes. Existing DNN or CNN-based testing approaches (e.g., [31], [32]) may not sufficiently reflect real-world driving scenes [30]. To overcome this challenge, Corse et al. [34] investigated to apply RL and proposed a framework by integrating an extended Adaptive Stress Testing [35] method and two reward augmentation designs to identify critical and distinct scenarios to fail ADSs.

Besides the above-mentioned ML-based methods directly targeting at testing DNNs in ADSs, several works focusing on deep learning testing such as [36], [37], [38], [39] also exhibit the potential for testing ADSs. For example, test coverage metrics such as neuron boundary coverage [38] and neuron coverage [39] have been proposed for guiding the testing of ADSs [31], [32] to achieve higher coverage of activated neurons. However, due to an ADS being more complicated than a single DNN model, such DNN testing methods also exhibit limitations when being applied for testing multi-module ADSs. Moreover, it has been argued that whether metrics such as the neuron coverage are meaningful for testing a real-world ADS [40].

As compared to the above works, we propose DeepCollision by applying RL for ADS testing. DeepCollision relies on DQN to automatically learn and dynamically change an autonomous vehicle's operating environment. Such changes are instant and directly reflected in the employed simulator to simulate a driving scenarios. Inheriting the feature of an agent dynamically interacting with the environment from RL, DeepCollision is adaptive to the continuously and quickly changing environment. Furthermore, the operating environment is configured dynamically during a driving task from its origin to destination.

4 DeepCollision Method

In this section, we present an overview of DeepCollision, provide a definition of configurable environment parameters and describe the calculation of collision probability, then describe how we mathematically formalize operating environment configuration problem of autonomous vehicles as an MDP, which makes it possible for reinforcement learning to play a role.

4.1 DeepCollision Overview

As show in Figure 1, DeepCollision learns environment configurations with DQN to maximize collisions of an Autonomous Vehicle Under Test (AVUT), employs a *Simulator* (e.g., LGSVL [20]) to simulate and configure a *Testing Environment* comprising the AVUT and its operating environment, and integrates with an *Autopilot Algorithm Platform* (e.g., the Baidu Apollo [19]) deployed on the AVUT.

More specifically, the employed DQN of DeepCollision generates actions to configure the environment of the AVUT, e.g., adding pedestrians crossing a road. As shown in Figure 1, at each time step t, the DQN agent observes a state S_t describing the current states of the AVUT and its environment. With the state, DeepCollision decides an action A_t based on the Q-network and the behavior policy. With *Environment Configuration REST API Endpoints* (particularly developed for DeepCollision), such an action A_t can be considered as an HTTP request for accessing the simulator to introduce new environment configurations. After the AVUT driving in a newly configured environment for a fixed time period, i.e., at $t+1$, both the AVUT and its environment enter a new state S_{t+1}. Based on the observed states of the AVUT and its environment, *Reward calculator* calculates a

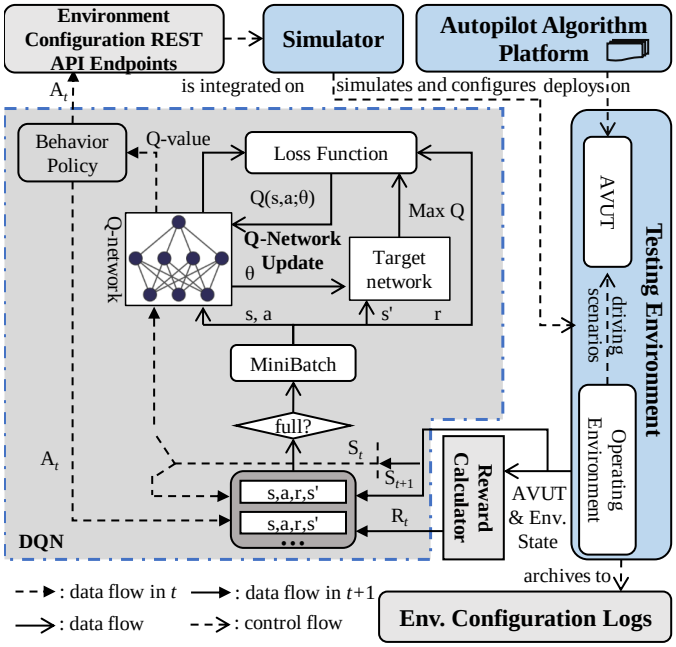


Fig. 1: Overview of DeepCollision

reward R_t for A_t and S_t at $t+1$. Then DQN stores them (as $< S_t, A_t, R_t, S_{t+1} >$) into the replay memory buffer. Once the replay memory is full, the Q-Network is updated as $Q(s, a; \theta)$, using the loss function by a mini-batch randomly sampled from the updated replay memory. Meanwhile, the target network parameters θ^- are updated with Q-Network parameters θ after a fixed number of steps, and remain unchanged between two updates. In addition, with S_{t+1}, the (updated) Q-network with behavior policy decides the next action: A_{t+1}. In DeepCollision, an episode is finished once the AVUT arrives at its destination or the AVUT is stuck and cannot move in one place for a specific duration. For such cases, we observed collisions among NPC vehicles that cause traffic jams, or collisions between the AVUT and obstacles. Note that in our current design, if the AUVT collides with obstacles, but still can continue its current driving task, we do not terminate an episode. At each time step t, the information about the AVUT (e.g., its driving and collision status) and its environment (e.g., its status and driving scenarios) are stored as *Environment Configuration Logs* for further analyses and collision replaying. More details about the DQN's Hyper-parameters setting used in DeepCollision are available in the replication package (see Section 5.6).

4.2 Configurable Environment Parameters

ADS functions are impacted by several physical phenomena and environment factors [7]. For example, a wet road affects the road friction influencing the AVUT's kinematics parameters like speed, braking distance, ultimately affecting the AVUT's behavior. The operating environment of an AVUT constantly changes, however, when testing ADSs in a simulated environment, the number of parameters that can be manipulated is limited to the simulator being used. Based on the simulator we used (i.e., LGSVL) and our analysis, we presents a taxonomy of configurable parameters of the operating environment of an ADS in Table 1.

TABLE 1: Configurable environment parameters*

Type	Parameter	Configuration
Weather	rain	None, Light, Moderate, Heavy
	fog	None, Light, Moderate, Heavy
	wetness	None, Light, Moderate, Heavy
Time of Day	timeOfDay	Morning, Noon, Night
Pedestrian	position	(x_0^p, y_0^p, z_0^p)
	rotation	(rx_0^p, ry_0^p, rz_0^p)
	speed	v_0^p: Real Number
	behavior	standing on the road, road crossing, walking on the road
NPC Vehicle	position	(x_0^n, y_0^n, z_0^n)
	rotation	(rx_0^n, ry_0^n, rz_0^n)
	speed	v_0^n: Real Number
	type	SUV, Sedan, Jeep, Hatchback, SchoolBus, BoxTruck
	behavior	lane maintaining, lane switching, overtaking, road crossing, parked

* All the REST API endpoints are provided in the replication package (see Section 5.6).

There are four types of environment parameters manipulated in the simulator. Some of them (e.g., Weather) are further classified into subtypes (e.g., rain). The first type is about weather conditions, which are further classified into three subtypes: rain, fog and wetness. For each of the weather types, we specify its level with the possible configurations in the third column. The second type of parameters is the time of day and we specify the possible configurations in the third column. Configuring the time of day impacts illumination conditions such as shadows, direct sunlight or over/under exposed, which can lead to poor performance of vision-based modules of an ADS [41], thereby important for testing.

The remaining two types of configurable parameters are used to generate pedestrians and Non-Player Character (NPC) vehicles in the environment. The pedestrian is characterized by four parameters which are its initial position along x, y and z axes (x_0^p, y_0^p, z_0^p), initial rotation along x, y and z axes (rx_0^p, ry_0^p, rz_0^p), initial speed v_0^p and its behavior. The NPC vehicle is characterized by five parameters: initial position (x_0^n, y_0^n, z_0^n), initial rotation (rx_0^n, ry_0^n, rz_0^n), initial speed v_0^n, type of the vehicle, and its behavior.

Parameters of *position*, *rotation* and *speed* are real numbers and they specify the initial status of pedestrians/NPC vehicles, while the *behavior* parameter characterizes behaviors of pedestrians/NPC vehicles during simulation. Specifically, the behavior of a pedestrian is classified into three common categories (*standing on the road*, *road crossing* and *walking on the road*), and the behavior of an NPC vehicle is classified into five common categories (*lane maintaining*, *lane switching*, *overtaking*, *road crossing* and *parked*). We classify NPC vehicle types into six categories: *SUV*, *Sedan*, *Jeep*, *Hatchback*, *SchoolBus* and *BoxTruck*. Note that these categories may have impacts on the size, speed and mobility of NPC vehicles. All in all, the selection of these parameters is mainly because their configurations have impact on the safety of autonomous vehicles.

Realistic Constraints. To ensure generated scenarios are realistic, in addition to ensure valid configurations of each

configurable parameter[1], we also specify a set of constraints on configurable parameters to ensure the realism of the generated scenarios. For example, the speed of a pedestrian should not exceed the average walking or running speed of a human people, and the initial speed of an NPC vehicle should not exceed the city road speed limit. Below, we give detailed constraints.

Specifying positions of pedestrians and NPC vehicles must be within boundaries of a map, on which the vehicle is driving. These positions must be also on the lane of the road to ensure not generating obstacles located in unrealistic places of the map (e.g., inside a building, or at top of a tree).

To ensure that an object is moving in the right direction, its rotation should be within a valid range. More specifically, an object's rotation is denoted as the angle between a unit vector and the vertical axis (the north direction) of the plane coordinate system, it has a positive value in the counter clockwise direction, with a maximum of $180°$ (π or 3.14rad), and a negative value in the clockwise direction, with a minimum of $-180°$ (-π or -3.14rad). As for an object's rotation, a valid rotation also needs to consider the road direction, implying that an obstacle cannot cross the road in the way that seldom happens in the real world.

As for speed, in our current design, we follow regulations from the National Highway Traffic Safety Administration [42]. They define, for instance, the speed of a sedan can range from 8km/h (parking lots) to 110km/h (e.g., highways), when driving on city roads, the speed of a sedan should follow the city road speed limit (e.g., no more than 60km/h in San Francisco). In addition, the speed of a pedestrian can range from 4.5km/h (average walking speed) to 10.5 km/h (average running speed).

Regarding weather and time of day, the realism is ensured with the constraint that: within one period of simulation time (e.g., 6s), the current environment setting should not be changed (e.g., from raining to sunny).

Environment Configuration REST API Endpoints. Eventually, we implemented the invocations of configurable environment parameters as REST API endpoints [43], which have been integrated with the simulator we used, i.e., LGSVL. Each REST API endpoint is associated with a simulation time, named *Observation Time Period* (OTP), OTP indicates the time we let the AVUT drive after invoking an REST API endpoint, and OTP is divided into several time steps with equal time duration *dt*. Additionally, an REST API endpoint has several outputs related to the effectiveness of the corresponding environment configuration, in our current design, we define the outputs as collision probabilities which will be introduced in Section 4.3.

The environment configuration can be handled via the REST API endpoints directly through HTTP requests. One example on how to use these APIs is provided in the replication package (see Section 5.6).

4.3 Calculating Collision Probability

DeepCollision measures the effectiveness of an environment configuration leading the AVUT to collisions as *collision probability*—a commonly used mechanism for testing autonomous vehicles [4]. A higher collision probability indicates a more effective environment configuration. We use collision probability to construct the reward function, which is calculated based on *Safety Distance* and *Current Distance* between the AVUT and obstacles in its environment.

4.3.1 Safety Distance

It is the minimum distance by which the AVUT avoids a collision with obstacles (e.g., NPC vehicles, pedestrians, and static obstacles) [44]. Note that this metric is commonly appli in the literature ([45], [46], [47]) to describe the status of the AVUT after braking, i.e., being safe if no collision is occurred after taking a braking decision. In this paper, we consider both longitudinal and lateral safety distances, which are commonly considered in the process of developing collision avoidance systems for autonomous vehicles [48], [49].

Longitudinal Safety Distance (LoSD) measures the safety distances of the AVUT with its following and leading vehicles driving on the same lane. LoSD has been used to measure safety distance for rear-end collision avoidance [44], which relies on a mathematical model to calculate safety distance. Particularly, in our current design of DeepCollision, we adapt the Berkeley algorithm [47], which has shown good accuracy on the safe distance prediction for rear-end collision [50], [51], as below:

$$LoSD(v_f, v_l) = \frac{1}{2}\Big(\frac{v_f^2}{\alpha_f} - \frac{v_l^2}{\alpha_l}\Big) + v_f\tau + R_{min} \qquad (7)$$

The Berkeley algorithm is a traditional kinematic approach, which calculates safety distances based on the worst case principle [47]. It assumes that initially both the leading and following vehicles maintain constant speeds v_f and v_l, respectively. Then the leading vehicle starts to brake at the maximum constant deceleration level α_l, while the following vehicle starts to brake after reaction time τ at the maximum constant deceleration level α_f. At the time the leading and following vehicles come to a full stop, they should maintain a minimum distance R_{min}, which is essentially a tolerance for the calculation error [44]. Note that the reaction time τ here accounts for driver's reaction time, and the algorithm assumes that the following vehicle's deceleration is zero during the reaction time τ.

In our current design, values of v_f and v_l can be obtained from the simulator at runtime via the DeepCollision REST APIs endpoints. α_f and α_l are the deceleration of the following and leading vehicles, which are treated uniformly by the Berkeley algorithm. We used the default value of $-6m/s^2$ in the Berkeley algorithm. τ is the driver's reaction time, which can be set τ to 0 as the AVUT is autonomous and its reaction can be considered as nearly instant [52], [53]. R_{min} is the minimum distance between two vehicles and the default value is 5 meters, same as in the Berkeley algorithm. Note that the AVUT itself can either be a following or leading vehicle, and when the behaviors of NPC vehicles are not static (i.e., with their speeds over zero), we use Equation 7 to calculate the safety distance. With Equation 7, we can also calculate the safety distance between AVUT and pedestrians walking in front of it. Note that as reported in [54], [55], the average pedestrian deceleration is about $-1.5m/s^2$.

[1]In our context, we consider that valid configurations are those allowed by the simulator being used. For instance, the speed of an NPC vehicle is allowed from 0 to 180km/h in LSGVL.

Pedestrians might be "standing on the road" or NPC vehicles might be "parked" (Table 1), i.e., being static. We consider static obstacles in the calculation of the safety distance when they are on the same lane with the AVUT and the collision may occur when they are leading the AVUT. Equation (8) simplifies Equation (7) for calculating the safety distance of the AVUT with every static obstacle by considering only the AVUT's speed because static obstacles have zero speed.

$$LoSD(v_f) = \frac{1}{2}\frac{v_f^2}{\alpha_f} + v_f\tau + R_{min} \tag{8}$$

where v_f, a_f, τ and R_{min} are the same as defined in Equation (7).

Lateral Safety Distance (LaSD) is the minimum lateral spacing between two neighbour vehicles under the condition that no collision occurs [56]. This safety distance is considered because lateral collisions with obstacles may occur due to turning, switching lanes, overtaking, etc. We treat lateral safety distances between the AVUT and NPC vehicles, pedestrians or static obstacles in the same way as shown in Equation (9) from [57].

$$LaSD(v_{avut}) = \frac{(v_{avut} \times \sin\beta)^2}{\alpha_{avut} \times \sin\beta} \tag{9}$$

where v_{avut} is the velocity of the AVUT, β is the angle between its direction and the lane where an obstacle is located, and α_{avut} is the deceleration of the AVUT. Values of v_{avut} and β are obtained from the simulator via DeepCollision REST API endpoints. α_{avut} is set as $-6m/s^2$ same as the default setting in the Berkeley algorithm.

4.3.2 Current Distance

Corresponding to the two safety distances, we use the Euclidean distance (calculated by Equation 10) to estimate the *current longitudinal distance (CLoD)* and *current lateral distance (CLaD)* between the AVUT with position $(x_{avut}, y_{avut}, z_{avut})$ and an obstacle with position (x_{ob}, y_{ob}, z_{ob}). The Euclidean distance is commonly used in vehicle tracking and localization [34] and it has also shown good accuracy when estimating the distance between AVUT and obstacles in the context of testing ADSs [7], [9].

$$CD = \sqrt{(x_{avut} - x_{ob})^2 + (y_{avut} - y_{ob})^2 + (z_{avut} - z_{ob})^2} \tag{10}$$

where CD denotes CLoD or CLaD.

4.3.3 Collision Probability

Equation (11) calculates the collision probability ($ProC$) between the AVUT and obstacles. Based on the safety distances and current distances, we calculate the *longitudinal collision probability (LoProC)* and *lateral collision probability (LaProC)* separately (Equation 11). $ProC$ ranges from 0.0 to 1.0, where a value of 1.0 indicates an unavoidable collision and the value of 0.0 means no chance for collision.

$$ProC = \begin{cases} \dfrac{SD - CD}{SD}, & \text{CD <SD} \\ 0.0, & \text{else} \end{cases} \tag{11}$$

where ProC is LoProC or LaProC, $SD \in \{LaSD, LoSD\}$, and $CD \in \{CLaD, CLoD\}$.

We would like to emphasize that all environment configuration actions conform to real traffic driving, implying that at the moment when an environment configuration action is executed, the vehicle maintains safety distances with all obstacles in the configured driving environment. These safety distances may be violated when the vehicle drives in the configurable environment, which consequently results in collisions. Therefore, we need to calculate the collision probability in each time step after invoking an environment configuration REST API endpoint.

DeepCollision allows the AVUT drive for a fixed period of time, named *Observation Time Period* (OTP), after one environment configuration API is invoked to change the environment (Section 4.2). Each OTP (e.g., 4 seconds) is divided into several time steps *dt*. At each time step *dt*, DeepCollision calculates $ProC$ between the AVUT and all obstacles and obtains LoProCs and LaProCs values with all of them. Then, DeepCollision chooses the maximum $LoProC_{dt}$ and $LaProC_{dt}$ to denote the LoProC and LaProC at time step *dt*, respectively. $LoProC_{dt}$ and $LaProC_{dt}$ are further weighted together to obtain the final probability at time step *dt*: $ProC_{dt}$. Denoting $max(LoProC_{dt}, LaProC_{dt})$ as $ProC_{dtmax}$ and $min(LoProC_{dt}, LaProC_{dt})$ as $ProC_{dtmin}$, the weighting formula is:

$$ProC_{dt} = ProC_{dtmax} + (1 - ProC_{dtmax}) \times ProC_{dtmin} \tag{12}$$

where $ProC_{dt}$ falls into [$ProC_{dtmax}$, 1].

Then, we compute the maximum collision probability during each OTP: obtaining all $ProC_{dt}$ values and select the maximum as the final $ProC$ to assess the effectiveness of the corresponding environment configuration action. DeepCollision sets time step *dt* to 0.5s. Since the OTP is a crucial factor, we control it as an independent variable in our empirical study (Section 5.1).

4.4 Formulating Environment Configurations Learning as an MDP

We present formulation of the environment configuration problem as an MDP, which can be defined with a 5-tuple $< \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, \gamma >$. We adopt Deep Q-Learning (DQN) as the RL algorithm and we adopt the following state, action and reward formulation such that the problem can be solved with DQN.

4.4.1 State

To successfully apply RL to generate critical scenarios for autonomous vehicles, an appropriate state encoding is vital. To apply RL, DeepCollision needs to extract features of AVUT and its operating environment to encode a state. A state may include the AVUT and obstacles, their locations, lane positions, states of traffic lights, pedestrians, etc. Leurent [58] has provided a comprehensive review of existing state encoding approaches for autonomous driving, among which a commonly used approach relies on variables of the autonomous vehicle and its operating environment.

Similarly, DeepCollision encodes AVUT and its environment states for DQN with a set of variables listed in Table 2. A state is defined as a state-tuple $(w_1, w_2, ...w_i)$, where w_i is the value of a state variable v_i. Based on the observed state, the agent of DeepCollision determines which environment

configuration parameters to configure and what values to assign (i.e., action A_t in Figure 1). To experiment different settings, we developed $DeepCollision_8$, $DeepCollision_{10}$, and $DeepCollision_{12}$, conducted a pilot study (Section 5.2) to select the best one for the formal experiment (Section 5.3).

TABLE 2: Parameters defining DQN states in DeepCollision*

#	Type	Parameter	Description
1	Weather	rain	Identify the level of rain
2		fog	Identify the level of fog
3		wetness	Identify the level of wetness
4	Time of Day	timeOfDay	Identify time in one day
5	Traffic Light	trafficLightColor	Identify the color of the traffic light ahead of the AVUT. Set value 0 if no traffic light.
6	AVUT Position	positionX	Locate the AVUT's position on X-axis
7		positionY	Locate the AVUT's position on Y-axis
8		positionZ	Locate the AVUT's position on Z-axis
9	AVUT Speed	speed	Identify AVUT's speed
10	AVUT Rotation	rotationX	Identify the road slope on X-axis
11		rotationY	Identify the AVUT driving direction
12		rotationZ	Identify the road slope on Z-axis

* $DeepCollision_i$: the DeepCollision model using parameters from $\#1$ to $\#i$, and i $\in \{8, 10, 12\}$.

4.4.2 Action

As discussed in Section 4.2, the environment parameters manipulated in the simulator (i.e., LGSVL) are of four types (Table 1), some of which (e.g., weather) are further classified into subtypes (e.g., rain). The third column of the table shows possible configurations of each parameter. Invocations of the environment parameters are realized via REST API endpoints, and we implemented 52 REST API endpoints, which form the *action space* in DQN. At each step, the DQN agent selects an action A_t from the *action space* and gets corresponding APIs to be invoked.

One key challenge in RL is about balancing the trade-off between exploration and exploitation [4]. We adopt the ϵ-greedy policy (Equation (13)) to select an action at each time step t based on the principle of exploration versus exploitation, commonly used in RL algorithms [59]. DeepCollision exploits the agent's current action-value estimates to select current best actions with the highest Q-value with probability 1-ϵ and explores the action space to randomly select an action with probability ϵ.

$$A_t = \begin{cases} arg\max_a Q(s,a), & 1-\epsilon \\ \text{random select an action}, & \epsilon \end{cases} \tag{13}$$

Notice that ϵ is an important Hyperparameter in DQN as its value determines the probability of random exploration, a proper value of ϵ can help to get the optimal solution more quickly. Intuitively, at the beginning of the training process, the agent should explore more since it knows little about the operating environment. As the training progresses, the agent will gain knowledge about the environment and may gradually conduct more exploitation than exploration. Being aware of that, we follow the ϵ setting from Mnih et al. [15] and let ϵ anneal linearly from 1.0 to 0.1 at the first 6000 observations, and fix at 0.1 thereafter.

4.4.3 Reward

After an environment configuration action A_t being invoked on S_t, both the AVUT and its operating environment will move to a new state S_{t+1}, then the DQN agent will receive a immediate reward R_t associated with the transition $<S_t, A_t, S_{t+1}>$. The reward is used to evaluate the environment configuration action, and guide the DQN agent to find collision-prone configurations. In DeepCollision, the reward function is constructed based on *collision probability*, and the action with higher collision probability will lead to larger reward. Specifically, as shown in Equation (14), the reward function value is assigned to -1.0, i.e., punishment, when the calculated collision probability is less than the predefined threshold value, it means the agent do not expect such environment configuration actions since they contribute less to collision. Otherwise, the value is assigned as the calculated collision probability, i.e., [threshold, 1.0]. Note that a value of 1.0 indicates that a collision happened (i.e., collision probability equals to 1.0), whereas a lower value implies that no collision occurred. Based on the results of the pilot study (Section 5.2), we set the threshold value to be 0.2 as we observed that when the model converges the reward is always greater than 0.2.

$$R_t = \begin{cases} -1.0, & 0 \le ProC < threshold \\ ProC, & threshold \le ProC < 1 \\ 1.0, & \text{collision occurred} \end{cases} \tag{14}$$

5 Empirical Evaluation

Section 5.1 presents the experimental setup. Section 5.2 describes the pilot study. The formal experiment is presented in Section 5.3, followed by results in Section 5.4 and the threats to validity in Section 5.5.

5.1 Subject System, Simulator, and AVUT

Subject System. We adopt Baidu's Apollo as the subject system under test, which is an open-sourced autopilot algorithm platform with a flexible architecture to support software in loop and hardware in loop testing. According to the work by Peng [39], Apollo is a complex ADS with a large number of ML models. These models are responsible for various tasks such as traffic light recognition, lane detection and maintaining, obstacle perception and avoidance, and trajectory prediction. With this platform, it is expected to rapidly develop and validate autonomous vehicles. Based on the definitions of the six levels of intelligence in a vehicle by the Society of Automotive Engineers (SAE) [60], Apollo aims to address the industrial Level 4: *High Driving Automation*, i.e., not requiring human interaction in most circumstances but providing an option for manual overriding of the control [61]. According to SAE, the highest level is Level 5: *Full Driving Automation*, which is defined as not requiring human attention at all.

Simulator. LGSVL Simulator is a Unity-based and open-sourced simulator for autonomous driving. Empowered by the high definition render pipeline technology of Unity, LGSVL Simulator can be used to develop realistic environment simulations. LGSVL Simulator is equipped with a rich list of sensors such as camera, LiDAR, GPS and

Radar, to enable the status observation of the autonomous vehicle and environment. To simulate vehicle dynamic and control, LGSVL has a vehicle dynamics model. Importantly, LGSVL can be integrated with various autopilot algorithm platforms, including Apollo. With the bridge interfaces of LGSVL, a little effort is needed to connect autopilot algorithm platforms with LGSVL.

AVUT. In this experiment, we used the Apollo Open Platform 5.0 [19] (*Autopilot Algorithm Platform*) as the subject system and employed the LGSVL Simulator 5.0 for simulating the driving environment (Figure 1). Specifically, we used the San Francisco map in LGSVL, which is a digital re-creation of a section of the South of Market Street (SoMa) in San Francisco. This section has interesting driving environment characteristics, e.g., many traffic light intersections and multi-lane streets. The AVUT controlled by Apollo is Lincoln2017MKZ, a four door Sedan car, which supports a variety of sensor configurations for different autonomous driving systems, and is often used by autonomous driving development groups [62], [63].

5.2 Pilot Study

We conducted a pilot study to understand the problem and potential solutions for enabling the better design of the formal experiment. We used two roads and three different variable-configurations of DQN states for DeepCollision models (i.e., 8, 10 and 12 DQN state parameters, see Table 2), and a random strategy (randomly picking an action from the action space) was used as the comparison baseline. More details about the pilot study are available in the replication package (see Section 5.6).

We trained three DeepCollision models for 600 episodes for each on the San Francisco map [20], denoted as $DeepCollision_8$, $DeepCollision_{10}$, and $DeepCollision_{12}$, respectively. Then, all the models and the random strategy were used to change the environment of the AVUT driving on each road, for 30 runs. We consequently obtained 8 datasets: 6 for the three trained models and 2 for random.

TABLE 3: Results of the Pilot Study*

Road		Method	Pr_C	$\#C'$	DT(s)
R_a	Model	$DeepCollision_8$	28%	1.67	47.33
		$DeepCollision_{10}$	26%	1.27	**41**
		$DeepCollision_{12}$	**37%**	**2.3**	49.83
	Random		12%	0.43	49.5
R_b	Model	$DeepCollision_8$	28%	1.97	59.17
		$DeepCollision_{10}$	26%	1.97	65.67
		$DeepCollision_{12}$	**33%**	**2.37**	70.5
	Random		8%	0.5	**49**

* Pr_C: the average collision probability; $\#C'$: the average number of collisions; DT: average driving time.

We studied the performance of DeepCollision with: the average number of collisions, the average collision probability and the average running time per run. Results (Table 3) show that $DeepCollision_{12}$ achieved the best performance in terms of all the metrics except for the average running time. Regarding the average driving time per run, though $DeepCollision_{12}$ spent more time than $DeepCollision_8$ and $DeepCollision_{10}$, it still took less than one minute on average. Moreover, the differences with $DeepCollision_{10}$ (performed the best for road R_a) and $DeepCollision_8$ (outperformed the others for road R_b) are less than 12 seconds, which is practically negligible. Therefore, in the formal experiment, we opted for the method of employing the 12 parameters for the DQN state representation.

5.3 Formal Experiment

We present comparison baselines, research questions, experiment design and execution, evaluation metrics, and statistical test in Sections 5.3.2-5.3.5.

5.3.1 Comparison Baselines

To evaluate DeepCollision, we implemented two comparison baselines: a random strategy (named RS) and a greedy strategy (named GS). RS is commonly used as the baseline for sanity check on whether an optimization problem is complex enough. GS is commonly used as the comparison baseline for assessing a newly proposed RL-based approach for RL-based decision making for autonomous lane changing [64], RL-based navigation optimization [65], etc.

Random Strategy (RS, Algorithm 1). RS randomly selects an action to configure the environment at each OTP.

Algorithm 1 RS-based Environment Configuration

Input
Let *Env* be the test environment;
A be the environment configuration action space;
done be the stopping criteria
Output
Act_{buffer}: a buffer of actions in A

1: $Act_{buffer} \leftarrow []$
2: **while** $\neg$ *done* **do**
3: $act \leftarrow$ random select an action from A
4: *done*, *prob*, *obs* $\leftarrow$ step(act)
5: Put act into Act_{buffer}
6: **end while**
7: **return** Act_{buffer}

Greedy Strategy (GS, Algorithm 2). To determine the next action at a given OTP, GS executes all the actions (Lines 7–14), then selects an action that achieves the best performance in terms of collision probability (Line 15). To enable a fair comparison of the performance of all the actions of the action space in one OTP, we implemented a rollback of environment configurations, which allows a reset of the environment to a given configuration (see Line 13). In the end, a sequence of best actions is returned and executed.

5.3.2 Research Questions (RQs)

RQ1: Compared with the two baselines (RS and GS, Section 5.3.1), how is the overall performance of DeepCollision? RQ1 is necessary since it tells whether we address a complex problem, which is hard to be solved efficiently with RS and GS. **RQ2**: How is the performance of the DeepCollision models (trained for each OTP, see Section 5.3.3) when compared to each other? RQ2 helps to find the most suitable OTP setting to configure DeepCollision based on different road structures. Recall that OTP indicates the time that the AVUT drives after invoking an REST API endpoint to

Algorithm 2 GS-based Environment Configuration

Input
Let *Env* be the test environment;
A be the environment configuration action space;
done be the stopping criteria.
Output
Act_{buffer}: a buffer of actions in A

$Act_{buffer} \leftarrow []$
obs $\leftarrow$ observe a state of *Env*
$Env_{pre} \leftarrow$ SaveEnv(*Env*, *obs*) ▷ save the *Env* which can observe state *obs*
while $\neg$ *done* **do**
  $prob_{max} \leftarrow 0$
  $act_{with-max-prob} \leftarrow 0$
  for each action *a* in A **do** ▷ try each action in action space
    _, $prob$, _ $\leftarrow$ step(*a*)
    if $prob > prob_{max}$ **then**
      $prob_{max} \leftarrow prob$
      $act_{with-max-prob} \leftarrow a$
    end if
    Env $\leftarrow$ RollBackEnv(Env_{pre})
  end for
  $act \leftarrow act_{with-max-prob}$ ▷ let act be the action with maximum collision probability
  done, $prob$, *obs'* $\leftarrow$ step(act)
  $Env_{pre} \leftarrow$ SaveEnv(*Env*, *obs'*)
  Put act into Act_{buffer}
end while
return Act_{buffer}

change the environment. Configuring DeepCollision with different OTP values (e.g., 8s) has impact on the realism of generated scenarios and the time performance of DeepCollision. For instance, a short OTP (e.g., 4s) might generate unrealistic scenarios for complex road conditions, whereas a model with a long OTP (e.g., 10s) might take an unnecessarily long time to generate scenarios.

5.3.3 Experiment Design and Execution

DeepCollision Models. To answer the RQs, we trained four DeepCollision models. Each corresponds to one OTP: 4s (M4), 6s (M6), 8s (M8), 10s (M10). Accordingly, we defined 4 RS and 4 GS strategies. Each DeepCollision, RS and GS strategy was run 30 times (i.e., $k = 1...30$) on each of the four roads (R1...R4). In total, we obtained results of 1440 executions (30 runs $\times$ 4 roads $\times$ 3 approaches (DeepCollision, RS and GS) $\times$ 4 OTP strategies). For each execution, we collected statistics every 0.5s which is sufficiently small as the minimum OTP we experiment with is 4s and therefore sampling with 0.5s is fair for all the OTPs.

Network Architecture and Parameter Settings. As discussed in Section 2, DQN utilizes a DNN to approximate the Q function. In the context of autonomous driving, a good network architecture is important for learning tasks such as motion planning [13] and end-to-end autonomous driving [14]. In DeepCollision, the input of the network is a state-tuple $(w_1, w_2, ...w_{12})$ of the 12 parameters, which encode the state of the AVUT and environment (Section 4.4.1). The output of the network is a set of Q-values for all the actions. The action with the maximum Q-value is selected to be executed. We followed the neural network design guidance from Liu et. al [66] and the network architecture proposed by Zhu et. al [67] to design the DNN for DeepCollision. Specifically, we used a four-layer fully connected DNN with the following architecture: an input layer with 12 neurons corresponding to the 12 state parameters, one output layer with 52 neurons corresponding to the 52 environment configuration actions, and 2 hidden layers with 200 neurons each. For these two hidden layers, the Rectified Linear Unit (ReLU) activation was applied to their neurons to accelerate the convergence of the network parameter optimization.

As for parameters' settings when training the models, we adopted default hyperparameter settings of DQN based on the tutorial by Zhou [68] and applied default settings of behavior policy (i.e.,ϵ-greedy) based on the work by Mnih et al. [15]. All the models were trained with the same network architecture, learning algorithm, reward design, and hyperparameter settings, in order to make fair comparisons among the different models. More details about the network architecture and parameter settings of the DQN in DeepCollision are available in the replication package (see Section 5.6).

Driving Scenarios. A driving scenario is a temporal development of the environment in which the AVUT operates [69]. Based on the definition, we define a driving scenario as a 4-tuple <AVUT Operation, AVUT Speed, Environment State, Time>, where: 1) *AVUT Operation* denotes AVUT operations such as cruise; 2) *AVUT Speed* indicates a speed level of the AVUT, e.g., zero or slow; 3) *Environment State* is about weather condition, time of day, traffic rules, obstacles and their behaviors, etc; and 4) *Time* is the time period that the scenario spans. More specifically, we used 11 properties (column 3, Table 4) to characterize AVUT's states (rows 1 and 2, Table 4), the environment (rows 3 to 11, Table 4). Note that *NPC vehicles*, *Pedestrians*, *Static obstacles*, *Traffic Lights* and *Sidewalks* are collected within the sensing range of the radar deployed on the AVUT.

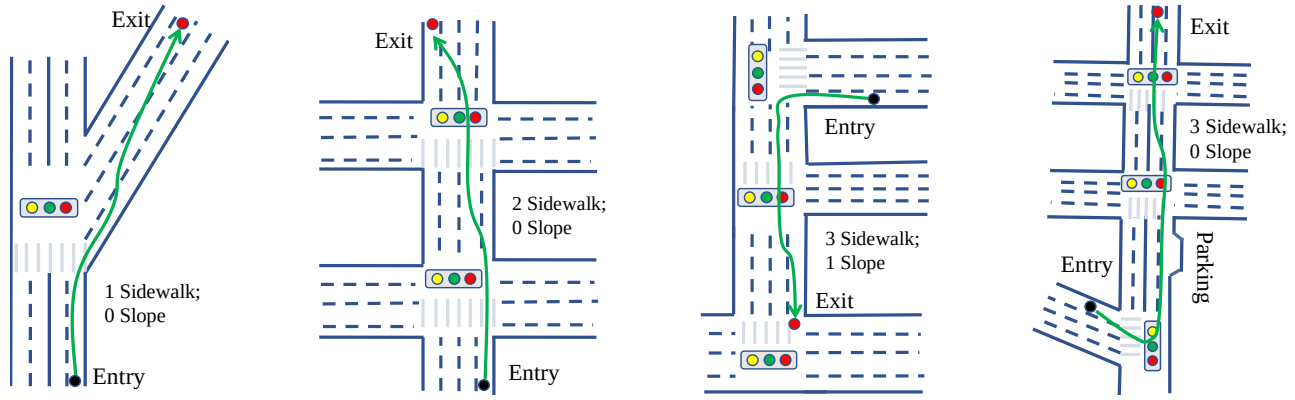
Roads. The environment configuration process of DeepCollision spans over a time period of driving of an AVUT. Therefore, in our experiment, we select different roads for the purpose of evaluating DeepCollision. Each road has an origin (Entry) and a destination (Exit), and is featured with various scenes and road structures. Czarnecki [70] proposed the *Operational World Model* (OWM) ontology for road structures. This ontology defines road structures such as their lane and roadside structures, traffic signs and signals, crosswalks, and intersections.

Based on this road structure definition, we selected four different roads. Figure 2 depicts their characteristics. On ***Road R1***, the AVUT (ω) first drives on a dual-way road with one traffic light and one sidewalk, then turns right to a one-way road of four lanes until the exit. With this setting, when driving on the dual-way road, ω can only drive on the two lanes without crossing the yellow line, and before turns right, it needs to switch to the right-turn lane. On ***Road R2***, ω drives on a straight one-way road with four lanes, two traffic lights, two sidewalks, and two intersections. Though there are no turns, ω needs to switch three lanes to reach

TABLE 4: Properties characterizing scenarios in the formal experiment*

#	Property		Property Value
1	AVUT	Operation	Stop; Cruise; SpeedUp; SpeedCut; EmergencyBrake; SwitchLaneToRight; SwitchLaneToLeft; TurnLeft; TurnRight
2		Speed (m/s)	Zero (0 <speed $\leq$ 1); Slow (1 <speed $\leq$ 5); Moderate (5 <speed $\leq$ 12); Fast (speed >12)
3	Env	Weather: rain	None ($rain_{level}$ = 0); Light (0 <$rain_{level}$ $\leq$ 0.2); Moderate (0.2 <$rain_{level}$ $\leq$ 0.5); Heavy ($rain_{level}$ > 0.5)
4		Weather: fog	None (fog_{level} = 0); Light (0 <fog_{level} $\leq$ 0.2); Moderate (0.2 <fog_{level} $\leq$ 0.5); Heavy (fog_{level} > 0.5)
5		Weather: wetness	None ($wetness_{level}$ = 0); Light (0 <$wetness_{level}$ $\leq$ 0.2); Moderate (0.2 <$wetness_{level}$ $\leq$ 0.5); Heavy ($wetness_{level}$ > 0.5)
6		Time of the day	Morning (10am); Noon (12am); Night (8pm)
7		NPC vehicle*	*Volume* (m^3): Small (volume $\leq$ 10); Medium (10 < volume $\leq$ 60); Large (volume >60) *Operation*: Stop; SwitchLane; LeftLaneDriving; RightLaneDriving; CurrentLaneDriving *Speed (m/s)*: Zero (0 <speed $\leq$ 1); Slow (1 <speed $\leq$ 5); Moderate (5 <Speed $\leq$ 12); Fast (speed >12) *Distance from AVUT (m)*: Zero ($\|d\|$ = 0); Very near (0 <$\|d\|$ $\leq$ 8); Near (8 <$\|d\|$ $\leq$ 18); Far (18 <$\|d\|$ $\leq$ 28); Very far ($\|d\|$ >28)
8		Pedestrian*	*Volume* (m^3): Small (volume <1) *Operation*: Stop; Crossing the road; FrontLaneWalking *Distance from AVUT (m)*: Zero ($\|d\|$ = 0); Very near (0 <$\|d\|$ $\leq$ 8); Near (8 <$\|d\|$ $\leq$ 18); Far (18 <$\|d\|$ $\leq$ 28); Very far ($\|d\|$ >28)
9		Static obstacles*	*Volume* (m^3): Small (volume $\leq$ 3); Medium (3 < volume $\leq$ 10); Large (volume >10) *Distance from AVUT (m)*: Zero ($\|d\|$ = 0); Very near (0 <$\|d\|$ $\leq$ 8); Near (8 <$\|d\|$ $\leq$ 18); Far (18 <$\|d\|$ $\leq$ 28); Very far ($\|d\|$ >28)
10		Traffic Rule: traffic light	None; Green (allow to pass but slow at intersection); Yellow (stop for a while); Red (do not run a red light)
11		Traffic Rule: side walk	None; SlowDown

* Env denotes Environment; An NPC vehicle is defined with its *Volume*, *Operation*, *Speed*, and *Distance from AVUT*, similarly for pedestrians and static obstacles; Sign $*$ denotes that there might exist 0 to many NPC vehicles, pedestrians and static obstacles; Property values of speed and volume are from the National Highway Traffic Safety Administration, NHTSA [42].



(a) R1 (0.7km) (b) R2 (1.1km) (c) R3 (1.0km) (d) R4 (1.5km)

Fig. 2: Graphical representations of the selected roads

the exit point. On ***Road R3***, ω drives on a one-way road with one left turning, two traffic lights, two sidewalks, two intersections, one dedicated left-turn lane and three straight lanes. To reach the exit, ω first needs to perform a left turn to switch to a straight downhill lane, then drives on one of the three straight lanes without turning. On ***Road R4***, ω first drives on a one-way road with one left turn, one traffic light and one sidewalk, then turns left to drive on a dual-way road with two traffic lights, two sidewalks and two intersections. To reach the exit located on the right-hand side after left turning, ω first needs to switch to the left-turn lane, then turns left and drives on the right-hand side of the road until reaching the exit. Note that ω is expected to follow traffic rules and take actions such as lane-switch to avoid collisions at any time.

5.3.4 Evaluation Metrics

Collision Metrics: $C^{r,k} = \{c_i^{r,k}, pr_i^{r,k} | i = 0..n\}$: A potential collision is the one in which the distance between the AVUT and its closest obstacle is less than the safe distance, and its probability ($pr_i^{r,k}$) of triggering a collision would be more than 0.0, i.e., $\in (0.0, 1.0]$. Note that $pr_i^{r,k} = 1.0$ indicates that the AVUT collides with an obstacle. We also distinguish collisions occurred from potential collisions as $C'^{r,k}$, i.e., a subset of $C^{r,k}$ that the AVUT collides with an obstacle with the collision with 100% probability; ***Scenario Metrics***: A subset of generated scenarios led to potential collisions $S_C^{r,k}$ and collision occurrences $S_{C'}^{r,k}$; ***Time Performance***: *Time to the first collision occurrence* $Time_{1stC'}^{r,k}$; *Time interval between two occurred collisions*: $TI^{r,k}$; and *Average time interval* $TI_{avg}^{r,k} = \frac{\sum_{c=1}^{\#C'} ti_c^{r,k}}{\#C'}$.

Scenario's Demand on ADS: When facing a driving task, depending on scenario complexity and environment factors, different scenarios may put different driving challenges to road users such as ADSs and NPC vehicles [71]. To describe the challenge that a scenario puts on an ADS, Czarnecki [72] proposed a concept, named *demand*, to represent the challenge for an ADS to handle the driving scenario. More specifically, in Czarnecki's work [72], scenarios are classified into various demand levels (e.g., low, medium and high) corresponding to different degrees of challenges. Normally,

a higher demand scenario would make it more difficult for ADS to handle. Inspired by this concept, we propose metric *demand* to measure the extent of the challenge in which the AVUT is demanded to deal with, given a generated scenario, based on the 11 scenario properties presented in Table 4. Concretely, each scenario property has a *demand value*. For example, considering that when facing the same or similar environment, a higher speed of the AVUT would possibly result in a higher demand, hence we consider the AVUT speed when calculating the demand. The speed can take demand values going up from 0 (for *Zero*), 1 (for *Slow*), 2 (for *Moderate*), to 3 (for *Fast*). Similarly, the demand value of *rain* can be 0, 1, 2, and 3, corresponding to no rain, light, moderate or heavy rain. Consequently, we consider properties with demand values equal or greater than its medium value, i.e., $(Max_D - Min_D)/2$ as properties with high demand values, where, Max_D and Min_D denote the maximum and minimum demands of one property. For instance, for property *rain* (i.e., medium=$(3-0)/2$), *Moderate* (2) or *Heavy* (3) rain can be considered as high demand. Considering the 11 scenario properties, ***High Demand Scenarios*** are identified when there exist no less than half of the properties (i.e., ≥ 5=$\lfloor 11/2 \rfloor$) whose demand values are high, denoted as $S^{r,k}_{HighD}$. In DeepCollision, we aim to find high demand scenarios.

Average Improvement (AIP): To measure relative improvements when comparing different strategies, we calculate *AIP*, for each metric, as below:

$$AIP_\theta = \frac{Model_\theta - Baseline_\theta}{Baseline_\theta} \times 100\%$$

where, $\theta \in \{\#C, \#C', Pr_C, \#S_C, \#S_{C'}, \#S_{HighD}, \#S, Time_{1stC'}, TI_{avg}\}$. Note that $Baseline \in$ {RS, GS}, and $(Baseline_\theta - Model_\theta)$ is used for $Time_{1stC'}$ and TI_{avg}.

5.3.5 Statistical Tests

We followed the guidelines proposed by Arcuri and Briand [73] to select statistical tests and chose the significance level of 0.05 for all the tests. First, we checked if our samples are normally distributed with the Kolmogorov-Smirnov test [74]. The test results show that they are not normally distributed. We then used the Kruskal-Wallis rank test (non-parametric) to compare multiple samples, which reveals a significant difference among them. Thus, we further performed pairwise comparisons with the Mann–Whitney U test [75], and the Vargha and Delaney effect size [76] to calculate $\hat{A}_{12}$. Given an evaluation metric χ, $\hat{A}_{12}$ was used to compare the probability of yielding higher values of the metric χ for two strategies A and B. If $\hat{A}_{12}$ is 0.5, then they are equal. If $\hat{A}_{12}$ is greater than 0.5, then A has a higher chance to obtain higher values of χ than B, and vice versa. The Mann–Whitney U test calculates p-values to determine if the performance difference between A and B is significant. A p-value less than 0.05 indicates the significant difference between A and B. By following the guidelines proposed by Kitchenham et al. [77], we divide the Vargha and Delaney effect size magnitude of $\hat{A}_{12}$ into four levels: *negligible* ($\hat{A}_{12} \in$ (0.444, 0.556)), *small* ($\hat{A}_{12} \in$ [0.556, 0.638) or (0.362, 0.444]), *medium* ($\hat{A}_{12} \in$ [0.638, 0.714) or (0.286, 0.362]), *large* ($\hat{A}_{12} \in$ [0.714, 1.0] or [0, 0.286]).

5.4 Results and Analysis

In this section, we summarize key results of statistical tests. The replication package is listed in Section 5.6.

5.4.1 Results for RQ1

To compare the performance of DeepCollision with the two baselines: RS and GS, we employed all results of all the OTP configurations (i.e., 30 runs $\times$ 4 OTPs = 120 samples) to perform statistical tests to assess the *overall performance* of each approach on each road. Results are summarized in Table 5, where, to save space, we use M to denote DeepCollision, and [Road]-[Strategy] (e.g., R1-GS) refers to results achieved by a strategy on the road.

Number of potential collisions ($\#C$). As shown in Table 5, for all of the four roads, DeepCollision significantly outperformed the two baselines. This is demonstrated by results of significant differences (p-values) and effect sizes ($\hat{A}_{12}$): p-value < 0.05 for all the cases; $\hat{A}_{12}$ is at least at the *medium* magnitude level (i.e., ≥ 0.638) for all the cases; and $\hat{A}_{12}$ is at the *large* (i.e., ≥ 0.714) magnitude level in 5 out of 8 cases. Regarding *AIP* of $\#C$, DeepCollision achieved at least 44% (109%) improvement over RS (GS) on *R3* (*R4*).

Number of collisions occurred ($\#C'$). For this metric, DeepCollision significantly outperformed RS and GS on all the roads, and the $\hat{A}_{12}$ magnitude is *large* (i.e., ≥ 0.714), except for R1-RS with which the *medium* level results were obtained. Regarding the results of *AIP*, DeepCollision achieved 67% - 280% improvements over RS and GS.

Collision probability (Pr_C). In terms of Pr_C, DeepCollision significantly outperformed RS on all the roads with the effect size of $\hat{A}_{12}$ being all at the level of *large*. For the *AIP*, DeepCollision achieved improvements from 124% to 191% over RS. When comparing with GS, DeepCollision significantly outperformed GS on *R2* and *R4* with the *large* and *small* $\hat{A}_{12}$ magnitude values, respectively. For *R1* and *R3*, DeepCollision showed a comparable performance with GS, as evidenced with p-value being 0.16 and $\hat{A}_{12}$ being 0.448. Notice that GS only optimizes collision probability at each step (Section 5.3.1) locally; therefore it has limited potential for optimizing $\#C$ and $\#C'$.

Number of scenarios leading to potential collisions ($\#S_C$). In terms of $\#S_C$, DeepCollision significantly outperformed GS and RS on all the four roads. The $\hat{A}_{12}$ values are all at the level of *large* when comparing with GS. When comparing with RS, the $\hat{A}_{12}$ values are either at the *large* (on *R1* and *R2*) or *medium* (on *R3* and *R4*) levels. Regarding *AIP*, DeepCollision improved the performance of RS and GS by 25% - 408%.

Number of scenarios leading to collisions ($\#S_{C'}$). In terms of $\#S_{C'}$, DeepCollision significantly outperformed GS with a medium effect size on *R1*, and a large effect size on *R2* and *R4*. Results of *AIP* range from 100% to 300%, i.e., one to three times of improvements over the baselines. DeepCollision significantly outperformed RS on *R2*, *R3* and *R4* with the effect sizes being at the *large* level and AIP values ranging from 173% to 265%. For *R1*, no significant difference can be observed between DeepCollision and RS. After further investigation, we observed cases where one scenario led to multiple collisions, implying that DeepCollision achieved significant higher $\#C'$ with one scenario than

TABLE 5: Results of the Vargha and Delaney statistics and the Mann–Whitney U test— RQ1*

Road		Collision (M vs. GS/RS)			Scenario (M vs. GS/RS)				Time (GS/RS vs. M)	
		$\#C$	$\#C'$	Pr_C	$\#S_C$	$\#S_{C'}$	$\#S_{HighD}$	$\#S$	$Time_{1stC'}$	TI_{avg}
		$\hat{A}_{12}$/p/AIP%	$\hat{A}_{12}$/p/AIP%	$\hat{A}_{12}$/p/AIP%	$\hat{A}_{12}$/p/AIP%	$\hat{A}_{12}$/p/AIP%	$\hat{A}_{12}$/p/AIP%	$\hat{A}_{12}$/p/AIP%	$\hat{A}_{12}$/p/AIP%	$\hat{A}_{12}$/p/AIP%
R1	GS	**.696**/**<.05**/260	**.835**/**<.05**/100	.451/.19/-18	**.817**/**<.05**/171	**.640**/**<.05**/100	**.798**/**<.05**/68	**.689**/**<.05**/43	**.793**/**<.05**/73	**.683**/**<.05**/96
	RS	**.797**/**<.05**/179	**.713**/**<.05**/67	**.907**/**<.05**/187	**.789**/**<.05**/108	.548/.180/9	**.999**/**<.05**/794	.386×/<.05/-21	**.768**/**<.05**/84	**.645**/**<.05**/96
R2	GS	**.897**/**<.05**/411	**.798**/**<.05**/100	**.756**/**<.05**/100	**.905**/**<.05**/408	**.852**/**<.05**/300	**.755**/**<.05**/55	**.730**/**<.05**/71	.504/.907/88	.452/.154/-91
	RS	**.728**/**<.05**/66	**.747**/**<.05**/170	**.822**/**<.05**/124	**.756**/**<.05**/96	**.769**/**<.05**/173	**.999**/**<.05**/986	**.634**/**<.05**/24	.444/.122/14	.525/.431/0
R3	GS	**.784**/**<.05**/109	**.736**/**<.05**/100	.448/.16/-18	**.752**/**<.05**/127	**.849**/**<.05**/200	**.841**/**<.05**/45	**.717**/**<.05**/26	.333×/<.05/-65	**.599**/**<.05**/96
	RS	**.656**/**<.05**/44	**.848**/**<.05**/234	**.909**/**<.05**/170	**.677**/**<.05**/25	**.803**/ **<.05**/212	**.996**/**<.05**/783	.398×/<.05/-11	**.816**/**<.05**/53	**.726**/**<.05**/98
R4	GS	**.789**/**<.05**/429	**.772**/**<.05**/133	**.575**/**<.05**/26	**.829**/**<.05**/350	**.823**/**<.05**/233	**.749**/**<.05**/52	**.802**/**<.05**/61	**.866**/**<.05**/96	.432/.067/-28
	RS	**.638**/**<.05**/52	**.853**/**<.05**/280	**.872**/**<.05**/191	**.703**/**<.05**/88	**.849**/ **<.05**/265	**.994**/**<.05**/819	**.613**/**<.05**/21	**.824**/**<.05**/96	**.680**/**<.05**/63

* p is *p*-value; a bold $\hat{A}_{12}$ value implies DeepCollision performed significantly better than GS/RS; a $\hat{A}_{12}$ value decorated by symbol "×" indicates DeepCollision was significantly worse than RS/GS; for the rest $\hat{A}_{12}$ values, no significant differences between two methods can be observed; $\hat{A}_{12}$ magnitude: *negligible* ($\hat{A}_{12} \in$ (0.444, 0.556)), *small* ($\hat{A}_{12} \in$ [0.556, 0.638) or (0.362, 0.444]), *medium* ($\hat{A}_{12} \in$ [0.638, 0.714) or (0.286, 0.362]), *large* ($\hat{A}_{12} \in$ [0.714, 1.0] or [0, 0.286]).

RS. This is because the road structure of *R1* is very simple (Figure 2a).

High demand scenario generated ($S^{r,k}_{HighD}$). Overall, as shown in Table 5, DeepCollision achieved significant and consistent improvements over the baselines with *large* effect sizes on all of the roads.

Number of scenarios generated ($\#S$). DeepCollision generated significantly more scenarios than GS on all of the roads. DeepCollision performed significantly better than RS on *R2* and *R4*, while RS significantly outperformed DeepCollision on *R1* and *R3*. DeepCollision was trained for generating high-effective scenarios, that is with high chances to trigger collisions. This might account for the two cases where DeepCollision under-performed RS.

Time to the first collision ($Time_{1stC'}$). For $Time_{1stC'}$, DeepCollision overall outperformed the baselines, i.e., significant improvements on (R1, R4)-GS and (R1, R3, R4)-RS and showed no significant differences with R2-GS and R2-RS. Though DeepCollision significantly under-performed GS on R3, the difference of $Time_{1stC'}$ between DeepCollision and GS is, however, practically negligible, i.e., on average, 5.6s (23.1s with DeepCollision, 17.5s with GS).

Average time interval (TI_{avg}). In terms of TI_{avg}, when comparing with the baselines on the various roads, DeepCollision significantly outperformed RS and GS in most of the cases (i.e., 5 out of 8) except for (R2, R4)-GS and R2-RS, in which cases comparable results were achieved with DeepCollision and the baselines. Regarding time related measures, the improvement achieved by DeepCollision might not be significant as the other measures, i.e., collision-related and scenario-related. This might be because DeepCollision is prone to generate much more complex scenarios (i.e., high demand scenarios) which could lead to the AVUT driving at a relatively low speed.

Conclusion for RQ1: Compared with Random Strategy and Greedy Strategy, DeepCollision achieved the overall best performance i.e, effectively led to higher numbers of potential collisions and collision occurrences, generated high demand scenarios, took less time to reach the first collision, and had shorter time intervals. The differences of DeepCollision with the baselines are mostly significant.

5.4.2 Results for RQ2

Pair-wise comparisons. To answer RQ2, we performed pair-wise comparisons of the four DeepCollision models (i.e., M4, M6, M8, M10). Results are given in Table 6. When looking at the **collision metrics**, in terms of $\#C$, for *R1*, M4 significantly outperformed M8 with $\hat{A}_{12}$ at the *medium* magnitude level (i.e., $\geq$ 0.638). M6 achieved the best performance, all significantly, and $\hat{A}_{12}$ is at least at the *medium* magnitude level (i.e., $\geq$ 0.638 or $\leq$ 0.362). For *R2*, the four models are comparable to each other. For *R3* and *R4*, M4 significantly under-performed the other models, and $\hat{A}_{12}$ is at the *large* magnitude level for all the significant cases (i.e., $\leq$ 0.286). In terms of $\#C'$ and Pr_C, for *R1*, M4, M6 and M8 significantly outperformed M10 with at least the *medium* magnitude level (i.e., $\geq$ 0.638). For *R2*, both M6 and M8 outperformed M4 and M10 significantly, and $\hat{A}_{12}$ is at the *large* magnitude level (i.e., $\geq$ 0.714) for all the significant cases (i.e., $\geq$ 0.714 or $\leq$ 0.286). For *R3*, M4 was significantly better than the other models in terms of $\#C'$, but significantly under-performed the others for Pr_C. $\hat{A}_{12}$ is at least at the *medium* magnitude level (i.e., $\geq$ 0.638 or $\leq$ 0.362) for all the significant cases. For *R4*, all the other models performed significantly better than M4, and $\hat{A}_{12}$ is at the *large* magnitude level for all the significant cases (i.e., $\leq$ 0.286). When looking at the **scenario metrics**, for *R1* and *R2*, there are more comparisons for which M6 outperformed the other models significantly. For *R3*, in terms of $\#S_{C'}$, M4 significantly outperformed the others with $\hat{A}_{12}$ at the *medium* magnitude level (i.e., $\geq$ 0.638), but under-performed the others when evaluated with the other 3 metrics for most of the comparisons with $\hat{A}_{12}$ at the *medium* magnitude level (i.e., $\geq$ 0.638 or $\leq$ 0.362) for all the cases. For *R4*, M4 significantly under-performed the other models for all the scenario metrics and $\hat{A}_{12}$ is at the *large* magnitude level (i.e., $\leq$ 0.286) for all the cases. In terms of the **time performance** of the models, we see that the four models performed quite differently. For instance, M4 performed significantly worst than the others for *R4*.

In addition, we observed that M4 performed consistently worse than the other models on roads *R2* and *R4*. After carefully replaying the results, we found that few collisions were due to traffic jams (mostly caused by NPC vehicles coming from various directions). In such situations, the shortest OTP (i.e., 4s) of M4 is the plausible reason that caused the traffic jams, as the NPC vehicles had no sufficient time to drive

TABLE 6: Results of the Vargha and Delaney statistics and the Mann–Whitney U test—RQ2*

Road			Collision (A vs. B)			Scenario (A vs. B)				Time (B vs. A)	
			$\#C$	$\#C'$	Pr_C	$\#S_C$	$\#S_{C'}$	$\#S_{HighD}$	$\#S$	$Time_{1stC'}$	TI_{avg}
			$\hat{A}_{12}$/p	$\hat{A}_{12}$/p	$\hat{A}_{12}$/p	$\hat{A}_{12}$/p	$\hat{A}_{12}$/p	$\hat{A}_{12}$/p	$\hat{A}_{12}$/p	$\hat{A}_{12}$/p	$\hat{A}_{12}$/p
R1	M4	M6	.292×/<.05	.531/.672	.556/.459	.282×/<.05	.456/.524	.146×/<.05	.363/.070	.534/.651	.547/.539
		M8	**.654/<.05**	.519/.797	.564/.399	.556/.464	.571/.319	.324×/<.05	.426/.329	**.790/<.05**	**.739/<.05**
		M10	.647/.052	**.713/<.05**	**.689/<.05**	**.736/<.05**	**.653/<.05**	.378/.107	.438/.416	.646/.053	.640/.063
	M6	M8	**.768/<.05**	.494/.944	.530/.695	**.739/<.05**	.617/.098	**.697/<.05**	.588/.246	**.686/<.05**	**.689/<.05**
		M10	**.668/<.05**	**.691/<.05**	**.674/<.05**	**.819/<.05**	**.706/<.05**	**.701/<.05**	.639/.064	.570/.354	.603/.173
	M8	M10	.644/.055	**.717/<.05**	**.663/<.05**	**.702/<.05**	.581/.253	.544/.559	.596/.206	.433/.373	.448/.491
R2	M4	M6	.473/.728	.214×/<.05	.075×/<.05	.374/.095	.223×/<.05	.261×/<.05	.189×/<.05	.383/.120	.331×/<.05
		M8	.514/.859	.167×/<.05	.052×/<.05	.412/.246	.176×/<.05	.138×/<.05	.120×/<.05	.322×/<.05	.413/.252
		M10	.539/.605	.478/.764	.207×/<.05	.416/.267	.466/.639	.122×/<.05	.076×/<.05	**.678/<.05**	**.661/<.05**
	M6	M8	.523/.762	.480/.794	.458/.584	.551/.501	.504/.958	.364/.072	.371/.086	.419/.285	.600/.186
		M10	.548/.530	**.754/<.05**	**.718/<.05**	.544/.564	**.750/<.05**	.362/.068	.222×/<.05	**.680/<.05**	**.768/<.05**
	M8	M10	.527/.728	**.797/<.05**	**.774/<.05**	.498/.982	**.771/<.05**	.528/.712	.385/.128	**.746/<.05**	**.746/<.05**
R3	M4	M6	.238×/<.05	**.675/<.05**	.297×/<.05	.212×/<.05	**.667/<.05**	.302×/<.05	.207×/<.05	.170×/<.05	**.835/<.05**
		M8	.106×/<.05	**.708/<.05**	.092×/<.05	.157×/<.05	**.757/<.05**	.084×/<.05	.638/.067	**.717/<.05**	**.883/<.05**
		M10	.271×/<.05	**.772/<.05**	.302×/<.05	.298×/<.05	**.840/<.05**	.192×/<.05	.640/.063	**.748/<.05**	**.942/<.05**
	M6	M8	.164×/<.05	.545/.547	.266×/<.05	.283×/<.05	.624/.096	.164×/<.05	**.942/<.05**	**.927/<.05**	.592/.225
		M10	.466/.652	.618/.109	.479/.790	.513/.865	**.721/<.05**	.321×/<.05	**.963/<.05**	**.912/<.05**	**.719/<.05**
	M8	M10	**.672/<.05**	.573/.321	**.682/<.05**	**.661/<.05**	.602/.169	**.651/<.05**	.476/.756	.513/.864	.621/.109
R4	M4	M6	.161×/<.05	.122×/<.05	.040×/<.05	.093×/<.05	.108×/<.05	.160×/<.05	.233×/<.05	.108×/<.05	.223×/<.05
		M8	.104×/<.05	.098×/<.05	.021×/<.05	.085×/<.05	.108×/<.05	.218×/<.05	.188×/<.05	.042×/<.05	.234×/<.05
		M10	.092×/<.05	.246×/<.05	.059×/<.05	.027×/<.05	.232×/<.05	.076×/<.05	.086×/<.05	.030×/<.05	.316×/<.05
	M6	M8	.436/.399	.445/.468	.361/.065	.452/.525	.490/.900	.522/.773	.411/.237	.394/.099	.496/.965
		M10	.401/.188	.625/.097	.489/.894	.353/.051	.608/.152	.378/.107	.279×/<.05	.382/.066	**.666/<.05**
	M8	M10	.460/.600	**.661/<.05**	.598/.193	.384/.126	.623/.101	.368/.081	.377/.102	.487/.827	**.655/<.05**

* p is *p*-value; a bold $\hat{A}_{12}$ value implies A performed significantly better than B; a $\hat{A}_{12}$ value decorated by symbol "×" indicates A was significantly worse than B; for the rest there was no significant difference between A and B; $\hat{A}_{12}$ magnitude: *negligible* ($\hat{A}_{12} \in$ (0.444, 0.556)), *small* ($\hat{A}_{12} \in$ [0.556, 0.638) or (0.362, 0.444]), *medium* ($\hat{A}_{12} \in$ [0.638, 0.714) or (0.286, 0.362]), *large* ($\hat{A}_{12} \in$ [0.714, 1.0] or [0, 0.286]).

away from the environment and sometimes even collide to each other, the AVUT cannot sufficiently explore the road to properly cope with chaotic situations such as congested intersections, and consequently its movement was blocked. For the models with longer OTP (i.e., M6, M8 and M10), we rarely observed such traffic jams on *R2* and *R4*, after replaying some scenarios. On the other hand, the road structures of *R1* and *R3* are relatively simple (to compare with *R2* and *R4*) because of lack of cross intersections, which are not easy to have traffic jams. Thus, M4 (with the shortest OTP) introduces new environment configurations in a shorter period of time, thereby having higher chances to cause higher potential collisions(i.e., higher $\#C'$ and $\#S_{C'}$).

Model recommendations. We design Table 7, from which we derive our recommendations on the models. The last row (Rec_M) of the Table 7 represents our recommendation for each metric across the roads, and the last column (Rec_{all}) reports our recommendation for each road across the metrics. In terms of **collision** category (Rec_C), M8 is recommended, as it achieved the best performance for all the roads except for *R1*. Regarding the **scenario** category (Rec_S), M6 is recommended for both *R1* and *R2*, which have simple road structures (Figure 2). For more complicated roads (i.e., *R3* and *R4*), M8 and M10 are recommended, for a low frequency of changing the environment (i.e., longer OTPs) on complex roads tend to reduce the possibility of traffic jams, as we discussed earlier. For the **efficiency** to observe (potential) collisions, M6 is recommended for all the roads except for *R1*, for which M4 won over the other models. When counting the total number of recommendations across the roads, we recommend M6 as the solution for the time category of metrics. In total, by considering all the metrics on all the roads, M6 achieved the best. With respect to the **time performance** of the models across the metrics, M6 is recommended for *R1*, *R2* and *R4*; M8 is recommended for *R3*. For the most complex road *R4*, M6 and M8 performed equally good for all the metric categories, and M10 performed the best for the scenario metrics. This tells us that for complex road structures, to test an AVUT, we should think from the perspective of various testing objectives, e.g., targeting more on generating effective scenarios or discovering more collision occurrences.

At last, we want to emphasize that a high frequency of changing the environment (i.e., shorter OTPs such as 4s for M4) contributes to generate scenarios that might result in a high probability of triggering collisions (e.g., the case of *R3* for $\#C'$ by M4), meanwhile it is also risky that the generated scenarios are too complicated to be dealt with by the AVUT (e.g., traffic jams on *R2* and *R4* by M4 as discussed before). This, once again, indicates that different roads put on different requirements on the minimum OTP, e.g., larger OTPs (more than 6s) are needed for generating more effective scenarios on complicated roads. It is therefore not optimal to use an uniformed model for all roads. A strategy that automatically adapts to various road structures is needed, in the future, to select suitable models for different roads, which is one of our future work.

Conclusion for RQ2: The four DeepCollision models performed significantly differently for more than half of the 216 comparisons ($4(roads) \times 6(pairs) \times 9(metrics)$), and M6 is recommended across all the measures and roads.

TABLE 7: DeepCollision model recommendations—RQ2*

	Collision			Rec_C	Scenario			Rec_S	Time		Rec_T	Rec_{all}
Road	#C	#C'	Pr_C		#S_C	#$S_{C'}$	#S_{HighD}		$Time_{1stC'}$	TI_{avg}		
R1	M6	M4, M8, M6	M4, M6, M8	**M6**	M6	M6, M4, M8	M6	**M6**	M4, M6, M10	M4, M6	**M4**	**M6**
R2	M6, M4, M8, M10	M8, M6	M8, M6	**M8**	M6, M8, M10, M4	M6, M8	M8, M10, M6	**M6**	M8, M6	M6, M8	**M6/M8**	**M6**
R3	M8	M4	M8	**M8**	M8	M4	M8	**M8**	M6	M4	**M4/M6**	**M8**
R4	M10, M8, M6	M8, M6	M10, M8, M6	**M8**	M10, M8, M6	M8, M6, M10	M10, M6, M8	**M10**	M10, M8, M6	M6, M8	**M6**	**M6/M8/M10**
Rec_M	M6/M8	M8	M8	**M8**	M6/M8	M6/M8	M6/M8	**M6**	M6	M6	**M6**	**M6**

* For each metric and each road (e.g., #C' and *R1*), select model(s) that do not significantly under-perform any other model based on p-values (Table 6) and order them in the descendant sequence based on the $\hat{A}_{12}$ values of their pairwise comparisons; for each road and each metric category (e.g., *collision* and *R1*), a model recommended for all the metrics of the metric category (e.g., M6 for *R1*), the most times (e.g., M8 for *R3*), or the same number of times recommended but ordered at the first place (e.g., M8 for *R4*), is recommended; the overall recommendation for each road (i.e., Rec_{all}) is made by selecting the most recommended model across the metric categories (e.g., for *R1*, recommending M6 for Rec_C, Rec_S and Rec_T); when facing a tie, / is for showing that both models are recommended; the same rule is applied for recommendations made across all the roads for each metric and each metric category.

5.5 Threats to Validity

Conclusion Validity. To draw reliable conclusions, we employed appropriate statistical tests and followed a rigorous statistical procedure to analyze the collected data. Specifically, to reduce the possibility that the results were obtained by chance, we ran the trained models and the baselines 30 times. The setting of the 30 runs follows the guidelines in the work by Mnih [15], in which DQN was first proposed and adopted this setting to account for randomness. Furthermore, we performed pairwise comparisons with the Mann-Whitney U test and used the Vargha-Delaney $\hat{A}_{12}$ metric for effect size as recommended by Arcuri and Briand [73].

Internal Validity. To mitigate potential threats to the internal validity caused by DQN parameter settings, we trained different DeepCollision models under identical network architecture, learning algorithm, reward design and hyperparameter settings, to ensure fair comparisons among them. Furthermore, we understand that the hyperparameter tuning of DQN may influence the performance of DeepCollision. Thus, in this paper, our implementation is based on the PyTorch [78] framework and we trained all the models based on the hyperparameters from [68]. Besides, the performance of DeepCollision may be improved by tuning hyperparameters of DQN, which however requires dedicated and possibly large-scale empirical studies, which unfortunately cannot be covered in this paper. We also presented detailed information about the subject systems, simulator used, and DQN algorithm settings, etc. More details are available in the replication package (see Section 5.6).

Construct Validity. We used comparable metrics (e.g., number of collisions (#C'), Section 5.3.4) to compare DeepCollision with the baselines.

External Validity. For the evaluation, we employed an industrial scale ADS, i.e., Apollo, and the commonly applied LGSVL simulator. To use another simulator, DeepCollision can be tailored to integrate our already-developed REST API endpoints with the simulator with minimum effort. We however acknowledge that conducting more case studies is always preferred as doing so will strengthen the conclusions we have drawn. However, it is time-wise expensive to conduct large empirical studies with more roads and AVUTs because running simulations take a lot time. In the future, we plan to gradually conduct more case studies to further strengthen our conclusions.

5.6 Data Availability

The replication package that supports the findings of this study is available in Zenodo with the identifier data DOI[2]. The replication package is also available in our Github online repository[3]. The replication package contains the following contents: 1) The algorithm of DeepCollision, including the network architecture and the DQN hyperparameter settings; 2) All the available data of the pilot study (Section 5.2); 3) The dataset containing all the raw data for the analyses and the generated scenarios of the formal experiment (Section 5.3); and 4) The REST API endpoints for the environment configuration, along with one example demonstrating how the APIs should be used.

6 Discussion

Encoding AVUT environment and states. Operating environments of AVUT are complex. In our current implementation and experiments, we chose 12 parameters to demonstrate the applicability of DeepCollision, five of which are about the operating environment and the other seven are about AVUT state. Nonetheless, it can include more parameters and AVUT states. Our current encoding already showed good results, which gives us the confidence to extend this encoding for additional parameters and AVUT states in the future. Also, as reported by Chen et al. [21], a complex input can easily get over-fitted during a reinforcement learning process. A dedicated empirical study is needed, in the future, to help to select a proper set of parameters and AVUT states to be encoded in DeepCollision.

Using outputs for supporting further diagnoses and regression testing. With DeepCollision, scenarios and environment configurations that have been tested can be outputted for replaying. By selecting and replaying (potential) collisions/scenarios, DeepCollision can facilitate diagnoses. In addition, to better perform diagnoses, DeepCollision has been integrated with a tool named LiveTCM [79] which enabled a friendly interface to show detailed states of AVUT and environments. Moreover, DeepCollision can classify collisions according to different road structures and volumes of NPC vehicles such that users can replay specific categories of collisions (e.g., all collisions with NPC vehicles of large volumes) when needed.

Such outputs could also be used for other ADS related studies, e.g., selecting scenarios from the outputs and prioritizing them for supporting regression testing of ADSs [80].

[2]https://doi.org/10.5281/zenodo.5906634
[3]https://github.com/simplexity-lab/DeepCollision

Analyzing collisions with respect to obstacle types. We observe that the AVUT is prone to collide with pedestrians, and static obstacles of small volumes (volume ≤ 3, Table 4) and NPC vehicles of large volumes (volume > 60) at very close distances apart. This observation motivated us to replay the collisions, and we observed that when the AVUT drives very close to small static obstacles, though the radar can detect them, it is often too late for the AVUT to avoid collisions. When facing large NPC vehicles (e.g., school bus) at very close distances, the radar cannot even detect them for unknown reasons and hence lead to collisions. We observed in total 905 such cases in our experiments. For pedestrians, the radar cannot detect most of them and hence lead to collisions, especially when the pedestrians are being static.

Distributions of longitudinal and lateral collisions. Recall that we consider two types of collision probabilities in DeepCollision (Section 4.3), i.e., longitudinal and lateral. To know their distributions, we replayed all the 1815 collisions that DeepCollision collected and observed that: 1) On *R1* (no cross-intersection), 19 (12%) of the total 163 collisions are lateral collisions and 144 (88%) of them are longitudinal collisions; 2) On *R2* (with two cross-intersections), we observed 371 (84%) lateral and 70 (16%) longitudinal collisions. Similarly on *R4* (with also two cross-intersections), there are 661 (92%) lateral and 59 (8%) longitudinal collisions; and 3) On *R3* where the AVUT drove on a one-way-road with two T-intersections, we observed 27% (134 out of 491) lateral and 73% (357 out of 491) longitudinal collisions. Our further investigations reveal that lateral collisions were mostly due to NPC vehicles crossing intersections and longitudinal collisions are prone to occur as rear-end collisions. For instance, on *R2* and *R4* (with more intersections than the others), we observed more scenarios of NPC vehicles crossing intersections, which accounts for the observations of the high percentages of lateral collisions: 84% and 92%, respectively. However, to generalize results, we need to conduct more experiments with more road structures.

Adaptive DeepCollision. From the experiment, we found that the road structure can impact collisions in various ways. For instance, the slope on a road might affect the AVUT's speed, an intersection can affect traffic flows, and sidewalks might enable the generation of more pedestrians. First, it might be needed to adapt the OTP configuration (e.g., 4s-10s, as experimented in our experiment) of DeepCollision for optimally handling various road structures (as parts of inputs to the models). In addition, various types of obstacles might refer to different analyses, e.g., risk analysis about trash bin vs. pedestrians and ethical studies on infants vs. elders [81]. With our study, we already observed that different volumes of obstacles affect occurrences of collisions. To test whether an AVUT can make decisions when facing different types of obstacles, we think that there is a need to provide different obstacle generation strategies in the future.

7 Conclusion and Future Work

In this paper, we investigated ADS testing in continuously-changing environment and formalized the testing problem as a Markov Decision Process (MDP) problem. We presented *DeepCollision*—an environment configuration learning approach based on reinforcement learning (RL), which intelligently learns environment configurations that lead an autonomous vehicle to crash. It uses a Deep Q-Network (DQN) with collision probability as the reward function to guide the learning. To assess the cost-effectiveness of *DeepCollision*, we evaluated DeepCollision by comparing it with two baselines, i.e., random and greedy. Results show that all the DeepCollision models significantly outperformed the two baselines in most cases and the DeepCollision model with 6s observation time period is recommended.

Though DeepCollision has demonstrated good performance, several interesting issues are still worthy of being investigated in the future. ***Adopting more safety metrics***. DeepCollision now only adopts *collision probability* to define the reward function of reinforcement learning (RL). And safety distances and current distances are selected as measures for calculating collision probability. In the future, we plan to refine the way how DeepCollision calculates the distance and evaluate it on curvy roads. Additionally, we plan to employ other metrics (e.g., time to collision and distance to obstacles) to enhance the reward function from more perspectives. ***Investigating algorithm settings and various RL algorithms***. Hyperparameter settings have impact on the performance of RL models, thus, tuning and optimizing them is one of our future works. We plan to employ automatic hyperparameter optimization framework such as Optuna [82]. We will also investigate other RL algorithms and compare their performance with DQN employed in DeepCollision. ***Devising adaptive DeepCollision***. As observed from our experiment results, the performance of DeepCollision configured with different OTPs varies for different roads. Thus, in the future, we plan to propose a strategy to automatically determinate how to configure the OTP of DeepCollision such that it adapts, at runtime, to various road structures. For instance, we can see that such a strategy will be useful for configuring DeepCollision with different OTPs to test various fragments of a road. ***Simulating a continuously changing weather and time***. The weather and time is currently fixed in a period of time, and we would also like to point it out that simulating a continuously changing weather and time and ensuring their realism are still open issues [83]. In the future, we will investigate other approaches to simulate a continuously changing weather and time. Finally, ***environment uncertainties*** are crucial for testing robustness of ADSs. Thus, systematically introducing and quantifying uncertainties in the operating environment is another future work. More empirical studies need to be carried out to assess the applicability of DeepCollision in various ADS testing contexts, e.g., real-world and hardware-in-the-loop.

Acknowledgements

This work is supported by National Natural Science Foundation of China under Grant No. 61872182. It is also supported by the Co-evolver project (No. 286898/F20) funded by the Research Council of Norway. Huihui Zhang is supported by the project (grant No. ZR2021MF026) funded by Shandong Provincial Natural Science Foundation. Man Zhang is funded by the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No. 864972).

REFERENCES

[1] S. Shafaei, S. Kugele, M. H. Osman, and A. Knoll, "Uncertainty in machine learning: A safety perspective on autonomous driving," in *International Conference on Computer Safety, Reliability, and Security*, pp. 458–464, Springer, 2018.

[2] J. Chen, C. Tang, L. Xin, S. E. Li, and M. Tomizuka, "Continuous decision making for on-road autonomous driving under uncertain and interactive environments," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1651–1658, IEEE, 2018.

[3] M. Zhang, B. Selic, S. Ali, T. Yue, O. Okariz, and R. Norgren, "Understanding uncertainty in cyber-physical systems: A conceptual model," in *Modelling Foundations and Applications* (A. Wąsowski and H. Lönn, eds.), (Cham), pp. 247–264, Springer International Publishing, 2016.

[4] B. R. Kiran, I. Sobh, V. Talpaert, P. Mannion, A. A. A. Sallab, S. Yogamani, and P. Pérez, "Deep reinforcement learning for autonomous driving: A survey," *arXiv preprint arXiv:2002.00444*, 2020.

[5] O. Buhler and J. Wegener, "Automatic testing of an autonomous parking system using evolutionary computation," *SAE SP*, pp. 115–122, 2004.

[6] O. Buehler and J. Wegener, "Evolutionary functional testing of a vehicle brake assistant system," in *6th Metaheuristics International Conference, Vienna, Austria*, Citeseer, 2005.

[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] 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.

[10] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: Nsga-ii," *IEEE transactions on evolutionary computation*, vol. 6, no. 2, pp. 182–197, 2002.

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

[12] X. Zhang, J. Tao, K. Tan, M. Törngren, J. M. G. Sánchez, M. R. Ramli, X. Tao, M. Gyllenhammar, F. Wotawa, N. Mohan, *et al.*, "Finding critical scenarios for automated driving systems: A systematic literature review," *arXiv preprint arXiv:2110.08664*, 2021.

[13] L. Chen, X. Hu, B. Tang, and Y. Cheng, "Conditional dqn-based motion planning with fuzzy logic for autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[14] 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.

[15] 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.

[16] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[17] S. Phaniteja, P. Dewangan, P. Guhan, A. Sarkar, and K. M. Krishna, "A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots," in *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pp. 1818–1823, IEEE, 2017.

[18] B. Hetzel, *The Complete Guide to Software Testing*. USA: QED Information Sciences, Inc., 2nd ed., 1988.

[19] "Baidu apollo." https://github.com/ApolloAuto/apollo, 2020.

[20] 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," *arXiv preprint arXiv:2005.03778*, 2020.

[21] J. Chen, B. Yuan, and M. Tomizuka, "Model-free deep reinforcement learning for urban autonomous driving," in *2019 IEEE intelligent transportation systems conference (ITSC)*, pp. 2765–2771, IEEE, 2019.

[22] F. Hauer, A. Pretschner, and B. Holzmüller, "Fitness functions for testing automated and autonomous driving systems," in *International Conference on Computer Safety, Reliability, and Security*, pp. 69–84, Springer, 2019.

[23] O. Bühler and J. Wegener, "Evolutionary functional testing," *Computers & Operations Research*, vol. 35, no. 10, pp. 3144–3160, 2008.

[24] 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.

[25] M. Althoff and S. Lutz, "Automatic generation of safety-critical test scenarios for collision avoidance of road vehicles," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, pp. 1326–1333, IEEE, 2018.

[26] H. Ishibuchi, Y. Nojima, and T. Doi, "Comparison between single-objective and multi-objective genetic algorithms: Performance comparison and performance measures," in *2006 IEEE International Conference on Evolutionary Computation*, pp. 1143–1150, IEEE, 2006.

[27] Y. Jin, "Surrogate-assisted evolutionary computation: Recent advances and future challenges," *Swarm and Evolutionary Computation*, vol. 1, no. 2, pp. 61–70, 2011.

[28] M. J. Kochenderfer and K. H. W. Tim A. Wheeler, *Algorithms for Decision Making*. MIT press, 2022.

[29] A. E. Sallab, M. Abdou, E. Perot, and S. Yogamani, "Deep reinforcement learning framework for autonomous driving," *Electronic Imaging*, vol. 2017, no. 19, pp. 70–76, 2017.

[30] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid, "Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems," in *2018 33rd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 132–142, IEEE, 2018.

[31] K. Pei, Y. Cao, J. Yang, and S. Jana, "Deepxplore: Automated whitebox testing of deep learning systems," in *proceedings of the 26th Symposium on Operating Systems Principles*, pp. 1–18, 2017.

[32] Y. Tian, K. Pei, S. Jana, and B. Ray, "Deeptest: Automated testing of deep-neural-network-driven autonomous cars," in *Proceedings of the 40th international conference on software engineering*, pp. 303–314, 2018.

[33] H. Zhou, W. Li, Z. Kong, J. Guo, Y. Zhang, B. Yu, L. Zhang, and C. Liu, "Deepbillboard: Systematic physical-world testing of autonomous driving systems," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, pp. 347–358, 2020.

[34] A. Corso, P. Du, K. Driggs-Campbell, and M. J. Kochenderfer, "Adaptive stress testing with reward augmentation for autonomous vehicle validatio," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 163–168, IEEE, 2019.

[35] R. Lee, M. J. Kochenderfer, O. J. Mengshoel, G. P. Brat, and M. P. Owen, "Adaptive stress testing of airborne collision avoidance systems," in *2015 IEEE/AIAA 34th Digital Avionics Systems Conference (DASC)*, pp. 6C2–1, IEEE, 2015.

[36] X. Xie, L. Ma, F. Juefei-Xu, M. Xue, H. Chen, Y. Liu, J. Zhao, B. Li, J. Yin, and S. See, "Deephunter: A coverage-guided fuzz testing framework for deep neural networks," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 146–157, 2019.

[37] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, *et al.*, "Deepmutation: Mutation testing of deep learning systems," in *2018 IEEE 29th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 100–111, IEEE, 2018.

[38] Q. Hu, L. Ma, X. Xie, B. Yu, Y. Liu, and J. Zhao, "Deepmutation++: A mutation testing framework for deep learning systems," in *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pp. 1158–1161, IEEE, 2019.

[39] Z. Peng, J. Yang, T.-H. Chen, and L. Ma, "A first look at the integration of machine learning models in complex autonomous driving systems: a case study on apollo," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 1240–1250, 2020.

[40] F. Harel-Canada, L. Wang, M. A. Gulzar, Q. Gu, and M. Kim, "Is neuron coverage a meaningful measure for testing deep neural networks?," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, pp. 851–862, 2020.

[41] W. Zhou, J. S. Berrio, S. Worrall, and E. Nebot, "Automated evaluation of semantic segmentation robustness for autonomous

driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 5, pp. 1951–1963, 2019.

[42] C. Liu and R. Subramanian, "NHTSA," *National Center for Statistics and Analysis. Washington, DC: US Department of Transportation, National Highway Traffic Safety Administration*, 2020.

[43] R. T. Fielding, *Architectural styles and the design of network-based software architectures*. University of California, Irvine, 2000.

[44] J. W. Ro, P. S. Roop, and A. Malik, "A new safety distance calculation for rear-end collision avoidance," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[45] A. Doi, T. Butsuen, T. Niibe, T. Takagi, Y. Yamamoto, and H. Seni, "Development of a rear-end collision avoidance system with automatic brake control," *Jsae Review*, vol. 15, no. 4, pp. 335–340, 1994.

[46] N. Uno, Y. Iida, S. Itsubo, and S. Yasuhara, "A microscopic analysis of traffic conflict caused by lane-changing vehicle at weaving section," in *Proceedings of the 13th Mini-EURO Conference-Handling Uncertainty in the Analysis of Traffic and Transportation Systems, Bari, Italy*, pp. 10–13, 2002.

[47] F. Bella and R. Russo, "A collision warning system for rear-end collision: a driving simulator study," *Procedia-social and behavioral sciences*, vol. 20, pp. 676–686, 2011.

[48] S. Cheng, L. Li, H.-Q. Guo, Z.-G. Chen, and P. Song, "Longitudinal collision avoidance and lateral stability adaptive control system based on mpc of autonomous vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 21, no. 6, pp. 2376–2385, 2019.

[49] Z. Zhang, C. Wang, W. Zhao, and J. Feng, "Longitudinal and lateral collision avoidance control strategy for intelligent vehicles," *Proceedings of the Institution of Mechanical Engineers, Part D: Journal of Automobile Engineering*, p. 09544070211024048, 2021.

[50] K. Manjunath and N. Jaisankar, "A survey on rear end collision avoidance system for automobiles," *International journal of engineering and technology*, vol. 5, no. 2, pp. 1368–1372, 2013.

[51] T. Chen, K. Liu, Z. Wang, G. Deng, and B. Chen, "Vehicle forward collision warning algorithm based on road friction," *Transportation research part D: transport and environment*, vol. 66, pp. 49–57, 2019.

[52] D. Ngoduy, "Analytical studies on the instabilities of heterogeneous intelligent traffic flow," *Communications in Nonlinear Science and Numerical Simulation*, vol. 18, no. 10, pp. 2699–2706, 2013.

[53] D.-F. Xie, X.-M. Zhao, and Z. He, "Heterogeneous traffic mixing regular and connected vehicles: Modeling and stabilization," *IEEE Transactions on Intelligent Transportation Systems*, vol. 20, no. 6, pp. 2060–2071, 2018.

[54] M. Fransen, J. Heussler, E. Margiotta, and J. Edmonds, "Quantitative gait analysis—comparison of rheumatoid arthritic and non-arthritic subjects," *Australian Journal of Physiotherapy*, vol. 40, no. 3, pp. 191–199, 1994.

[55] S. C. Miff, D. S. Childress, S. A. Gard, M. R. Meier, and A. H. Hansen, "Temporal symmetries during gait initiation and termination in nondisabled ambulators and in people with unilateral transtibial limb loss.," *Journal of Rehabilitation Research & Development*, vol. 42, no. 2, 2005.

[56] S. Shalev-Shwartz, S. Shammah, and A. Shashua, "On a formal model of safe and scalable self-driving cars," *arXiv preprint arXiv:1708.06374*, 2017.

[57] D. Qu, X. Chen, W. Yang, and X. Bian, "Modeling of car-following required safe distance based on molecular dynamics," *Mathematical Problems in Engineering*, vol. 2014, 2014.

[58] E. Leurent, "A survey of state-action representations for autonomous driving," 2018.

[59] B. C. Stadie, S. Levine, and P. Abbeel, "Incentivizing exploration in reinforcement learning with deep predictive models," *arXiv preprint arXiv:1507.00814*, 2015.

[60] S. O.-R. A. V. S. Committee *et al.*, "Taxonomy and definitions for terms related to on-road motor vehicle automated driving systems," *SAE Standard J*, vol. 3016, pp. 1–16, 2014.

[61] H. Fan, F. Zhu, C. Liu, L. Zhang, L. Zhuang, D. Li, W. Zhu, J. Hu, H. Li, and Q. Kong, "Baidu apollo em motion planner," *arXiv preprint arXiv:1807.08048*, 2018.

[62] 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.

[63] V. Vegamoor, S. Rathinam, and S. Darbha, "String stability of connected vehicle platoons under lossy v2v communication," *IEEE Transactions on Intelligent Transportation Systems*, 2021.

[64] M. Mukadam, A. Cosgun, A. Nakhaei, and K. Fujimura, "Tactical decision making for lane changing with deep reinforcement learning," 2017.

[65] S. F. Abedin, M. S. Munir, N. H. Tran, Z. Han, and C. S. Hong, "Data freshness and energy-efficient uav navigation optimization: A deep reinforcement learning approach," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[66] W. Liu, Z. Wang, X. Liu, N. Zeng, Y. Liu, and F. E. Alsaadi, "A survey of deep neural network architectures and their applications," *Neurocomputing*, vol. 234, pp. 11–26, 2017.

[67] M. Zhu, Y. Wang, Z. Pu, J. Hu, X. Wang, and R. Ke, "Safe, efficient, and comfortable velocity control based on reinforcement learning for autonomous driving," *Transportation Research Part C: Emerging Technologies*, vol. 117, p. 102662, 2020.

[68] M. Zhou, "pytorch tutorials," https://github.com/MorvanZhou/PyTorch-Tutorial, 2021.

[69] 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.

[70] K. Czarnecki, "Operational world model ontology for automated driving systems–part 1: Road structure," *Waterloo Intelligent Systems Engineering Lab (WISE) Report, University of Waterloo*, 2018.

[71] R. Fuller, "Towards a general theory of driver behaviour," *Accident analysis & prevention*, vol. 37, no. 3, pp. 461–472, 2005.

[72] K. Czarnecki, "Operational design domain for automated driving systems: Taxonomy of basic terms," *Waterloo Intelligent Systems Engineering (WISE) Lab, University of Waterloo, Canada*, 2018.

[73] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *2011 33rd International Conference on Software Engineering (ICSE)*, pp. 1–10, IEEE, 2011.

[74] I. M. Chakravarty, J. Roy, and R. G. Laha, "Handbook of methods of applied statistics," 1967.

[75] F. Wilcoxon, *Critical Values and Probability Levels for the Wilcoxon Rank Sum Test and the Wilcoxon Signed Rank Test*. American Cyanamid, 1963.

[76] A. Vargha and H. D. Delaney, "A critique and improvement of the cl common language effect size statistics of mcgraw and wong," *Journal of Educational and Behavioral Statistics*, vol. 25, no. 2, pp. 101–132, 2000.

[77] B. Kitchenham, L. Madeyski, D. Budgen, J. Keung, P. Brereton, S. Charters, S. Gibbs, and A. Pohthong, "Robust statistical methods for empirical software engineering," *Empirical Software Engineering*, vol. 22, no. 2, pp. 579–630, 2017.

[78] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *arXiv preprint arXiv:1912.01703*, 2019.

[79] Y. Shi, C. Lu, M. Zhang, H. Zhang, T. Yue, and S. Ali, "Restricted natural language and model-based adaptive test generation for autonomous driving," in *2021 ACM/IEEE 24th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, pp. 101–111, IEEE, 2021.

[80] C. Lu, H. Zhang, T. Yue, and S. Ali, "Search-based selection and prioritization of test scenarios for autonomous driving systems," in *Search-Based Software Engineering* (U.-M. O'Reilly and X. Devroey, eds.), (Cham), pp. 41–55, Springer International Publishing, 2021.

[81] K. Hao, "Should a self-driving car kill the baby or the grandma? depends on where you're from.," in *AI Ethics*, MIT, 2018.

[82] T. Akiba, S. Sano, T. Yanase, T. Ohta, and M. Koyama, "Optuna: A next-generation hyperparameter optimization framework," in *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 2623–2631, 2019.

[83] J. Sillmann, T. Thorarinsdottir, N. Keenlyside, N. Schaller, L. V. Alexander, G. Hegerl, S. I. Seneviratne, R. Vautard, X. Zhang, and F. W. Zwiers, "Understanding, modeling and predicting weather and climate extremes: Challenges and opportunities," *Weather and climate extremes*, vol. 18, pp. 65–74, 2017.