Adversarial Testing with Reinforcement Learning: A Case Study on Autonomous Driving

Andréa Doreste, Matteo Biagiola, Paolo Tonella
Software Institute - Università della Svizzera italiana, Lugano, Switzerland
{andrea.doreste, matteo.biagiola, paolo.tonella}@usi.ch

***Abstract*—Testing autonomous driving systems (ADSs) is essential to ensure their safety. Existing testing techniques manipulate the objects of the driving environment in order to trigger a misbehavior of the ADS under test. Reinforcement learning (RL) approaches have been applied to effectively modify the dynamic objects of the environment (e.g., pedestrians and other vehicles), also known as Non-Playable Characters (NPCs). However, existing RL approaches implement centralized controllers of the environment, resulting in possibly unrealistic and even invalid behaviors of the NPCs.**

In this paper, we propose to model NPCs as independent and fully autonomous agents to challenge the ADS under test (i.e., the ego ADS). In the first step of our approach, we train an adversarial ADS by designing a reward function as a linear combination of two components: (1) a component that encourages it to drive well in the given driving scenario, and (2) an adversarial component, that smoothly guides the agent towards a collision with the ego ADS. In our second step, we resume training of the ego ADS, to increase its robustness towards the behaviors of the adversarial ADS.

Our experiments on a highway driving scenario show that the adversarial ADS is significantly more effective at generating collisions of the ego ADS than a random baseline. Moreover, adversarial retraining induces safe behaviors of the ego ADS, preventing the adversarial ADS from colliding.

***Index Terms*—Autonomous Driving Reinforcement Learning Adversarial Testing.**

I. Introduction

Deep Neural Networks (DNNs) are increasingly applied to a wide range of complex tasks, including autonomous driving. Due to their safety-critical nature, autonomous driving systems (ADSs) need to be thoroughly tested before deployment.

Most of the ADS testing techniques proposed in the literature manipulate the environment in which the ADS operates [1]. Their objective is to cause a perturbation in the environment, eventually resulting in a misbehavior of the ADS (e.g., a collision with another object in the environment or an out-of-lane event). The testing problem is formulated as an optimization problem, where the objects in the environment, both static and dynamic ones, are manipulated to maximize an objective function (e.g., distance to misbehavior). One common way to address this problem is to use search-based testing techniques [1], [2], [3], [4], [5], [6], to effectively find critical configurations of the environment (i.e., test cases) given a limited search budget.

While search-based techniques are suitable to generate static configurations of the environment (e.g., the road shape on which the ADS drives), they are not designed to deal with sequential interactions at runtime, which are required when manipulating dynamic objects. On the other hand, in the Reinforcement Learning (RL) paradigm, the agent learns by interacting with the environment at each step, and naturally deals with the runtime effects of its actions. Recently, DeepCollision [7] and Morlot [8] formulated the ADS testing problem as an RL problem. In particular, they act on the dynamics of Non-Playable Characters (NPCs) in the environment, such as pedestrians and vehicles other than the one controlled by the ego ADS (i.e., the ADS under test). However, their RL agent is a centralized controller of the environment evolution, while in real driving scenarios each NPC acts independently.

We propose to model each NPC as a fully autonomous and independent adversarial agent (i.e., the adversarial ADS) to challenge the behavior of the ego ADS. Specifically, we model the NPC vehicle that is in front of the ego ADS, as an independent RL agent, mimicking a typical driving scenario in which the ego ADS shares the road with other intelligent actors. Existing studies show that such driving scenarios cover the majority of challenging situations for an ADS, as more than 80% of the accidents involving an ADS in California are caused by the maneuvers of other vehicles [9], [10].

Our approach takes as input an ego ADS, modeled as an RL agent, that it is able to drive in the presence of other autonomous vehicles. In particular, the ego ADS is trained to drive as fast as possible on the right lane of a highway and to avoid collisions with the other vehicles (we name this reward function, Quality of Driving (QoD) reward). In the first step of our approach, we bootstrap the training of the adversarial ADS by starting from the weights of the ego ADS. The ego ADS acts in inference mode, and its weights are frozen during training of the adversarial ADS. Such agent is trained to maximize a reward function that is a linear combination of two components. The first component is the same reward function used to train the ego ADS (i.e., the QoD reward), while the second component is purely adversarial, and it aims to smoothly guide the adversarial ADS towards triggering a collision with the ego ADS. In particular, we penalize the adversarial ADS if it is moving away from the ego ADS (proportionally to its relative velocity) and we reward it with the inverse of the distance between the two vehicles if the two agents are getting closer. Additionally, we reward the adversarial ADS by a large amount whenever it causes a collision.

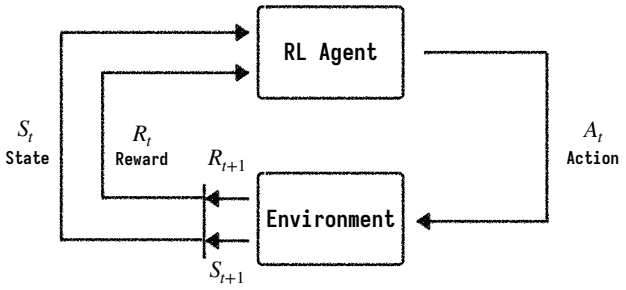

Fig. 1: Reinforcement learning loop. At each timestep t the agent receives a state from the environment S_t. Given the state, the agent outputs an action A_t that modifies the environment, which in turn gives back a reward R_{t+1} that the agent uses for learning.

The two components of the reward are conflicting, as maximizing the QoD component decreases the chances of a collision (i.e., the adversarial component decreases). However, the objective of the testing agent is to maximize the sum of both scalar values at each timestep. The net effect is that pure adversarial behaviors, which would result in invalid failures the ego ADS cannot avoid, are discouraged, since the QoD component of the reward counteracts them. As a result, our reward function minimizes the chances of invalid failures, while encouraging collisions the ego ADS can learn from to improve its robustness. To achieve this objective, the second step of our approach consists of resuming the training of the ego ADS in the presence of the adversarial ADS, which now acts in inference mode.

In our experiments, accounting for approximately 100 between training and testing simulation runs, we statistically compare the adversarial ADS resulting from our reward function with a random baseline. We empirically show that the adversarial ADS is significantly more effective at generating collisions than the random agent. Moreover, the retraining step is effective at improving the robustness of the ego ADS, by inducing a safe behavior that prevents collisions with the adversarial ADS.

II. Background

A. Overview

Reinforcement Learning (RL), is a learning paradigm that consists of learning a policy, i.e., how to map states into actions, to maximize a numerical reward signal [11]. The agent does not know a-priori which actions lead to the highest rewards, but it needs to learn it on its own by interacting with the environment.

At each timestep t, the agent perceives the current state of the environment S_t and selects an action A_t, which will make the environment transition to the next state S_{t+1} (see Figure 1). The action modifies the environment, which, in turn, sends a feedback signal to the agent, i.e., the reward R_{t+1}, guiding the agent towards the desired objective. The most common RL formulation is episodic, i.e., the interaction between the agent and the environment ends when certain conditions hold. In this context, the objective of the agent is to find a policy π that maximizes the sum of the cumulative sum of the rewards it gets in its lifetime.

This process can be formalized using a Marking Decision Process (MDP). An MDP is a 5-tuple, $\langle \mathcal{S}, \mathcal{A}, \mathcal{R}, \mathcal{P}, p_0 \rangle$, where:

1) $\mathcal{S}$ is the set of all valid states;
2) $\mathcal{A}$ is the set of all valid actions
3) $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \to \mathbb{R}$ is the reward function;
4) $\mathcal{P} : \mathcal{S} \times \mathcal{S} \times \mathcal{A} \to P(S)$ is the transition probability function, with $P(S_{t+1} \mid S_t, A_t)$ being the probability of transitioning from state S_t to state S_{t+1} after selecting action A_t;
5) $p_0(S)$ is the starting state distribution, i.e., the probability that the environment starts from a specific state $S_0 \in \mathcal{S}$.

The MDP obeys the Markov property, i.e., the future S_{t+1} is conditionally independent of the past $(S_{t-1}, \ldots, S_0)$ given the present S_t. In other words, the current state S_t summarizes all relevant aspects of the agent-environment interactions the agent need to take the best decision in the present. Given a trajectory τ, i.e., a sequence of $(S_t, A_t, R_{t+1}, S_{t+1})$, the agent is tasked to find a policy π that maximizes the return, defined as:

$$G(\tau) = \sum_{t=0}^{T} R_t \tag{1}$$

where T is the total number of timesteps in an episode. In general, the RL objective is expressed as a maximization of the expected return, as both the policy and the environment are stochastic.

The optimal policy π^* can be extracted from two quantities, i.e., the state-value function $V_\pi(S)$ and the action-value function $Q_\pi(S, A)$. The former quantifies how good is a certain state S, while the latter measures the value of the state-action pair (S, A). Both functions obey the Bellman equations [11], i.e., recursive consistency equations that relate the values of a state S (respectively, state-action pair (S, A)) to the values of all possible successor states (respectively, state-action pairs). Bellman equations can be solved in closed-form, by knowing the transition probability function of the environment $\mathcal{P}$. In particular, by solving the Bellman equation for the action-value function Q_π, we obtain Q^*_π, from which we can extract π^*, by choosing in each state S the action A that maximizes Q^*_π.

B. Deep RL

In general the transition probability function of the environment $\mathcal{P}$ is not known, and tabular methods that heuristically solve the Bellman equations, such as Monte Carlo methods and temporal difference learning, are not applicable to environments with large state and actions spaces. Deep learning techniques enable Deep RL (DRL) algorithms to function in such complex scenarios [12]. In particular, DRL algorithms

Fig. 2: The ego (in green) and the adversarial (in magenta) ADSs in the highway driving scenario of the *HighwayEnv* simulator [17].

use Deep Neural Networks (DNNs) to approximate state and action-value functions. The DRL algorithm that pioneered the field is Deep Q-Newtork (DQN) [12], that introduced the conceptual tools to deal with the instability caused by the introduction of a non-linear function approximator.

The DQN algorithm is still one of the most popular and effective DRL algorithms, and it is implemented in the major open-source DRL libraries [13], [14], [15], [16]. In our work, we leverage the DQN algorithm to address the driving task, as the state space of the agent has continuous variables.

III. Case Study

We consider a highway driving scenario implemented by the *HighwayEnv* simulator [17], a widely used benchmark in reinforcement learning [18], [19], [20], [21], [22], [23], [24] (at the time of writing, the simulator has $2k$ starts on Github).

Figure 2 shows two vehicles driving in the highway scenario. The vehicle on the left (in green) is the ego ADS, while the vehicle on the right (in magenta) is the adversarial ADS. The ego ADS drives in the two-lane scenario, keeping the highest possible speed while avoiding the other vehicles in the scene.

In particular, the observation space [1] of the agent is a matrix $V \times F$, where V is the number of vehicles in the scene, and F is the number of features. For the example in Figure 2, the number of rows in the observation matrix is $V = 2$, as there two vehicles in the scenario, while the number of columns is $F = 4$. The first two features are the x and y coordinates of the position of a vehicle, measured in meters, while the remaining two features are the v_x and v_y coordinates of the velocity, measured in m/s.

The action space of the agent is discrete. In particular, the environment exposes meta-actions, i.e., abstract actions on top of the low-level throttle and steering controls. The agent has 5 actions available, namely, switch to left lane, switch to right lane, accelerate, decelerate and stay idle. Not all actions are available in all states, for instance, change to right lane when the vehicle is already at the right edge of the road, or accelerating/decelerating when the vehicle has already reached the maximum/minimum speed; in such cases, the unavailable actions have the same effect of the stay idle action. The agent's policy runs at $1Hz$, i.e., the agent takes one action every second (which corresponds to 1 timestep), while the environment runs at $5Hz$, such that the low-level controllers can smoothly execute the meta-action, by splitting it into 5 sub-actions.

The reward function used to train the ego vehicle consists of several components, encouraging the agent to drive well in the given scenario. We name this reward Quality of Driving (QoD) reward, defined as follows:

$$R_{QoD} = \nu\Big(w_{spd} R_{spd} + w_{rlane} R_{rlane} + w_{col} R_{col} \Big) \qquad (2)$$

where R_{spd} encourages the agent to go at high speed, R_{rlane} privileges the rightmost lane, and R_{col} takes into account when there is a collision. The scalar rewards are weighted by $w_{(\cdot)}$, and the symbol $\nu(\cdot)$ represents a normalization function in $[0, 1]$. The speed reward, weighted by $w_{spd} = 0.4$, is defined as $R_{spd} = \nu(v_x)$, i.e., the normalized longitudinal velocity; the normalization function in this case is a linear function mapping the interval $[20, 30]$ in the target interval $[0, 1]$, where $20\,m/s$ is the minimum velocity and $30\,m/s$ is the maximum velocity. The rightmost-lane reward is boolean in our scenario, and it is weighted by $w_{rlane} = 0.1$. Likewise, the collision reward is one only when the there is a collision between the two vehicles; the corresponding weight is $w_{col} = -1$. The environment checks if two vehicles are colliding, by computing whether the moving polygons associated with the vehicles intersect. Finally, the reward is normalized linearly, mapping the interval $[w_{col}, w_{spd} + w_{rlane}]$, i.e., respectively the minimum and the maximum scalar rewards in a timestep, to the interval $[0, 1]$.

At the beginning of the episode, the lanes and the speeds for the two vehicles are sampled randomly within the respective ranges. The longitudinal positions are also chosen randomly, with the constraint that the adversarial vehicle is always in front. The episode terminates successfully either when the maximum number of timesteps (set to 30) is reached, or when the ego ADS overtakes the adversarial ADS. This last condition prevents the adversarial ADS from learning to collide with the ego ADS from the back, which the ego ADS could not avoid. On the contrary, when a collision occurs, the episode terminates unsuccessfully (i.e., with a failure).

The *HighwayEnv* simulator natively supports the multi-agent setting, which is required by our approach to both train the adversarial ADS and resume training of the ego ADS in the presence of the adversarial ADS. In particular, the simulator defines a `MultiAgentObservation` type as a tuple of two or more observations. Both agents perceive themselves in the first row of the observation matrix, while the remaining vehicle is in the second row. For instance, for the adversarial ADS, the features in the first row are relative to the vehicle it is controlling, while the second row lists the features of the ego ADS; viceversa, if the perspective is that of the ego ADS. Likewise, the simulator defines a `MultiAgentAction` as a tuple of two or more actions, that correspond to the observations the two or more agents

[1] *Observation* is a more general term than *state*, as using *state* implies that all the variables of the environment are observable to the agent.

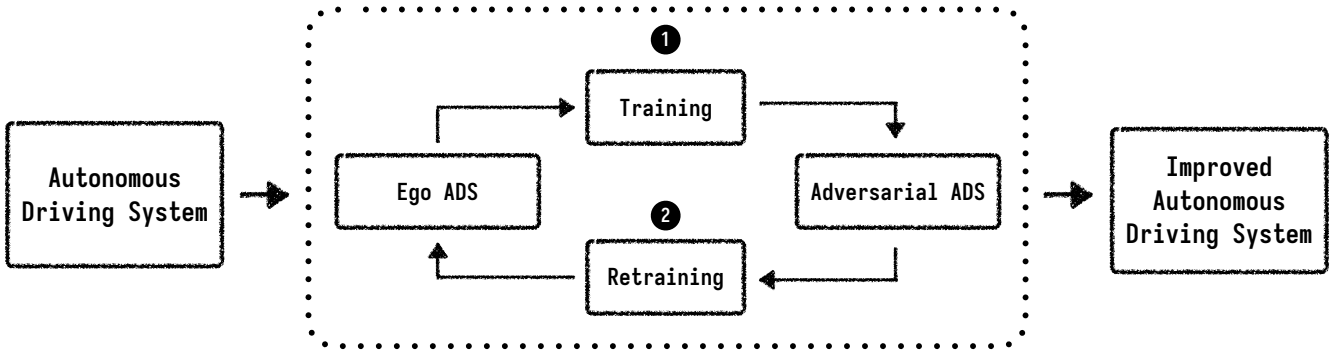

Fig. 3: Overview of our approach. The first step ❶ trains the adversarial ADS to challenge the ego ADS, while the second step ❷ uses the trained adversarial ADS to improve it.

receive as input. In our case, the actions of the ego and of the adversarial ADSs are collected by the simulation environment that sends them to the respective vehicles.

IV. Approach

Figure 3 shows the overview of our approach. Our approach takes as input a pre-trained ego ADS. As the environment is not equipped with pre-trained agents for the highway scenario, we train the ego ADS from scratch. During pre-training, the ADS in front (see the blue vehicle in Figure 2) is a rule-based agent that implements the Intelligent Driving Model (IDM) [25] behavior, which includes both a longitudinal decision policy and a lateral decision policy. The former computes the acceleration, given the ego's distance and speed; the latter decides when to change lane, to let the ego maximize its acceleration.

The first step ❶ of our approach trains the adversarial ADS, while the pre-trained ego ADS acts in inference mode. The training phase aims at teaching the adversarial ADS how to effectively challenge the ego ADS, increasing the chances of valid collisions, while minimizing the unavoidable ones. The second step ❷ resumes the training of the ego ADS in the presence of the adversarial ADS, which now acts in inference mode. Indeed, reinforcement learning enables continual learning, hence the possibility for the ego ADS to adapt to challenging situations. The two steps complement each other, forming a virtuous cycle the developers can leverage to improve the ego ADS.

A. Training the Adversarial ADS

In the first step of our approach, we bootstrap the training of the adversarial ADS by starting from the already trained ego ADS. In this way, the adversarial ADS does not need to learn how to drive well in the given driving scenario. On the other hand, the ego ADS acts in inference mode and its weights are frozen for the entire duration of the training process.

The adversarial ADS uses a custom reward function with two different components, namely a modified QoD reward $\hat{R}_{QoD}$ and the adversarial reward R_{adv}. The first component of the reward, encourages the adversarial ADS to keep the same behavior as the ego ADS. We slightly modify the original R_{QoD} (see Equation 2), by changing the weight of the R_{col} component, i.e., we set $w_{col} = 0$, in order not to discourage collisions.

The adversarial reward R_{adv} has two components, namely R_{diff}, that promotes the adversarial ADS to be close to the ego ADS, and R_{ep}, that rewards the adversarial ADS when it collides with the ego ADS. The reward difference component R_{diff} is defined as follows:

$$R_{diff} = \begin{cases} -D_{vx} - a, & \text{if } D_{vx} \geqslant 0; \\ \frac{-|D_{vy}|}{b}, & \text{if } D_{vy} \neq 0; \\ \eta(d), & \text{otherwise.} \end{cases} \quad (3)$$

where $D_{vx} = A_{vx} - E_{vx}$, i.e., the difference between the x components of the velocity of the adversarial ADS (i.e., A) and the ego ADS (i.e., E). Similarly, D_{vy} is the difference between the y components of the velocity of the two ADS. The distances between the positions are $D_x = A_x - E_x$ and $D_y = A_y - E_y$, respectively for the x and y axes, and $d = \sqrt{D_x^2 + D_y^2}$, i.e., the Euclidean distance between the two ADSs positions.

In the first case, the reward is negative as the adversarial ADS is getting far from the ego ADS, and hence far from a possible collision. In particular, when $D_{vx} \geqslant 0$, the adversarial ADS is getting far from the ego ADS in the longitudinal position. The constant a is a small positive constant that makes sure that the adversarial ADS is negatively rewarded even when $D_{vx} = 0$, as the two vehicles would have the same velocity, hence they would never collide.

In the second case, the reward function discourages lane changes, that occur when $D_{vy} \neq 0$. This is because a collision occurs when the two ADSs are on the same lane, while the ego ADS might overtake the adversarial ADS, making future collisions impossible, if the two vehicles are on different lanes. On the other hand, a lane change of the adversarial ADS might be crucial to cause a collision when the two vehicles are initially on different lanes. For this reason, the adversarial ADS is less penalized w.r.t. the first condition holding for

the longitudinal direction, as we divide $|D_{vy}|$ by a positive constant $b > 1$.

In the third case, the two ADSs are getting closer in the longitudinal direction and we positively reward the adversarial ADS with the inverse of the distance, i.e., $\eta(d) = \frac{1}{1+d}$ [26].

The second component of the adversarial reward is R_{ep}, which rewards the adversarial ADS if there is a collision with the ego ADS. This reward is episodic, i.e., it is given to the agent when the episode ends due to a collision. We define it as follows:

$$R_{ep} = \begin{cases} c \sum_t (\hat{R}_{QoD}(t) + R_{diff}(t)), & \text{if } R_{diff}(t) > 0 \ \forall t; \\ c \sum_t \hat{R}_{QoD}(t), & \text{otherwise.} \end{cases} \quad (4)$$

where the sum is considered over the number of timesteps t of a given episode, and c is a small constant in the interval $(0, 1]$. The episodic reward is proportional to the sum of $\hat{R}_{QoD}(t)$ and $R_{diff}(t)$, if $R_{diff}(t) > 0$ for each timestep t, while it is proportional to the sum of $\hat{R}_{QoD}(t)$ otherwise. In the initial phases of training, the adversarial ADS is exploring the possible strategies to cause a collision. As such R_{diff} will be occasionally negative throughout the episode. In this case, when a collision occurs, we reward the agent proportionally to its quality of driving, to encourage good driving behaviors. On the other hand, when the adversarial ADS has learned the right strategies for colliding with the ego ADS, the first condition holds and the episode reward takes into account the sum of the R_{diff} components for each timestep. In this case, we reward the adversarial ADS proportionally to how fast it gets close to the ego ADS before a collision occurs, as $\eta(d)$ gets higher across timesteps when d is rapidly reduced. In any case, we compute the episodic reward as a fraction of the aggregate return the agent receives for each episode, rather than using a constant value. In this way, we make sure that the bonus the agent receives is smoother and we avoid introducing sparse, large reward values, which may cause performance degradation during training [27].

At each timestep, the adversarial ADS receives the sum of the two scalar rewards, i.e., $R_{diff} + \hat{R}_{QoD}$. The two components of the reward are conflicting, as $\hat{R}_{QoD}$ encourages the adversarial ADS to drive well, while R_{diff} smoothly guides it to being close to the ego ADS, increasing the chances of a collision. Since the objective of the agent is to maximize the sum of the two scalar rewards, it needs to take into account $\hat{R}_{QoD}$, such that pure adversarial and possibly invalid behaviors incentivized by R_{diff} are discouraged. Indeed, pure adversarial behaviors might generate collisions that the ego ADS cannot avoid. Such collisions are not useful from the testing point of view, since they do not pinpoint any weakness of the ego ADS. The objective of our reward function is to induce valid collisions between the adversarial and the ego ADS, which the ego ADS could avoid by better training.

B. Adversarial Retraining

In the second step of our approach, we enable continual learning for the ego ADS in the presence of the adversarial ADS trained in the previous step. We reuse the same Quality of Driving reward R_{QoD} adopted in the pre-training phase and defined in Equation 2. On the other hand, the adversarial ADS acts in inference mode, choosing the action that maximizes its return $G(\tau)$.

V. Evaluation

To assess the practical benefits of our approach, we consider the following research questions:

RQ_1 (Failure Exposure): *How effective is the adversarial ADS compared to a random baseline?* RQ_1 aims at assessing how effective the adversarial ADS is at causing collisions with the ego ADS (i.e., how many failures it exposes). We compare the effectiveness of the adversarial ADS with an agent that samples the action of the ADS at random. In this research question, we evaluate the effectiveness of the adversarial reward function in guiding the adversarial ADS towards collisions.

RQ_2 (Fault Repair): *How effective is adversarial retraining in improving the ego ADS?* In RQ_2, we assess to what extent the ego ADS improves if retrained in the presence of the adversarial ADS. RQ_2 also analyzes how the ego ADS copes with the different behaviors of the adversarial ADS.

A. Procedure

Our experimental procedure consists of: (1) pre-training the ego ADS; (2) training the adversarial ADS and comparing its test time performance with that of a random agent (RQ_1); (3) resuming the training of the ego ADS in the presence of the adversarial ADS and evaluating the performance of the ego ADS at test time (RQ_2).

Pre-training the Ego ADS. We first trained the ego ADS using the DQN algorithm [12], a state-of-the-art DRL algorithm. In this pre-training phase, the vehicle in front (see Figure 2) is controlled by the IDM model. We used the hyperparameters provided by the *rl-agents* library [13] for the DQN algorithm, and we used $2k$ episodes as training budget. To save the best ego ADS model to be used in the testing phase, we considered the average reward over a window of 50 episodes.

To account for the randomness of the training process we trained the agent 10 times. In the testing phase, we loaded each ego ADS model, and we tested it by using the IDM rule-based agent controlling the vehicle in front. We tested each ego ADS instance 10 times for $2k$ episodes each. Finally, we chose as ego ADS the agent with the lowest average test failure rate.

RQ_1 (Failure Exposure). We used the ego ADS model as a starting point for the training of the adversarial ADS. We then trained the adversarial ADS controlling the vehicle in front 10 times with a training budget of $2k$ episodes, choosing as constants for the adversarial reward function $a = 0.01$, $b = 3$ (see Equation 3), and $c = 0.1$ (see Equation 4). We kept the hyperparameters of the DQN algorithm and the window size for saving the best agent unchanged.

In the testing phase, we load the ego ADS model and each instance of the adversarial ADS controlling the vehicle in front. For each instance we then executed 10 runs, each

TABLE I: Results for RQ_1 (Failure Exposure), and RQ_2 (Fault Repair). Bold-faced values indicate a statistically significant difference between the adversarial ADS and the random agent, while underlined values indicate that the magnitude of such difference (i.e., $\hat{A}_{12}$) is large.

	Pre-training			**RQ_1 (Failure Exposure)**							**RQ_2 (Fault Repair)**	
	Ego ADS			Adversarial ADS			Random		p-value	$\hat{A}_{12}$	Train FR	Avg Test FR
	Train FR	Avg Test FR	SEM	Train FR	Avg Test FR	SEM	Avg Test FR	SEM				
	0.03	0.00	0.00%	0.87	**0.92**	0.26%	0.39	0.30%	10^{-5}	1.00	0.10	0.00
	0.04	0.00	0.00%	0.89	**0.90**	0.22%	0.39	0.42%	10^{-5}	1.00	0.07	0.00
	0.03	0.00	0.00%	0.94	**1.00**	0.04%	0.39	0.36%	10^{-5}	1.00	0.15	0.00
	0.05	0.00	0.00%	0.86	**0.92**	0.23%	0.39	0.30%	10^{-5}	1.00	0.02	0.00
	0.07	0.00	0.00%	0.93	**1.00**	0.00%	0.39	0.34%	10^{-5}	1.00	0.17	0.00
	0.1	0.00	0.00%	0.89	**1.00**	0.00%	0.41	0.27%	10^{-5}	1.00	0.04	0.00
	0.2	0.00	0.00%	0.91	**0.90**	0.17%	0.39	0.38%	10^{-5}	1.00	0.14	0.00
	0.11	0.01	0.06%	0.90	**0.93**	0.12%	0.39	0.31%	10^{-5}	1.00	0.03	0.00
	0.11	0.00	0.00%	0.91	**0.93**	0.21%	0.39	0.57%	10^{-5}	1.00	0.28	0.00
	0.11	0.09	0.26%	0.97	**1.00**	0.00%	0.38	0.30%	10^{-5}	1.00	0.05	0.00
Average	0.09	0.01	0.03%	0.91	0.95	0.13%	0.39	0.35%	10^{-5}	1.00	0.11	0.00
SEM	—	0.92%	—	—	1.37%	—	0.17%	—	—	—	—	—

lasting $2k$ episodes. To build a baseline for comparison, we replaced the adversarial ADS instances with a *random agent* controlling the vehicle in front (i.e., an agent that chooses uniformly at random among the 5 available discrete actions in each timestep). We executed the random agent 10 times for $2k$ episodes each, to account for its randomness. We repeated the process 10 times to compare each random agent's execution with each instance of the adversarial ADS.

Each test episode in the testing phase consists of a different, randomly generated, test scenario. A test scenario in the considered environment is defined by the initial state of the ego vehicle and of the front (adversarial) vehicle, where each state consists of the vehicle's x, y positions, with y determining the lane, as well as the vehicle's v_x, v_y longitudinal and vertical components of the initial speed. Moreover, the longitudinal position of the vehicle in front is constrained to be greater than the longitudinal position of the ego vehicle by some margin Δx (in our experiments we randomly sample the initial separation Δx between vehicles in the range $[0.5, 2]$ meters).

RQ_2 (Fault Repair). To resume training of the ego ADS, we considered the adversarial ADS model with the highest average test failure rate. Then, we carried out retraining of the ego ADS 10 times for $2k$ episodes each, with the adversarial ADS controlling the vehicle in front. Also in this case, we kept the hyperparameters of the DQN algorithm and the window size for saving the best agent unchanged. Finally, we tested each retrained ego ADS 10 times for $2k$ episodes each.

B. Metrics

RQ_1 (Failure Exposure). To answer RQ_1, we measured the test failure rates of the adversarial ADS and the random agent across $2k$ episodes for each run. Regarding the adversarial ADS, we computed the standard error of the mean (SEM) of the test failure rate, to evaluate both its stability across training runs and its stability across repetitions of the test process, which is also affected by non-determinism, as at testing time we generate $2k$ test scenarios randomly. We used a 5% threshold to determine whether the test failure rate is stable [28] across runs/test executions. We rigorously compared the two failure rates using the Mann-Whitney U test [29] (with a significance level $\alpha = 0.05$), and the Vargha-Delaney effect size [30] to assess the magnitude of the difference, as previous literature suggests [31].

RQ_2 (Fault Repair). To answer RQ_2, we measured the test failure rates of each ego ADS instance and computed the SEM to analyze whether the test failure rate is stable across runs/test executions. Also in this case we used a 5% threshold.

C. Results

Pre-training the Ego ADS. Columns 2–4 of Table I show the results of the pre-training phase of the Ego ADS. Column 2 reports the training failure rate for each of the 10 runs. For each instance, we measured the training failure rate as the average failure rate across the last 100 training episodes. Across the 10 runs the training failure rate is, on average, 0.09. Column 3 shows the average test failure rate for each ego ADS instance, across 10 testing executions of $2k$ episode each. The average test failure rate ranges from a minimum of 0.0 to a maximum of 0.09, with an average of 0.01. The average SEM is 0.03% (Column 4) across test executions, with the highest being 0.26%. Across training runs (last row), SEM is also remarkably low (0.92%), indicating that the RL training

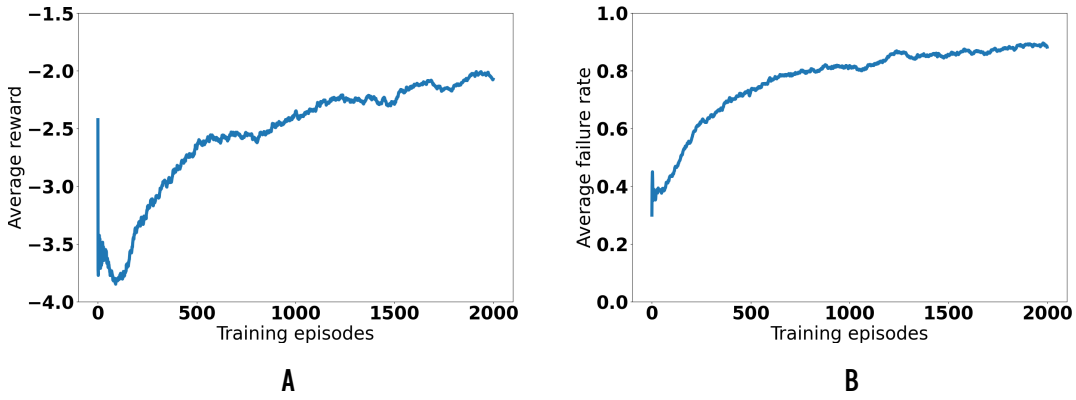

Fig. 4: Trend of the average cumulative reward (**A**) and of the average failure rate, during $2k$ training episodes. The two metrics are averaged across 10 training runs.

process for the ego ADS converges to a stable test-time failure rate.

It should be noticed that a lower failure rate at testing than at training time is expected with reinforcement learning (while the opposite happens with supervised deep learning), because the RL agent finds the optimal policy by exploration (trial and error), not just by weight optimization, which makes the learning process less linear. Often the optimal agent is not the one observed in the last episode, as the exploration promoted by RL can also introduce regressions. The average SEM is 0.03% (Column 4), with the highest being 0.26%.

Overall, by looking at the test failure rate we conclude that the ego ADS is well-trained. Moreover, an average SEM lower than 5% indicates that the training process is stable, i.e., that it produces ego ADS instances with similar test time performance across multiple test executions.

RQ_1 (Failure Exposure). The subplot **A** of Figure 4 shows the growth curve for the reward value across the episodes of adversarial training. Subplot **B** shows the trend of the training failure rate over time, which indicates the adversarial vehicle's ability to induce collisions. The trends of the two plots show that our reward function is effective and smoothly guides the learning process of the adversarial agent.

Columns 5–11 of Table 1 show the results for the first research question. Across 10 runs the average training failure rate is 0.91 (Column 5), with a minimum of 0.87, and a maximum of 0.97. This indicates that the training process is effective and the adversarial ADS learns how to make the ego vehicle fail. Column 6 shows the average test failure rate for each adversarial ADS instance. We observe that testing failure rates, on average, are higher than training failure rates, as the adversarial DQN agent is deterministic at testing time and exploits the best actions at each timestep. Comparing the average test failure rates of the adversarial ADS with those obtained with the random agent (Column 8), we see that the former achieves an average test failure rate of 0.95 across 10 runs, while the latter does not go above 40% (i.e., 0.39). For each pair of agents adversarial-random, we computed the p-value (Column 10) and the effect size $\hat{A}_{12}$ (Column 11). Results show that each adversarial ADS instance has a significantly higher average test failure rate, with a large effect size, than random. Moreover, the average SEM for the average test failure rate of the adversarial ADS is 0.13% (Column 7), with a maximum of 0.26%, across test executions; it is 1.37% across adversarial training runs. This indicates that adversarial training is stable (i.e., SEM $< 5\%$), producing adversarial ADS instances with similar test time performance across multiple runs. The SEM for Random is also low across repeated executions (i.e., 0.17%), mostly because the number of test executions (i.e., $2k$) is high enough to stabilize the test failure rate across 10 repetitions. Hence, we can conclude that our estimations of the failure rates both for the adversarial ADS and for Random are reliable and provide a solid ground for our comparative analysis.

> **RQ_1 (Failure Exposure)**: The adversarial ADS is effective at generating collisions with the ego ADS at test time. Compared to the random agent, it is significantly more effective with a large effect size. Moreover, the training of the adversarial ADS is stable across multiple runs, as it produces agents with similar test failure rate.

Qualitative analysis. Figure 5 shows two examples of collisions caused by the adversarial ADS. The subplot **A**, shows the adversarial ADS mimicking the behaviors of the ego ADS. In particular, both the ego and the adversarial ADS change lane at the same time. The ego ADS is not able to react quickly, by either changing lane or slowing down, to avoid the collision with the adversarial ADS in front. On the other hand, in the subplot **B**, the ego ADS keeps its lane, while the adversarial ADS changes it (from bottom lane to top lane). Also in this case the ego ADS is not able to slow down in time and cannot prevent the collision.

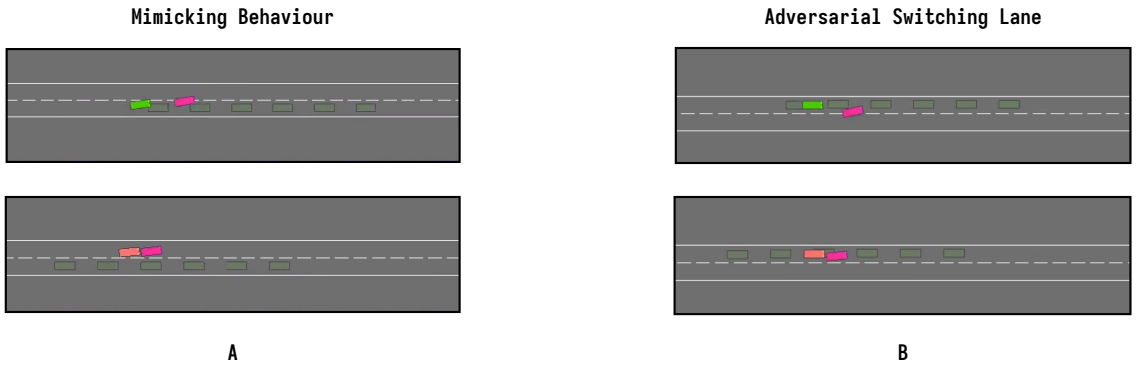

Fig. 5: Two collision patterns triggered by the adversarial ADS at testing time. The ego ADS is shown in green, while the adversarial ADS in front is shown in magenta. When the ego ADS collides, it changes color to orange.

RQ_2 (Fault Repair). Columns 12–13 of Table I show the results of the retraining process in terms of training failure rate (Column 12) and average test failure rate (Column 13). The training failure rate across the 10 runs is 0.11 on average, drastically reducing the failure rate observed during training of the adversarial ADS (i.e., 0.91 on average). Interestingly, the average failure rate is 0.0 at test time, for each instance of the retrained ego ADS.

We analyzed the reasons for this behavior by tracking the longitudinal component of the velocity of the ego and the adversarial ADSs in three scenarios: (1) after the pre-training phase of the ego ADS, (2) after step ❶, i.e., the training of the adversarial ADS, and (3) after step ❷, i.e., the retraining of the ego ADS in the presence of the trained adversarial ADS. For this analysis, we considered the best agents in each phase, i.e., those with the best test failure rate, and we tracked the velocities for 100 testing episodes. For each episode, we set the initial velocity at $25\,m/s$, instead of sampling it at random, to ease the visualization of the trend of the velocity over time. Moreover, since episodes might have different lengths, we padded episodes shorter than 30 timesteps, i.e., the maximum duration, with the last velocity value registered in the respective episodes. This is consistent with the reward function used to train the ego ADS, which promotes high speed and high quality of driving.

Figure 6 shows the longitudinal velocity trends in the three scenarios described above (respectively, subplots **A**, **B**, and **C**). The solid line represents the average velocity across 100 episodes, while the shaded contours represent the standard deviation. After the pre-training phase (i.e., subplot **A**), we observe that the ego ADS maximizes its velocity almost immediately after the beginning of the episode. The behavior is the same for all the runs, as the ego ADS reaches the maximum velocity of $30\,m/s$ after 4 timesteps on average.

After the training of the adversarial ADS (i.e., subplot **B**), the ego ADS initially increases its velocities in the first timesteps, while the adversarial ADS decreases it to collide with the ego ADS. The episodes are on average very short, i.e., around 3 timesteps, as the adversarial ADS ends the episode by causing a collision. The velocities of both vehicles go below the minimum, i.e., $20\,m/s$, as the last velocities values are recorded at the moment of the collision. Interestingly, the velocity of the adversarial ADS is, on average, lower than that of the ego ADS when there is a collision. This indicates some (unsuccessful) attempt to avoid the collision by reducing the speed of the vehicle, trying to keep it below the speed of the adversarial ADS.

After resuming the training of the ego ADS in the presence of the adversarial ADS (i.e., subplot **C**), the adversarial ADS has the same or similar behaviors as before, as during retraining of the ego ADS its weights are frozen. On the other hand, the ego ADS has learned that, in order to avoid collisions, it has to decrease its velocity to the minimum. In this way, the two vehicles will never collide, since they have the same velocity and are separated in the longitudinal/latitudinal direction. Indeed, all episodes last 30 timesteps on average with a zero standard deviation.

From a qualitative point of view, the strategy learned by the ego ADS to avoid collisions in the presence of adversarial vehicles, consists of slowing down in order to keep a safe distance from the adversarial ADS, or *"drive slow, drive safe"*.

> **RQ_2 (Fault Repair)**: Resuming training of the ego ADS in the presence of a trained adversarial ADS is beneficial to improve the robustness of the ego ADS. In particular, retraining induces a cautious driving behavior, consisting of reducing the velocity to keep a safe distance and avoid collisions with the adversarial ADS.

Discussion. Our retraining experiment shows that the overall approach is effective, and that the ego ADS can learn a strategy to deal with the presence of an adversarial vehicle that intentionally challenges it. However, one might question the likelihood of encountering such adversarial behaviors in real driving conditions, especially because the cost to avoid adversarial collisions is a substantial reduction of the ego vehicle's speed when the front vehicle slows down, which

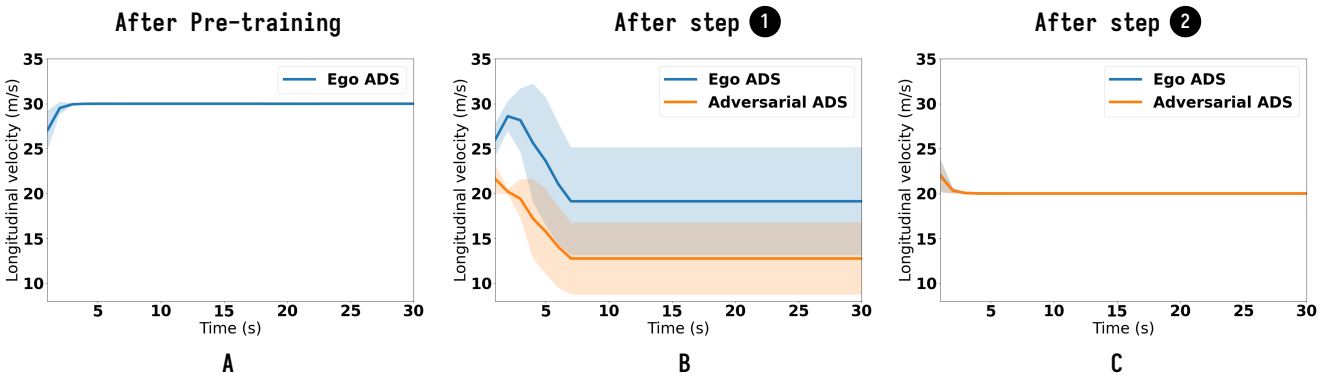

Fig. 6: Longitudinal velocity over time. The leftmost plot (i.e., **A**), shows the velocity of the ego ADS after the pre-training phase. The center plot (i.e., **B**) shows the velocities of the ego (in blue) and adversarial ADSs (in orange), after step ❶, i.e., the training of the adversarial ADS. Likewise, the rightmost plot (i.e., **C**) shows the velocities of the ego and adversarial ADSs after step ❷, i.e., the retraining of the ego ADS in the presence of the adversarial ADS.

might be seen as a reduction of the quality of driving, if speed is kept very low for no reason.

To accommodate different trade-offs between safety and speed of driving, one might play with the quality of driving reward (see Equation 2). Indeed, the parameter w_{col} can be seen as the ego vehicle's risk aversion. In our experiments, we set it to -1. A lower absolute value of this parameter (e.g., -0.5 or -0.1) could be used to induce a higher risk propensity, which means the ego vehicle would attempt to drive faster, accepting that occasionally a collision may happen in the presence of strongly adversarial behaviors. If such behaviors are deemed as very unlikely, slightly increasing the risk propensity of the ego ADS might be acceptable. We intend to conduct experiments in this direction in our future work, especially because they might trigger novel strategies to deal with adversarial behaviors.

D. Threats to Validity

External Validity. Using one driving simulator and one ADS, may pose an external validity threat. To mitigate this issue, we used the state-of-the-art DQN algorithm [12] to obtain a trained ADS representative of the state of the art in the usage of RL for autonomous driving. As driving simulator, we selected *HighwayEnv* [17] which is extensively used in reinforcement learning research, as it supports multi-agent training and testing [18], [19], [20], [21], [22], [23], [24].

Internal Validity. We extensively tested our tool and our test scripts to reduce the risk that bugs affecting our code could produce wrong results.

Conclusion Validity. Random variations in the execution of our experiments may pose threats to conclusion validity. To mitigate this issue, we executed multiple runs of training and testing to account for the randomness of the DRL algorithm, as well as the initial states of the vehicles in the driving simulator. The comparison between our approach and the random baseline is also affected by non-determinism. We mitigated this threat, by executing the two agents under identical conditions, and with the same testing budget (in terms of number of episodes). We also measured SEM to assess the stability of the measured failure rates, and we conducted statistical tests to corroborate our conclusions about the existence of a significant difference between our approach and the random baseline.

Reproducibility. To support reproducibility, we make our source code available online [32]. Moreover, we only used open-source tools and subjects in our evaluation.

VI. Related Work

Several search-based techniques have been proposed in the literature to test autonomous driving systems (ADSs) [1], [33], [34], [35]. ASFault [5] uses search-based algorithms combined with procedural content generation to generate virtual roads. Abdessalem et al. [6] generate critical scenarios by using decision trees to guide the search towards critical features. Riccio et al. propose DeepJanus [2] whose objective is to find road configurations at the frontier of behaviors of the ADS under test. Moreover, they propose DeepHyperion [3], [36] that characterizes the road shapes based on both their structural features and the behavioral features of the ADS under test. Calò et al. [37] propose two search-based algorithms to generate test scenarios that induce avoidable collisions. On the other hand, Samota [38] uses a surrogate model of the environment to speed up the generation of critical test scenarios. However, such approaches are effective when generating static configurations of the environment (e.g., road shapes). In our work, our testing agent is an adversarial ADS controlling the vehicle in front rather than a generator of static configurations. In such situations, Reinforcement Learning (RL) is more suitable to control the adversarial ADS at runtime to react to the actions of the ego ADS.

Regarding the application of RL to ADS testing, DeepCollision [7] and Morlot [8] model the state space of the RL agent as a set of continuous variables describing the ADS under test and its operating environment. Such continuous variables include, for example, the locations of the ADS

and of the other vehicles, the pedestrians, and the state of the traffic lights. The action space of the agent is discrete, acting on the dynamics of the Non-Playable Characters (NPCs) in the environment such as pedestrians and other vehicles. Regarding the reward function, DEEPCOLLISION rewards the agent by computing, at each step, the probability of collision between the ADS under test and the vehicle in front, while MORLOT aims to find violations of multiple safety and functional requirements, modeling each of them as a separate reward function. In both DEEPCOLLISION and MORLOT, the RL agent controls multiple actors in the environment, to maximize the reward function. However, a single, centralized entity controls the environment and all its NPCs. In real-world driving conditions, each NPC acts independently, and it is characterized by its own autonomous behavior. Indeed, a single agent controlling the whole environment may result in NPCs with possibly unrealistic or even invalid behaviors. In our work, we model the NPC vehicle-in-front as a fully autonomous and independent agent. We train such agent with an adversarial reward function to challenge the ego ADS under test. Moreover, we close the evaluate-train loop, by resuming training of the ego ADS in the presence of the adversarial ADS, to increase its robustness.

The idea of introducing an adversarial agent at testing time is not new: in a preliminary work, Sharif et al. [39] used an adversarial RL agent to induce collisions at road intersections. Their adversarial agent is guided by a binary reward function, giving a positive reward only when collisions occur. In our setting, we found that a smoother reward function, such as ours, is an essential component of the adversarial ADS. In their setting [39], the vehicle controlled by the adversarial agent is not present during the training of the ego ADS, which makes collisions much more likely when such vehicle is introduced, and might explain the effectiveness of a binary reward. On the contrary, our setting is more realistic as we include a front vehicle even when training the ego ADS, making the task of inducing collisions more challenging for the adversarial ADS. Moreover, their evaluation of the adversarial ADS [39] is quite preliminary, being based on a single observation with no baseline.

Ours is the first work where adversarial ADSs are (1) proposed as an integral part of a virtuous evaluate-train loop, and (2) evaluated systematically with a solid empirical study, taking into account the statistical variability of training DRL agents [40]. Ours is the first paper providing empirical and statistical evidence about the viability and practical usefulness of adversarial test agents for the improvement of autonomous vehicles.

VII. CONCLUSION AND FUTURE WORK

In this paper, we proposed an adversarial approach to test autonomous driving systems (ADSs). Our approach focuses on training the adversarial ADS with a custom reward function that balances quality of driving and chances of collisions, in order to avoid pure adversarial behaviors. The second step consists of improving the ego ADS by retraining it in the presence of the adversarial ADS. Results show that the adversarial ADS is significantly more effective than a random baseline at causing collisions with the ego ADS. Moreover, adversarial retraining induces a safe driving behavior in the ego ADS, that lets it avoid the collisions the adversarial ADS would otherwise cause.

In our future work, we plan to explore other driving scenarios and to investigate cooperation strategies among multiple adversarial agents, which work together to challenge and eventually improve the ego ADS under test. We will also investigate different parameters of the reward function used during retraining to introduce a higher risk propensity into the ego ADS. The purpose is to explore the trade-off between safety and speed of driving, and to analyze whether different strategies emerge to deal with adversarial behaviors while keeping a higher speed.

REFERENCES

[1] S. Tang, Z. Zhang, Y. Zhang, J. Zhou, Y. Guo, S. Liu, S. Guo, Y.-F. Li, L. Ma, Y. Xue *et al.*, "A survey on automated driving system testing: Landscapes and trends," *ACM Transactions on Software Engineering and Methodology*, 2023.

[2] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 876–888.

[3] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, "Deephyperion: exploring the feature space of deep learning-based systems through illumination search," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 79–90.

[4] G. Li, Y. Li, S. Jha, T. Tsai, M. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. Iyer, "Av-fuzzer: Finding safety violations in autonomous driving systems," in *2020 IEEE 31st international symposium on software reliability engineering (ISSRE)*. IEEE, 2020, pp. 25–36.

[5] A. Gambi, M. Müller, and G. Fraser, "Automatically testing self-driving cars with search-based procedural content generation," in *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis, ISSTA*, 2019, pp. 318–328.

[6] R. Ben 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)*, May 2018, pp. 1016–1026.

[7] C. Lu, Y. Shi, H. Zhang, M. Zhang, T. Wang, T. Yue, and S. Ali, "Learning configurations of operating environment of autonomous vehicles to maximize their collisions," *IEEE Transactions on Software Engineering*, vol. 49, no. 1, pp. 384–402, 2022.

[8] F. U. Haq, D. Shin, and L. C. Briand, "Many-objective reinforcement learning for online testing of dnn-enabled systems," in *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2023, pp. 1814–1826.

[9] F. M. Favarò, N. Nader, S. O. Eurich, M. Tripp, and N. Varadaraju, "Examining accident reports involving autonomous vehicles in california," *PLoS one*, vol. 12, no. 9, p. e0184952, 2017.

[10] N. H. T. S. Administration *et al.*, "Summary report: Standing general order on crash reporting for automated driving systems," 2022.

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

[12] 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.

[13] E. Leurent, "rl-agents: Implementations of reinforcement learning algorithms," https://github.com/eleurent/rl-agents, 2018.

[14] A. Raffin, A. Hill, A. Gleave, A. Kanervisto, M. Ernestus, and N. Dormann, "Stable-baselines3: Reliable reinforcement learning implementations," *Journal of Machine Learning Research*, vol. 22, no. 268, pp. 1–8, 2021. [Online]. Available: http://jmlr.org/papers/v22/20-1364.html

[15] S. Huang, R. F. J. Dossa, C. Ye, J. Braga, D. Chakraborty, K. Mehta, and J. G. Araújo, "Cleanrl: High-quality single-file implementations of deep reinforcement learning algorithms," *Journal of Machine Learning Research*, vol. 23, no. 274, pp. 1–18, 2022. [Online]. Available: http://jmlr.org/papers/v23/21-1342.html

[16] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica, "RLlib: Abstractions for distributed reinforcement learning," in *International Conference on Machine Learning (ICML)*, 2018.

[17] E. Leurent, "An environment for autonomous driving decision-making," https://github.com/eleurent/highway-env, 2018.

[18] M. Xu, P. Huang, F. Li, J. Zhu, X. Qi, K. Oguchi, Z. Huang, H. Lam, and D. Zhao, "Accelerated policy evaluation: Learning adversarial environments with adaptive importance sampling," *arXiv preprint arXiv:2106.10566*, 2021.

[19] H. Sun, S. Feng, X. Yan, and H. X. Liu, "Corner case generation and analysis for safety assessment of autonomous vehicles," *Transportation research record*, vol. 2675, no. 11, pp. 587–600, 2021.

[20] D. Chen, Z. Li, Y. Wang, L. Jiang, and Y. Wang, "Deep multi-agent reinforcement learning for highway on-ramp merging in mixed traffic," *arXiv preprint arXiv:2105.05701*, 2021.

[21] S. Feng, X. Yan, H. Sun, Y. Feng, and H. X. Liu, "Intelligent driving intelligence test for autonomous vehicles with naturalistic and adversarial environment," *Nature communications*, vol. 12, no. 1, pp. 1–14, 2021.

[22] B. Brito, A. Agarwal, and J. Alonso-Mora, "Learning interaction-aware guidance policies for motion planning in dense traffic scenarios," *arXiv preprint arXiv:2107.04538*, 2021.

[23] S. Zhang, L. Wen, H. Peng, and H. E. Tseng, "Quick learner automated vehicle adapting its roadmanship to varying traffic cultures with meta reinforcement learning," in *2021 IEEE International Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2021, pp. 1745–1752.

[24] B. Eysenbach, R. R. Salakhutdinov, and S. Levine, "Robust predictable control," *Advances in Neural Information Processing Systems*, vol. 34, pp. 27 813–27 825, 2021.

[25] M. Treiber, A. Hennecke, and D. Helbing, "Congested traffic states in empirical observations and microscopic simulations," *Physical review E*, vol. 62, no. 2, p. 1805, 2000.

[26] G. Fraser and A. Arcuri, "Whole test suite generation," *IEEE Trans. Software Eng.*, vol. 39, no. 2, pp. 276–291, 2013. [Online]. Available: https://doi.org/10.1109/TSE.2012.14

[27] S. Huang, S. Ontañón, C. Bamford, and L. Grela, "Gym-μrts: Toward affordable full game real-time strategy games research with deep reinforcement learning," in *2021 IEEE Conference on Games (CoG), Copenhagen, Denmark, August 17-20, 2021*. IEEE, 2021, pp. 1–8. [Online]. Available: https://doi.org/10.1109/CoG52621.2021.9619076

[28] N. Humbatova, G. Jahangirova, and P. Tonella, "Deepcrime: mutation testing of deep learning systems based on real faults," in *ISSTA '21: 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, Virtual Event, Denmark, July 11-17, 2021*, C. Cadar and X. Zhang, Eds. ACM, 2021, pp. 67–78. [Online]. Available: https://doi.org/10.1145/3460319.3464825

[29] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[30] 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.

[31] A. Arcuri and L. C. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Test. Verification Reliab.*, vol. 24, no. 3, pp. 219–250, 2014. [Online]. Available: https://doi.org/10.1002/stvr.1486

[32] "Replication package." https://www.dropbox.com/scl/fo/d6wqo7vgu1532c5jzm76m/h?rlkey=k1h1x44r6d17cdfh5q6vs02hz&dl=0, 2023.

[33] S. Panichella, A. Gambi, F. Zampetti, and V. Riccio, "SBST tool competition 2021," in *14th IEEE/ACM International Workshop on Search-Based Software Testing, SBST 2021, Madrid, Spain, May 31, 2021*. IEEE, 2021, pp. 20–27. [Online]. Available: https://doi.org/10.1109/SBST52555.2021.00011

[34] A. Gambi, G. Jahangirova, V. Riccio, and F. Zampetti, "SBST tool competition 2022," in *15th IEEE/ACM International Workshop on Search-Based Software Testing, SBST@ICSE 2022, Pittsburgh, PA, USA, May 9, 2022*. IEEE, 2022, pp. 25–32. [Online]. Available: https://doi.org/10.1145/3526072.3527538

[35] M. Biagiola, S. Klikovits, J. Peltomäki, and V. Riccio, "SBFT tool competition 2023 - cyber-physical systems track," in *IEEE/ACM International Workshop on Search-Based and Fuzz Testing, SBFT@ICSE 2023, Melbourne, Australia, May 14, 2023*. IEEE, 2023, pp. 45–48. [Online]. Available: https://doi.org/10.1109/SBFT59156.2023.00010

[36] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, "Efficient and effective feature space exploration for testing deep learning systems," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 2, pp. 1–38, 2023.

[37] 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)*. IEEE, 2020, pp. 375–386.

[38] F. U. Haq, D. Shin, and L. Briand, "Efficient online testing for dnn-enabled systems using surrogate-assisted and many-objective optimization," in *Proceedings of the 44th International Conference on Software Engineering*, 2022, pp. 811–822.

[39] A. Sharif and D. Marijan, "Adversarial deep reinforcement learning for improving the robustness of multi-agent autonomous driving policies," in *29th Asia-Pacific Software Engineering Conference, APSEC 2022, Virtual Event, Japan, December 6-9, 2022*. IEEE, 2022, pp. 61–70. [Online]. Available: https://doi.org/10.1109/APSEC57359.2022.00018

[40] P. Henderson, R. Islam, P. Bachman, J. Pineau, D. Precup, and D. Meger, "Deep reinforcement learning that matters," in *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, S. A. McIlraith and K. Q. Weinberger, Eds. AAAI Press, 2018, pp. 3207–3214. [Online]. Available: https://doi.org/10.1609/aaai.v32i1.11694