EMSE manuscript No.
(will be inserted by the editor)

Reinforcement Learning for Online Testing of Autonomous Driving Systems: a Replication and Extension Study

Luca Giamattei · Matteo Biagiola · Roberto Pietrantuono · Stefano Russo · Paolo Tonella

Accepted: 4 October 2024

https://doi.org/10.1007/s10664-024-10562-5

Abstract In a recent study, Reinforcement Learning (RL) used in combination with many-objective search, has been shown to outperform alternative techniques (random search and many-objective search) for online testing of Deep Neural Network-enabled systems. The empirical evaluation of these techniques was conducted on a state-of-the-art Autonomous Driving System (ADS). This work is a replication and extension of that empirical study. Our replication shows that RL does not outperform pure random test generation in a comparison conducted under the same settings of the original study, but with no confounding factor coming from the way collisions are measured. Our extension aims at eliminating some of the possible reasons for the poor performance of RL observed in our replication: (1) the presence of reward components providing contrasting feedback to the RL agent; (2) the usage of an RL algorithm (Q-learning) which requires discretization of an intrinsically continuous state space. Results show that our new RL agent is able to converge to an effective policy that outperforms random search. Results also highlight other possible improvements, which open to further investigations on how to best leverage RL for online ADS testing.

Luca Giamattei*, Roberto Pietrantuono, Stefano Russo
Università di Napoli Federico II
Via Claudio 21, 80125 – Napoli, Italy
Phone: +39 0817683820 – Fax: +39 0817683816
E-mail: {luca.giamattei, roberto.pietrantuono, stefano.russo}@unina.it
**corresponding author*

Matteo Biagiola, Paolo Tonella
Università della Svizzera Italiana
Via Buffi, 13 – Lugano, Switzerland
Phone: +41 58 666 40 00 – Fax: +41 58 666 46 47
E-mail: {matteo.biagiola, paolo.tonella}@usi.ch

Keywords Reinforcement Learning · Autonomous Driving Systems · Online Testing · Replication Study · Extension Study

1 Introduction

Testing Deep Neural Network-enabled systems is a challenging and expensive task, yet essential in the engineering of many modern systems with artificial intelligence components at their core. Testing of Autonomous Driving Systems (ADSs) has gained particular attention in the scientific community, as devising more effective and efficient techniques increases safety and reduces costs.

In the literature, a common way to test ADSs is to manipulate the simulation environment where they operates [36]. The objective of such techniques is to perturb the environment to try to cause a misbehavior of the ADS, e.g., a collision with another vehicle or a traffic rule violation. Tests are generated automatically by solving an optimization problem, which consists of finding the optimal configurations of the objects in the environment to optimize an objective function (usually the distance of the ADS from misbehavior). Researchers have proposed search-based techniques to address this optimization problem [3, 11, 21, 29, 36, 42], showing their effectiveness at generating static configurations of the environment that challenge the ADS under test.

However, search-based techniques struggle to deal at runtime with sequential interactions, required to manipulate dynamic objects in the environment (e.g., another vehicle). The Reinforcement Learning (RL) paradigm requires the agent to dynamically interact with the environment, learning from the effects of its actions. This offers an alternative way to test ADSs, by formulating the testing problem as an RL problem. The testing technique needs to choose a suitable RL algorithm to learn the actions that maximize the reward.

In an article presented in 2023 at the International Conference on Software Engineering (ICSE), Haq *et al.* [13] proposed Many-Objective Reinforcement Learning for Online Testing (MORLOT), an online testing technique that combines RL and many-objective search to test the ADS module of an autonomous vehicle. MORLOT was evaluated in the CARLA simulation environment [7], a widely used high-fidelity driving simulator [36]. The ADS under test was the TransFuser model [28], the highest ranked ADS in the CARLA leaderboard [17] at the time of that study. The evaluation shows that MORLOT outperforms random testing as well as state-of-the-art search-based techniques, in terms of safety requirements violations exposed in a given time budget.

This work is a replication and extension of the one by Haq *et al* [13]. Replication of past studies is a fundamental aspect of the scientific method [22], to validate (or not) their findings, and to generalize them by establishing different conditions under which they hold. It is essential in software engineering research [32], just as in "the construction of knowledge in any empirical science" [33]. Numerous replication studies have been conducted in this field [6, 24, 27, 34, 39], increasing confidence and providing extensions of original works. With this goal, we first present an *exact replication* [32] of the work

by Haq *et al* [13]; then we extend it to investigate the conditions under which RL is actually beneficial in ADS testing. The contribution is twofold:

Replication. The replication does not confirm the finding that many-objective Reinforcement Learning, specifically MORLOT, outperforms the random baseline in ADS testing. We reproduce the experiments of Haq *et al.*, showing that, if MORLOT and random are compared in the same conditions, they are statistically indistinguishable. We also analyze the design choices in Haq *et al.* formulation of ADS testing as an RL problem, and discuss how they reduce the learning capability of the RL agent, ultimately making the learning process ineffective with the given time budget.

Extension. In the extension study we show that, by formulating the testing problem as single-objective, a deep RL agent converges to an effective policy in most testing scenarios, and significantly outperforms the random baseline. The results of the extension study highlight that RL is a promising framework for testing highly dynamic systems such as ADSs, but further research is needed to address the limitations of the current formulation.

The paper is structured as follows. Section 2 provides background and basic definitions. Section 3 describes the study replicating the work by Haq *et al.* [13], while Section 4 presents the extension. Section 5 discusses the related work and highlights the novelty of the contributions. Finally, Section 6 provides concluding remarks.

2 Background

Reinforcement Learning (RL) is the process of learning what to do (i.e., how to relate circumstances to actions) in order to maximize a reward [35]. The *learner* (a.k.a. *agent*) is not instructed on actions to take, but, interacting with its *environment*, explores which actions produce the highest reward.

The RL process can be formalized through a Markov Decision Process (MDP), a classical model for sequential decision-making, where actions do not influence just immediate reward, but also the following states. An MDP is defined by a tuple (S, A, P, R, γ), where: S and A are respectively the sets of states and actions; P is the state transition probability function $P(s_{t+1}|s_t, a_t)$, assigning the probability of state s_{t+1} at time step $t+1$, given state s_t and action a_t at time step t; the reward function $R : S \times A \to \mathbb{R}$ maps a state-action pair to the set of real values; $\gamma \in [0, 1]$ is the discount factor controlling the trade-off between future and immediate rewards.

At time step t, the agent observes the state of the environment s_t, and selects an action a_t based on its *policy* π. The policy is generally a stochastic function $\pi : S \to A$ that yields the probability of selecting action $a_t \in A$ in state $s_t \in S$ at step t. At step $t+1$ the environment outputs the next state s_{t+1} and a scalar value r_{t+1}, rewarding the goodness of action a_t. The reward is the learning signal, that the agent aims to maximize. Through the interactions with the environment, the agent learns an *optimal policy* π^*, that maximizes

the total expected reward the agent gets in its lifetime (the expectation accounts for the randomness of both the transition probability function of the environment and the policy). The RL methods in the literature differ in how they update the agent's policy as further experience becomes available.

The most common RL algorithms are *model-free* (not equipped with a model of the environment). Within this family, RL algorithms can be categorized into *value-based*, *policy-based*, or a combination of the two. Value-based algorithms learn a *value function* giving an estimate of "how promising" a state (or a state-action pair) is. The estimate is computed as total expected reward through a *state-value function* $V(s)$ (or an *action-value function* $Q(s, a)$). The policy is then built by choosing in each state the action that maximizes the value function. Policy-based methods (e.g., policy-gradient) maximize the total expected reward by finding a policy through stochastic gradient ascent with respect to the policy parameters (e.g., the parameters of a neural network).

One of the most important value-based algorithms is *Q-learning*, proposed in 1989 by Watkins [40]. It learns an action-value function $Q(s, a)$, updated as new data becomes available. The agent's knowledge is represented as a table (named *Q-table*) mapping states and actions to the expected reward. At each time step, the agent starts from state s_t, selects the action with the highest Q-value ($\underset{a \in A}{max} Q(s_t, a)$), enters next state s_{t+1}, and collects the reward r_{t+1}. Finally, it updates the Q-value of the starting state-action pair (s_t, a_t) as:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \underset{a \in A}{max} Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

where: γ is the discount factor and $\alpha \in [0, 1]$ is the learning rate, controlling the step size at which the Q-values are updated.

Advances in deep learning have led to the development of deep RL (DRL) algorithms like *Deep Q-Network* (DQN) [26]. DQN combines the *Q-learning* paradigm with a neural network that receives a state as input, and approximates the Q-values for each potential action as output. The neural network replaces the Q-table and concisely stores the agent's experience, handling large state spaces such as continuous ones. To enhance stability during training, DQN typically utilizes a buffer of past experiences. During training, it randomly samples batches of experiences to update the weights of the neural network. Additionally, for stabilization DQN uses an auxiliary network (also called target network), a copy of the network being trained. The weights of such network are kept frozen for a certain number of training steps, so that the original network is trained with a fixed target. DQN handles continuous state spaces, but still requires actions to be discrete, as its update rule requires a maximization over all the actions for a particular state.

Both Q-learning and DQN use a behavior policy in training to explore the environment and look for actions that lead to high reward. A common policy is ϵ-*greedy*, the parameter ϵ representing the probability of choosing a random action instead of relying on the Q-value function (respectively a table and a neural network). Typically, ϵ is set to 1 at the beginning of training, so

Table 1 Safety and functional requirements for the ADS under test in Haq *et al.* [13].

Requirement	Description	Reward
R1	EV must keep the lane	1/(1-DCL); 1.0E+06
R2	EV must not collide with other vehicles	1/DV; 1.0E+06
R3	EV must not collide with pedestrians	1/DP; 1.0E+06
R4	EV must not collide with static meshes	1/DS; 1.0E+06
R5	EV must complete the route	1/(1-DT); 1.0E+06
R6	EV must abide by traffic rules (i.e., red lights)	0; 1.0E+06

as the agent starts choosing a random action. As training progresses and the agent acquires knowledge of the environment, ϵ gradually decreases, and the agent starts selecting actions greedily with higher probability. The annealing schedule, i.e., the rate at which ϵ decreases over time, is problem-specific, as it depends on the rate of exploration that is required to effectively learn a task.

3 Replication study

3.1 Problem definition

In the work by Haq *et al.* [13], the ADS is embedded within the CARLA simulator [7], which renders a realistic town environment including junctions, vehicles, pedestrians, traffic lights, and traffic signs. The ADS that controls the **ego-vehicle** (**EV**) has to drive it through a predefined route. At each time step, the ADS receives and processes data from sensors (e.g., camera and LIDAR) to generate driving commands (steering, throttle, and braking) to maximize the driving performance. The CARLA leaderboard [17] measures the ADS performance with the *driving score*, a combination of two metrics, namely route completion and infraction penalty. The former isthe percentage of route completion; the latter measures the number of infractions, including traffic rules violations (e.g., red lights and stop signs) and detected collisions with other vehicles, pedestrians, and static elements (e.g., road signs).

Haq *et al.* use three test routes, one in which the ADS has to drive through a straight road (Straight), one simulating a left turn (Left-Turn), and one for a right turn (Right-Turn). Each route has three actors, the EV, a **vehicle in front** (**VIF**), and a pedestrian. They define six functional and safety requirements for the ADS (listed in Table 1): the EV must *(R1)* keep the lane; it must not collide *(R2)* with the VIF, *(R3)* with the pedestrian, and *(R4)* with static meshes (e.g., traffic lights/signs); it must *(R5)* complete the route within the given time, and *(R6)* abide by traffic rules.

The simulator reports violations of the requirements, respectively: *(V1)* if the **distance from the center of the lane** (**DCL**) exceeds a threshold identifying the lane boundaries; *(V2)* if the **distance from the VIF** (**DV**), or *(V3)* the **distance from the pedestrian (DP)**, or *(V4)* **distance from static meshes (DS)** is less than or equal to zero (i.e., a collision occurs); *(V5)* if at the end of the scenario the **distance from the destination (DT)** is

greater than zero, and *(V6)* if it detects that the EV has violated a **traffic rule (TR)** (e.g., running a red light[1]). MORLOT [13] changes the environment in each route to find violations of the requirements.

The authors formulate testing as an RL problem. The state space has 19 variables specifying the position, speed, and acceleration of the EV and the VIF, the position and speed of the pedestrian, and environmental conditions (i.e., fog and rain intensity, and sun altitude). The action space of the RL agent is discrete: the agent can choose in a set of 17 actions controlling the VIF (throttle and steering), the pedestrian (speed and position), and the environmental conditions. Each action changes the value of a controlled variable by a small and constant amount - e.g., the action for the VIF throttle increase/decrease by 0.1 the current throttle value. Finally, a reward function is defined for each requirement. For the R6/TR requirement the agent gets a large reward when the requirement is violated, zero otherwise. For all other requirements, the reward is a function of the `distance` to the violation. Specifically, it is equal to $\frac{1}{\texttt{distance}}$ when $\texttt{distance} > 0$, and to 1.0E+06 otherwise (a requirement is violated). The testing goal is to minimize the distances DV, DP, and DS, and maximize the distances DCL and DT (in such cases the authors subtract 1 from the original distance value, i.e., $1 - \texttt{distance}$).[2]

MORLOT is designed as a many-objective search with Q-learning. It builds a Q-table for each requirement to be violated, such that every Q-table is updated with the transitions (i.e., $\langle$*state, action, reward, next state*$\rangle$ tuples) related to the respective reward function. In this way, each Q-table learns the optimal state-action pairs to violate the respective requirement. MORLOT stores a list of uncovered objectives (initially all of them are uncovered), and at each step selects the action from the Q-table associated with the uncovered objective closest to being covered (i.e., the objective with the highest reward at the previous step). For example, let us assume that at a certain step there are two uncovered objectives, i.e., R2 and R3, and that the distance between the EV and the VIF is 10 meters (i.e., DV = $10m$), while the pedestrian is at 20 meters from the EV (i.e., DP = $20m$). In this case, MORLOT chooses an action from the Q-table associated with R2, since $\frac{1}{DV} > \frac{1}{DP}$.

3.2 Subject and configuration

The ADS under test is TransFuser [28], a Deep Neural Network model submitted to the CARLA Autonomous Driving challenge. TransFuser is a Multi-Modal Fusion Transformer to predict the trajectory of the ego vehicle to determine the driving commands. Haq *et al.* used the first version of TransFuser, proposed in 2021 [28]. The code of the TransFuser agent, and the pre-trained

[1] Running a red light is the only violation considered for TR in the original study.

[2] DCL and DT range between 0 and 1.

models, are open-source.[3] The MORLOT replication package[4] links to those models and contains the same scenarios Straight, Left-Turn, Right-Turn.

We configured the environment to run TransFuser following the instructions in the replication package of MORLOT, as well as the instructions by the authors of TransFuser. We validated our local configuration of the ADS by running the agent in the three scenarios and checking that no violation occurred (i.e., the TransFuser agent drives well in nominal conditions).

3.3 Replication

3.3.1 Methodology

The replication package of Haq *et al.* provides the code to execute the considered test generators, after configuring the ADS and the simulator with the given scenario. In both the original paper and our replication study, MORLOT is compared against the random baseline that randomly selects the action to perform at each step. These two techniques, included in the replication package, require proper configuration of the execution environment. In particular, by inspecting the code in the package, we found an important, initially undocumented, configuration option (later added by the authors in the package README file), a boolean flag named "RL", that determines the way violations are detected. As the name suggests, it needs to be set to true when running RL-based test generators (like MORLOT), while it needs to be set to false when running the random baseline, to allow the use of simulator sensors for detecting collisions and lane invasions. Specifically, setting the flag to true allows MORLOT to extract requirement violations from a file storing all violations detected by sensors at each simulation step; the file is deleted after being processed. On the contrary, non-RL algorithms are designed to process this file only at the end of each episode, thus only detecting violations at the last step. As a consequence, when running the random baseline with the flag set to true, the detection is solely based on distances between objects given by the simulator. We execute our experiments with both versions of random. We call RANDOM_False the baseline with the flag set to false, and RANDOM_True, the baseline with the flag set to true (i.e., the default value in the original replication package).

We compare MORLOT to the baselines in terms of effectiveness (coverage of requirements) and efficiency (coverage achieved over time), following the authors' evaluation [13]. The coverage of a technique is 100% when it finds at least one violation per requirement within the time budget of 4 hours, as in the original study. MORLOT adopts an ϵ-greedy strategy with ϵ linearly decreasing from 1 to 0.1 in the first 48 minutes (20% of the budget). To account for randomness, we executed each technique 20 times (twice the repetitions used by Haq *et al.*) in each of the three scenarios, for a total of 720 hours of

[3] `https://github.com/autonomousvision/transfuser/tree/cvpr2021`.

[4] `https://doi.org/10.6084/m9.figshare.20526867`.

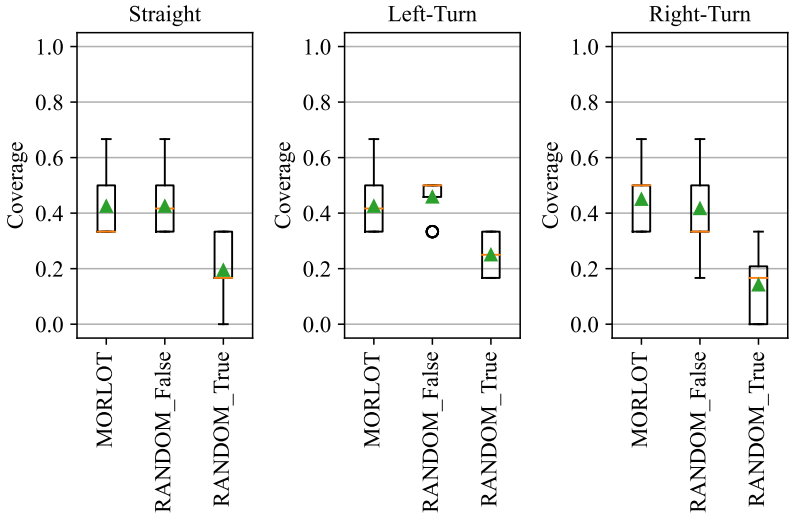

Fig. 1 Coverage of safety and functional requirements of the three techniques.

computation. All experiments were executed on the Google Cloud Compute Engine platform on a virtual machine with Ubuntu 18.04, an Intel Haswell CPU (4 cores) and an NVIDIA Tesla T4 with 16 GB of VRAM.

3.3.2 Testing techniques effectiveness

Figure 1 shows the coverage achieved by the three testing techniques over the 20 repetitions. The orange line and the green arrow represent the median and the mean coverage, respectively. Results show that RANDOM_True achieves the lowest coverage (0.33) as it never covers more than 2 requirements. On the other hand, MORLOT and RANDOM_False exhibit a similar average coverage. MORLOT's median is slightly lower than RANDOM_False's in Straight and Left-Turn routes but higher in the Right-Turn route. Both techniques cover at least 2 requirements, with a maximum of 4 (coverage: 0.66).

We run the Friedman test [10], a non-parametric hypothesis test for Analysis of Variance, to assess if there is at least one testing technique that significantly differs from the others. The Friedman test detects a significant difference for at least one pair for each route (the p-values are, respectively, 3.93E-06, 1.734E-06, 1.458E-08, for Straight, Left-Turn, and Right-Turn).

We run the Dunn test [8] for *post hoc* analysis to detect pairs of techniques differing significantly. Table 2 reports the p-values per pair. The test confirms there is no significant difference between MORLOT and RANDOM_False, while both show a statistically higher coverage than RANDOM_True.

Table 3 compares the testing techniques as for the average number of violations per requirement they trigger. It shows that MORLOT and RANDOM_False expose a comparable number of violations (respectively: 6.15 and

6.20 for Straight, 16.35 and 15.35 for Left-Turn, 5.45 and 5.30 for Right-Turn), with both being better than RANDOM_True in all the three routes.

Both the coverage of safety requirements and the average number of violations show that activating sensors to detect collisions and lane invasions (i.e., setting the "RL" boolean configuration flag to true) increases the measured effectiveness of the RANDOM_False baseline. Indeed, we noticed, by inspecting the execution logs, that most of the times a collision is detected by sensors, the distance value is close to the threshold but does not exceed it (the threshold value is 0 in the case of the DV, DP, DS requirements and equals 1.15 in the case of DCL). Upon further inspection of the code, we found that the distance between two objects is computed considering the geometric centers of their collision boxes, while sensors detect collisions when the bounding boxes of the respective objects intersect. Depending on the angle of collision, the distance between the two objects may be greater than zero, even when the bounding boxes intersect. Ultimately, this issue affects the RANDOM_True baseline, significantly decreasing its measured effectiveness, but does not represent an issue for the reward computation for MORLOT, since the implementation forces the distance value to the threshold value whenever a sensor detects a violation.

3.3.3 Testing techniques efficiency

We compare the testing techniques efficiency as coverage of the requirements over time. Figure 2 shows the trend of average coverage per technique in the 20 repetitions. Each average coverage value is sampled every 20 minutes; the shaded area represents the standard error of the mean (i.e., $\pm\, s/\sqrt{n}$). RANDOM_True always achieves the lowest coverage on all routes. In the Straight and Left-Turn routes, it achieves only half the coverage of MORLOT and RANDOM_False, and only at the end of the 4-hour budget; in the Right-Turn route it achieves only a third of the coverage. On the other hand, RANDOM_False and MORLOT achieve comparable coverage over time. MORLOT curve mostly stays slightly below the coverage curve of the RANDOM_False baseline, with the exception of the Right-Turn route.

We compute the Area Under the Curve (AUC) for the curves of the coverage of the techniques over time (Figure 2). Table 4 reports the (min-max)[5] normalized mean value and standard deviation for all routes. Again, RANDOM_True achieves significantly lower values, while MORLOT and RANDOM_False obtain comparable results, with the first being slightly worse

[5] With min and max respectively equal to 0 and the overall maximum value.

Table 2 Pairwise comparisons Dunn Test - Coverage of the requirements (statistically significant differences are in bold).

Techniques	Straight	Left-Turn	Right-Turn
△MORLOT vs. RANDOM_False▽	1.00E+00	8.61E-01	1.00E+00
△MORLOT vs. RANDOM_True▽	**△<1.00E-04**	**△<1.00E-04**	**△<1.00E-04**
△RANDOM_False vs. RANDOM_True▽	**△<1.00E-04**	**△<1.00E-04**	**△<1.00E-04**

Table 3 Average number of violations found by MORLOT, RANDOM_False (RAND_F), and RANDOM_True (RAND_T).

	Straight			Left-Turn			Right-Turn		
Requirement	MORLOT	RAND_F	RAND_T	MORLOT	RAND_F	RAND_T	MORLOT	RAND_F	RAND_T
R1, DCL	0.15	0.20	0.15	5.85	4.90	4.90	0.00	0.00	0.00
R2, DV	1.40	1.50	0.00	9.60	9.20	0.00	0.60	0.45	0.00
R3, DP	3.80	3.85	0.80	0.85	1.25	0.60	3.05	3.10	0.45
R4, DS	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00
R5, DT	0.75	0.65	0.75	0.00	0.00	0.00	0.55	0.55	0.65
R6, TR	0.05	0.00	0.00	0.05	0.00	0.00	1.25	1.20	0.00
TOTAL	6.15	6.20	1.70	16.35	15.35	5.50	5.45	5.30	1.10

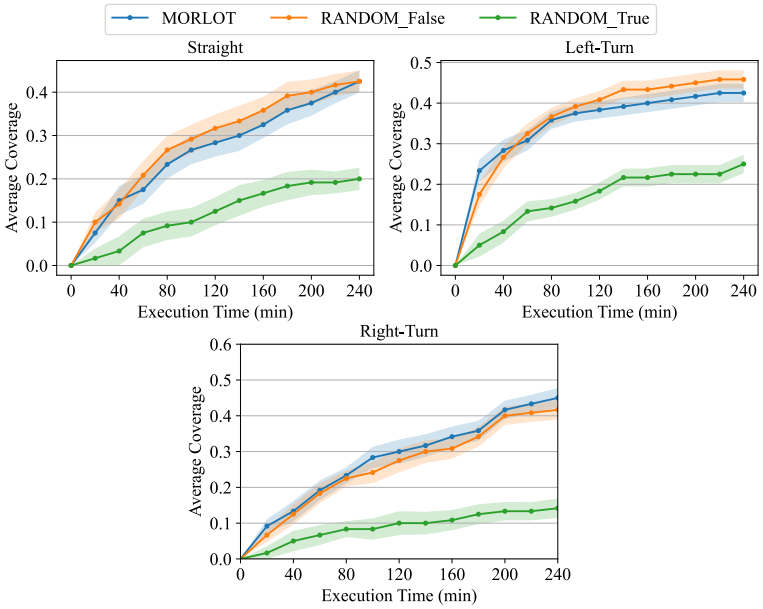

Fig. 2 Average coverage of safety and functional requirements over time.

Table 4 Area under the curve: Normalized Mean and Std.

Technique	Straight Mean (±Std)	Left-Turn Mean (±Std)	Right-Turn Mean (±Std)
MORLOT	0.52 (±0.22)	0.64 (±0.12)	0.60 (±0.18)
RANDOM_False	0.57 (±0.20)	0.67 (±0.12)	0.56 (±0.21)
RANDOM_True	0.23 (±0.18)	0.30 (±0.09)	0.19 (±0.20)

among all routes except for Right-Turn. The Friedman test applied to the AUC values detects a significant difference for at least one pair for each route (the p-values are respectively 1.09E-05, 4.11E-10, 4.11E-10 for Straight, Left-Turn, and Right-Turn). The Dunn Test (Table 5) confirms that MORLOT and RANDOM_False are equivalent, while both outperform RANDOM_True.

Table 5 Area under the curve: Pairwise comparisons Dunn Test (statistically significant differences are in bold).

Techniques	Straight	Left-Turn	Right-Turn
△MORLOT vs. RANDOM_False▽	1.00E+00	1.00E+00	1.00E+00
△MORLOT vs. RANDOM_True▽	△**1.00E-03**	△**<1.00E-04**	△**<1.00E-04**
△RANDOM_False vs. RANDOM_True▽	△**<1.00E-04**	△**<1.00E-04**	△**1.00E-04**

3.4 Discussion

Comparing our findings with those of Haq *et al.*, we observe that MORLOT and RANDOM_True match the results of the original paper. However, setting the sensor activation flag to false in the random baseline (which we call RANDOM_False in the evaluation), we found no significant difference with MORLOT, in both effectiveness and efficiency. We investigated the causes for the poor performance of MORLOT w.r.t. RANDOM_False and we identified two reasons, related to the design and implementation of the RL algorithm.

The first reason concerns Q-learning, the RL algorithm chosen by Haq *et al.* Q-learning requires both state and action spaces to be discrete. In MORLOT, actions are discretized, but the state is implemented as a string concatenating the continuous values of the 19 state variables. MORLOT's implementation of Q-learning is dynamic, i.e., it initializes an empty table and adds a row each time it encounters a new state. During training, if it revisits a previously discovered state, it updates the corresponding Q-value. To explore the evolution of the Q-table dimension, we monitored the count of distinct states memorized in the Q-table (i.e., its dimension) relative to the total number of steps executed by the agent in a repetition. To prevent potential biases due to the overhead of collecting this information, we ran 3 additional repetitions for MORLOT in the three routes. The average Q-table dimensions are depicted in Figure 3, revealing a nearly linear progression over time. This suggests that the agent rarely encounters the same state more than once, decreasing the learning effectiveness of the algorithm and its ability to detect violations.

The second reason concerns MORLOT's definition of the reward functions and the way the algorithm utilizes the Q-table. For requirement R6 on traffic rules (TR), the reward function is sparse (reward is different from 0 only in case of violations, which are very rare). For requirement R5 on distance from destination DT, the reward function starts at its maximum at the beginning of the route (maximum DT) and decreases as the EV progresses along the route. The function rewards the RL agent, independently[6] of the chosen action, which makes it difficult for the RL algorithm to properly assign credit to the actions that make the ADS violate R5. As for requirements R1–R4 (on DCL, DV, DP, DS, respectively), some of them are trivial to violate, while others are very challenging (e.g., R4/DS is rarely violated). We noticed that MORLOT covers first the easiest requirements to violate (namely: R2/DV and

[6] An exception is when the VIF stops and the EV waits for it to move out of its trajectory. In this case, the reward remains constant.

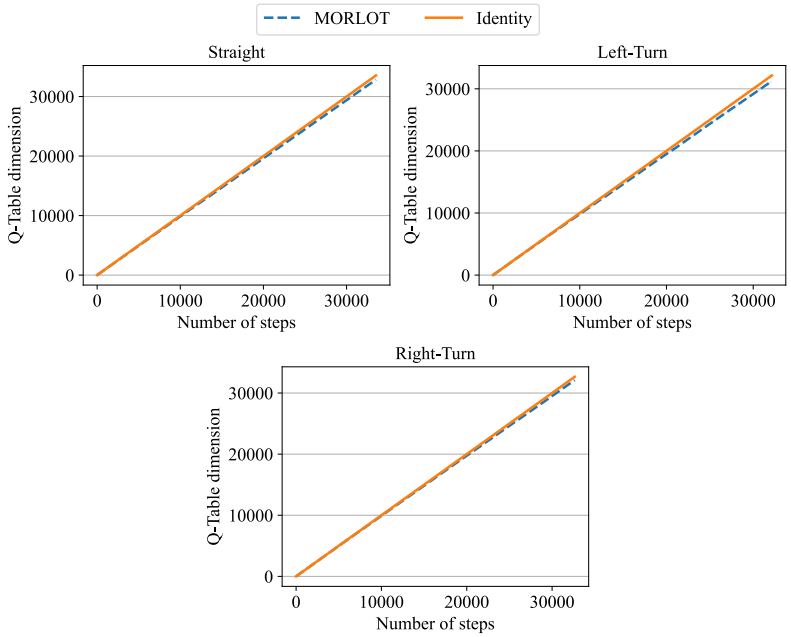

Fig. 3 Average Q-Table dimension (number of observed states) over number of steps.

R3/DP in Straight; R1/DCL and R2/DV in Left-Turn)[7] and then starts to select actions from the Q-table associated with the R4/DS requirements. This happens because DS has one of the highest average reward values, and MORLOT selects the Q-table with the highest reward at each step. Since R4/DS is difficult to violate, this results in MORLOT wasting the remaining search budget without addressing any further requirement. Table 6 shows the mean and standard deviations of the reward values obtained by MORLOT for each requirement.[8] Notably, DS ranks third in terms of reward value. In conclusion, MORLOT's formulation translates the *multi-objective* problem into a *multi-agent* one. However, addressing a multi-objective problem by using multiple agents, each associated with a single objective, requires specific methodologies to ensure balanced training, selection of different agents, and to promote "altruism" among the agents (e.g., as designed in distributed reinforcement learning frameworks [14]).

Table 6 Statistics of MORLOT rewards per requirement.

	R1/DCL	R2/DV	R3/DP	R4/DS	R5/DT
Mean (Median)	1.7 (1.1)	12.1 (11.1)	9.0 (5.9)	6.8 (5.3)	0.8 (0.9)
Std (IQR)	37.4 (0.2)	7.4 (11.1)	9.5 (6.0)	4.3 (5.64)	0.2 (0.33)

[7] We also observed that most of the times R2/DV is violated, R1/DCL is violated, too.
[8] TR is not reported as it is binary.

4 Extension study

4.1 Motivation

The results of the replication study motivate to investigate the extent to which RL can benefit online testing of ADS. To avoid potential inefficiencies arising from MORLOT's *multi-agent* formulation, the extension considers the reward function for only one requirement, in a single-objective problem formulation. We choose DQN as it can naturally handle continuous state spaces, and is expected to scale to the size of the case study state space. The goal is thus to assess if an RL agent trained with DQN performs better than RANDOM under the settings of Haq *et al.*, in a single-objective problem formulation.

We start focusing on the DV requirement (Section 4.2 and Section 4.4), as it proved to be a non-trivial yet coverable requirement in all routes of the replication study. Existing studies show that this requirement covers 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, 37]. Then, in Section 4.5 we investigate to what extent the findings generalize to two other requirements, DCL and DP.

We compare Q-learning, as implemented in MORLOT's replication package, and DQN with the RANDOM_False baseline (called simply RANDOM hereinafter) on the same tasks defined by Haq *et al.* In the replication, we execute 20 repetitions with a budget of 4 hours for all techniques on the three routes, for 720 hours of computation. Additionally, the experiments on the DCL and DP requirements demanded for a further 960 hours of computation, totaling 1,680 hours. We share all results in the online repository.[9]

4.2 Problem definition

The replication study showed that: (1) inclusion of multiple requirements, some of which are almost impossible to violate, but still return high rewards, is one of the reasons for the degenerate behavior of RL, which performed comparably to random; (2) adoption of Q-learning reduces the agent's learning capability, as similar continuous states are not recognized as recurring states. Hence, in our extension study, the main focus is on a single requirement (e.g., *respect safety distance from other vehicles*), and the comparison of two RL methods (DQN[10] and Q-learning) with RANDOM as baseline. We also update the termination condition of an episode, to take into account the achievement of the driving objective. An episode stops both when the simulator detects a collision between the ADS and the VIF (a violation is detected), and when the ADS overtakes the VIF (a violation can no longer occur).

[9] `https://doi.org/10.6084/m9.figshare.24794544`.

[10] The DQN model consists of a multi-layer perceptron with three hidden layers, two of size $2 \times$ `input_dimension`, and one of size `input_dimension` (where `input_dimension` is the size of the state space, i.e., 19). The hyperparameters are: `learning_rate` $= 1e-2$, `gamma` $= 0.9$, `batch_size` $= 32$, `buffer_size` $= 2,000$, `target_update_interval` $= 100$.

4.3 Experiments

4.3.1 Evaluation criteria

We compare DQN, Q-learning (hereafter simply Q), and RANDOM as for *effectiveness* and *efficiency*. Effectiveness is computed as total number of violations of the R2/DV requirement (collisions with VIF) in a 4 hours budget. Efficiency is computed as average number of violations over time. In the subsequent *qualitative evaluation* we analyze the failures exposed by the techniques.

4.3.2 Effectiveness

Figure 4 shows the box plots of the number of violations of the DV requirement found by all the techniques across the 20 repetitions in the three routes. We observe that Q triggers as many violations as the RANDOM baseline across all routes. This supports the conclusion of the replication study: even with a single-objective formulation, Q is indistinguishable from RANDOM. On the contrary, a deep RL agent like DQN demonstrates performance superior to the other two techniques in most of the routes (Straight and Right-Turn), where it triggers, on average, twice the number of violations compared to Q and RANDOM. DQN is the least effective in the Left-Turn route.

To statistically compare the three techniques, we run the Friedman test, which detects a significant difference for at least one pair of approaches for all the routes (p-value = 1.94E-03, 7.76E-03, 1.67E-03). Table 7 reports the p-values for all the pairwise comparisons, computed with the Dunn test. DQN significantly outperforms both Q and RANDOM in the Straight and Right-Turn routes, while performing worse than both in the Left-Turn route. On the other hand, Q and RANDOM are statistically indistinguishable.

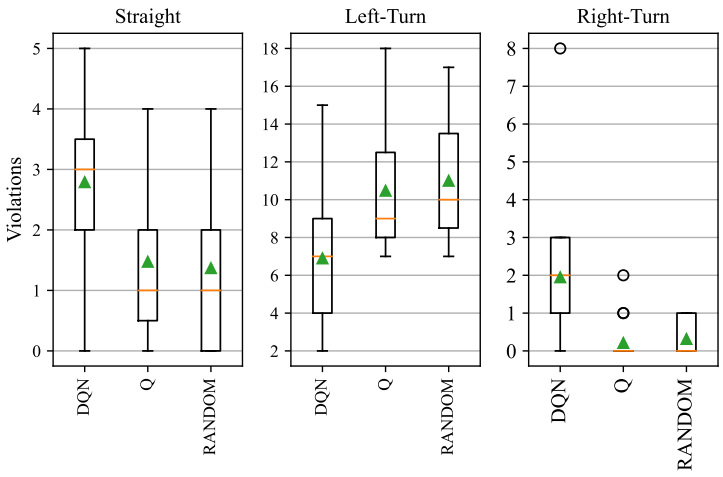

Fig. 4 Box plots of the number of violations of the DV requirement.

Table 7 Pairwise comparisons Dunn Test - Number of violations of the DV requirement (statistically significant differences are in bold).

Techniques	Straight	Left-Turn	Right-Turn
△DQN vs. Q▽	**△3.43E-02**	**▽9.20E-03**	**△<1.00E-04**
△DQN vs. RANDOM▽	**△4.00E-03**	**▽1.80E-03**	**△2.90E-03**
△Q vs. RANDOM▽	1.00E+00	1.00E+00	1.00E+00

4.3.3 Efficiency

Figure 5 shows the average number of violations identified by the three techniques over time, with the shaded area representing the standard error of the mean (i.e., $\pm\, s/\sqrt{n}$). We observe that DQN is the most efficient technique in 2 of the 3 routes. For the Straight route, DQN triggers the first violation, on average, one hour earlier than Q and RANDOM, and it requires less than half the budget to obtain the same number of violations. In the Right-Turn route, DQN is the only technique able to discover up to two violations in each repetition. In the Left-Turn route, DQN is less efficient than Q and RANDOM.

Q and DQN have the same trend as RANDOM in the initial 50 minutes, the time budget allocated for ϵ-greedy exploration. However, unlike Q, which maintains the same behavior as RANDOM through the entire period, DQN is able to learn an effective policy to trigger violations of the ADS under test.

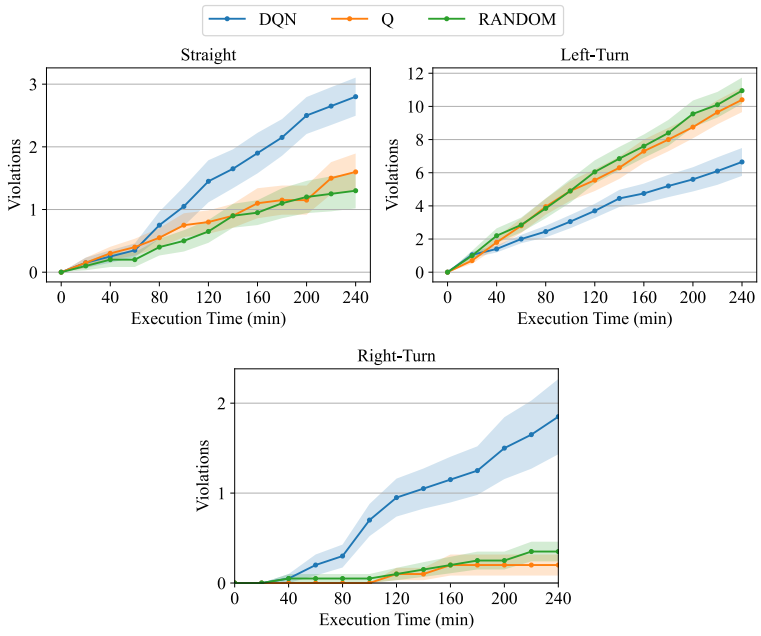

Fig. 5 Average number of violations of the DV requirement over time.

Table 8 Area under the average number of violations of the DV requirement curve: Normalized Mean and Std.

Technique	Straight	Left-Turn	Right-Turn
DQN	0.38 (±0.26)	0.32 (±0.15)	0.34 (±0.32)
Q	0.22 (±0.20)	0.48 (±0.20)	0.03 (±0.09)
RANDOM	0.19 (±0.19)	0.51 (±0.22)	0.06 (±0.10)

To quantitatively evaluate the efficiency, we measured the AUC of the average number of violations over time (Figure 5). Table 8 shows the (min-max) normalized AUC mean and standard deviation per technique, in all routes. DQN covers 54% of the area in Straight and 48% in Right-Turn, while the coverage is lower than Q and RANDOM in Left-Turn (i.e., 36% vs $\approx$ 56% of Q and RANDOM). The Friedman test applied to the AUC values detects a significant difference for at least one pair for Left-Turn and Right-Turn routes, but not for the Straight route (i.e., the p-values are 1.10E-01, 1.82E-02, and 5.88E-03 for Straight, Left-Turn, and Right-Turn, respectively).

We then ran the Dunn test for all pairs of approaches and for the routes in which Friedman exposed a significant difference (i.e., Left-Turn and Right-Turn). Table 9 shows that there is no significant difference between Q and RANDOM in all routes. DQN shows a higher AUC than both in the Right-Turn route. Despite no statistically significant difference in the Straight route, the trend is in favor of DQN (on average, the AUC values of DQN compared to Q and RANDOM are respectively 53% vs 31% and 26%). In the Left-Turn route, DQN has an AUC lower than both Q and RANDOM.

We investigate the performance difference of DQN in the three scenarios, by first analyzing the trend of the DV metric quantifying the distance from the VIF over time. As the reward function for Q and DQN is the inverse of DV (until a requirement violation occurs, when a reward of 1.0E+06 is returned), the RL agent aims at minimizing the distance between the EV and the VIF. Figure 6 shows the trend of this distance (in meters); each point represents an average of the distance values collected in a window of 10 minutes of execution; we then average all the values across the 20 repetitions.

In two scenarios - Left- and Right-Turn - DQN makes the distance decrease over time (i.e., it maximizes the average reward); however, observing the average distance over time (Figure 6) and the average number of violations over time (Figure 5), we notice that in one of the two scenarios (Left-Turn) a decreasing distance does not correspond to a higher number of violations. Moreover, in the Straight route, DQN outperforms both Q and RANDOM for

Table 9 Area under the the average number of violations of the DV requirement curve - Pairwise comparison Dunn Test (statistically significant differences are in bold).

Compared techniques	Straight	Left-Turn	Right-Turn
△DQN vs. Q▽	-	**△2.30E-02**	**△1.00E-04**
△DQN vs. RANDOM▽	-	**△1.13E-02**	**△3.50E-03**
△Q vs. RANDOM▽	-	1.00E+00	1.00E+00

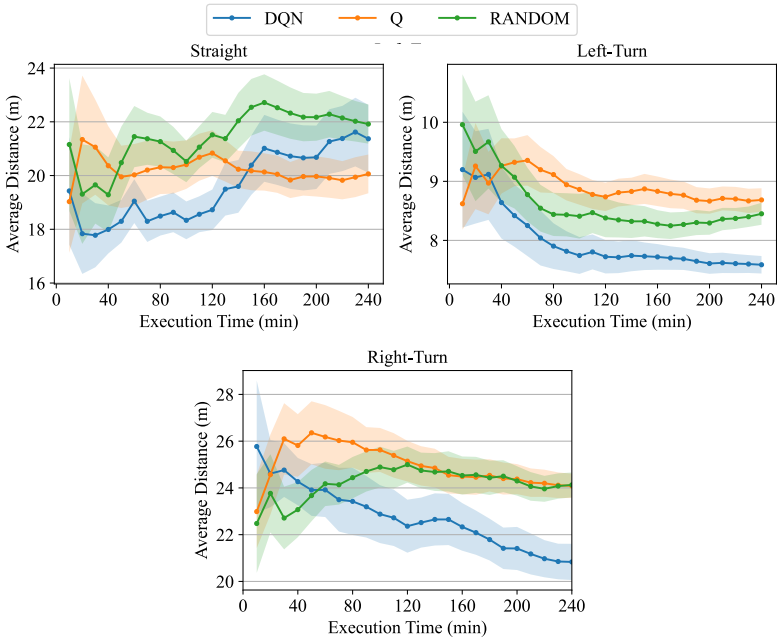

Fig. 6 Average distance from the Vehicle in Front (DV) over time.

number of violations, while the three seem to be equivalent w.r.t. the distance trend (left-most plots in Figure 5 and Figure 6). This might be due to the effectiveness of the large reward returned upon collision. Indeed, by looking at Figure 7, which plots the average reward computed within a time window of 10 minutes, we observe that the average reward for DQN has an increasing trend in the Straight and Right-Turn routes, and it is consistently higher than the average reward obtained by Q and RANDOM. In the Left-Turn route, DQN finds less violations than Q and RANDOM despite being able to minimize the distance between the two vehicles over time (center plots in Figure 5, and Figure 6). However, the reward it obtains (Figure 7) shows constantly lower values compared to Q and RANDOM. The only scenario where the number of violations triggered by DQN is higher, the reward increases, and the distance decreases is the Right-Turn. Also in this case the distance for Q and RANDOM does not decrease over time, as well as their respective reward curves.

These observations point to issues associated with the reward function (defined by Haq *et al.* and adopted unchanged in our extension study). Specifically, this reward function has two components: 1) a continuous value determined by the inverse of the distance between the EV and the VIF, and 2) a very large constant value (1.0E+06) when a collision occurs. The first component is dense, i.e., it is given to the agent at each time step, while the second one, although larger than the first one (1.0E+06 against the first one that ranges between 0 and 100), is sparse as it is only given when a collision occurs. In this setting, the agent needs to find a trade-off between the two

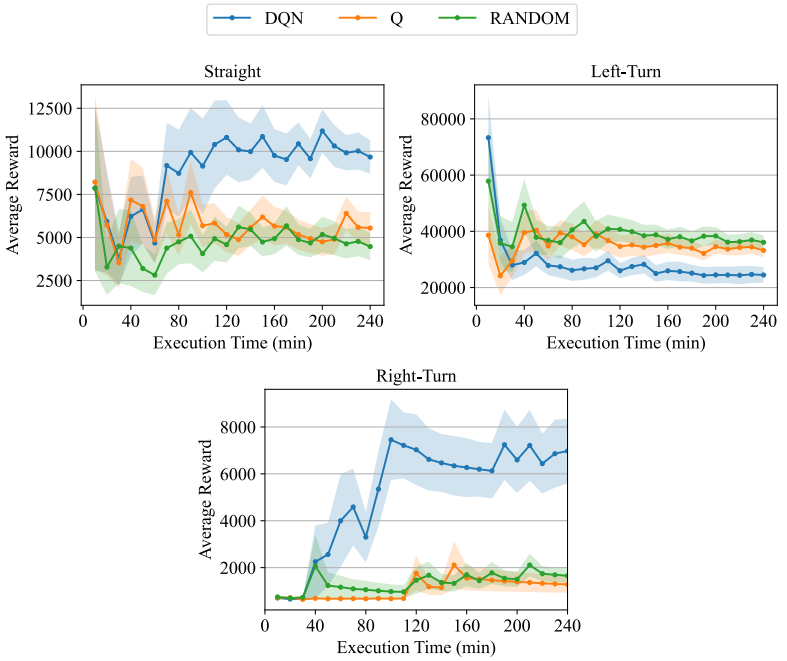

Fig. 7 Average reward over time (DV requirement).

components. For instance, in both Left-Turn and Right-Turn the agent tends to privilege the immediate reward by minimizing the distance between the EV and the VIF (see the decreasing distance trend in Figure 6). In particular, the agent, which controls the VIF, learns to steer to move backward, in order to get closer to the EV. However, while in the Left-Turn route, this behavior does not lead to collisions, in the Right-Turn route, where the two components of the reward function positively correlate with each other, the DQN agent learns to minimize the distance, which eventually leads to a collision and ultimately to a higher average reward (Figure 7). In summary, the inverse distance used as a dense reward component is not always guiding the agent toward violations of the R2/DV safety requirement.

Figure 8 shows the distribution of the actions selected by each technique in the three routes, across 20 repetitions. We can observe that the actions selected by Q and RANDOM follow a uniform distribution. This suggests that the Q agent does not converge, as it selects actions randomly. On the other hand, DQN tends to privilege certain sets of actions, despite the distributions include the exploration phase, when DQN samples actions uniformly at random. In particular, in the Straight and Right-Turn routes, where DQN finds more violations, it selects actions related to the VIF (i.e., actions 0—3) and weather conditions (i.e., actions 4—9). In the Left-Turn route, where DQN performs worse and converges to a suboptimal policy, besides actions related to the VIF, it also tends to select actions related to the pedestrian (i.e., actions 10—15).

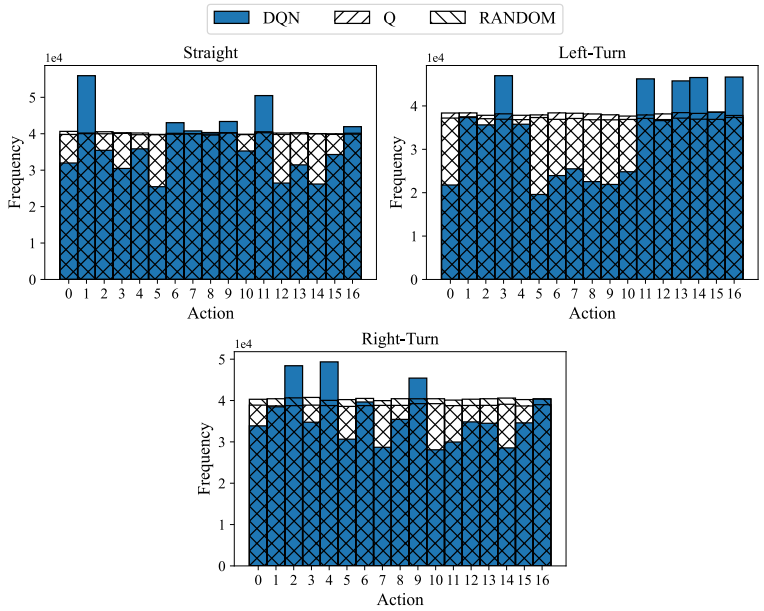

Fig. 8 Action distributions (DV requirement).

4.3.4 Qualitative evaluation

To analyze the different effectiveness of DQN in the three routes, we qualitatively evaluate the violations triggered by DQN and those triggered by RANDOM. We do not show the violations of the Q agent, as results show that Q is statistically indistinguishable from RANDOM both in terms of effectiveness and efficiency.[11] During the execution of the testing agents, we log the x and y coordinates of the VIF, and we keep only the trajectories that result in a violation. For each route, Figure 9 shows: on the left, a bird's eye view of a failing scenario, with the starting point of the EV (black circle), the most relevant obstacles in the route (zebra-striped rectangles), and the lane the EV is expected to follow (delimited by two solid red lines); on the right, the failure trajectories of the VIF for each technique, as well as the obstacles.

The failure trajectories show, for each route, that whenever the VIF stops and partially occupies the EV lane, the ADS controlling the EV is unable to avoid it, resulting in a collision (the three bird's-eye views in Figure 9 show such collisions in each route). This occurs in two main cases. The first one is when the sequence of VIF actions leads it to brake and stop in the critical zone. This case accounts for all the trajectories in Figure 9 that do not terminate close to an obstacle. The second case occurs when the VIF collides with an obstacle such that the tail of the vehicle remains partially in the EV lane.

[11] We report the violations of the Q agent in the replication package.

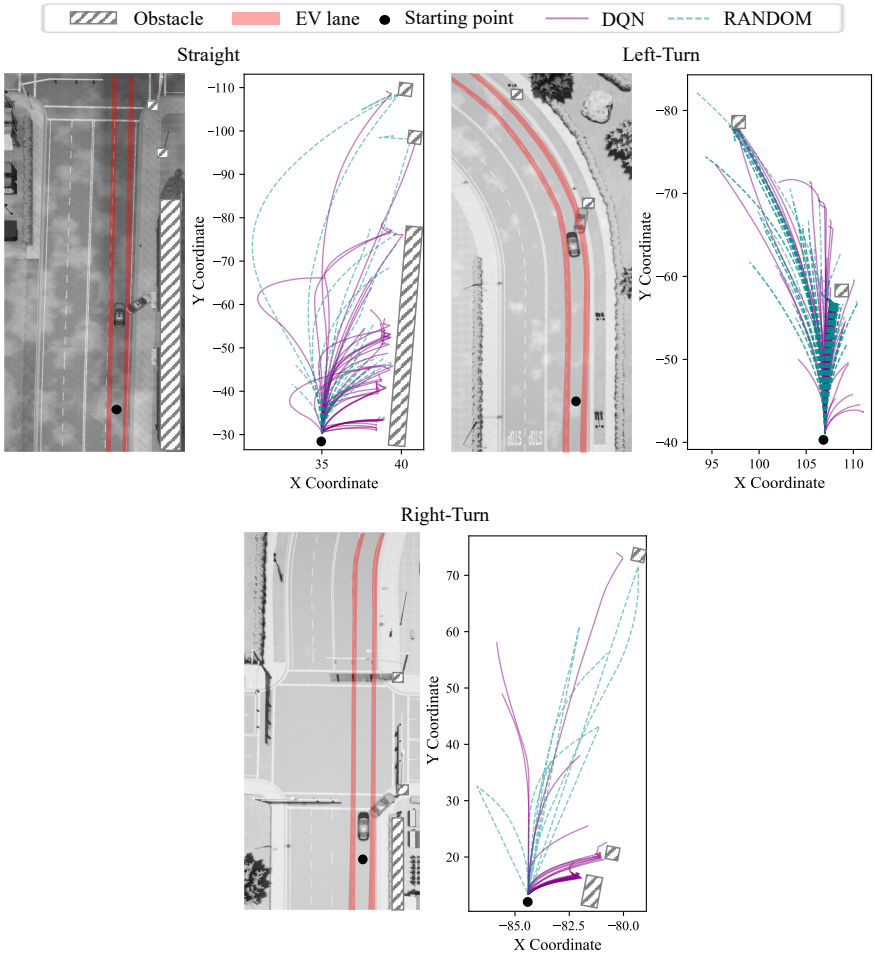

Fig. 9 Failure trajectories of the Vehicle in Front in the three scenarios.

Figure 9 shows that, independently of the route, the most common type of failure trajectory is the second one. In the Straight and Right-Turn routes one observes that DQN learns to collide the VIF with the obstacles on the right to effectively find a high number of violations. The behavior of DQN, in terms of failure trajectories, is very similar in these two routes. The Right-Turn route has a long straight road before the turn, and all the failure trajectories terminate before the turn. In these two routes, the two components of the reward function positively correlate. Indeed, the agent, by steering in one direction to minimize the distance, makes the VIF collide with obstacles and, at the same time, this maneuver causes the EV to collide with the VIF. This way, the agent gets positively rewarded at each time step by minimizing the distance (first component), receiving a large (constant) reward for the collision with the EV and the violation of the DV requirement (second component).

In the Left-Turn route (center plots), the turn is very close to the starting point, playing an active role in most of the failure trajectories. We observe that most violations found by RANDOM occur when the VIF collides with the first obstacle, located at the beginning of the curve, yet in a straight line from the starting point. Its position contributes to making the RANDOM approach effective in this route, together with the way the action space is defined.

We identify two reasons why the formulation of the action space is critical in this route. First, by selecting random actions, there is a small chance of choosing actions that affect the VIF trajectory (only 4 out of 17 actions increase/decrease the throttle or steer left/right; the others manipulate environment variables, such as the luminosity). Since the actions affecting the VIF consist of small variations of the throttle and steering commands, an action needs to be repeatedly selected to meaningfully affect the trajectory of the VIF in the short distance that separates the starting point from the obstacle. Second, even assuming that RANDOM repeatedly selects VIF-related actions, there is the possibility of selecting counter-actions, i.e., increase the throttle when the previous action decreased it, or steer right when the previous action turned the steer to the left, resulting in the VIF going straight. Indeed, Figure 9 shows that the trajectories of the VIF with RANDOM are mostly straight, which is one of the ADS failing conditions in the Left-Turn route.

In the Left-Turn route, the DQN agent follows the immediate reward of the first component of the reward function, which privileges steering actions to minimize the distance between the EV and the VIF. This makes collisions between the two vehicles very infrequent. Correspondingly, the agent is unable to trade off immediate and long-term rewards, within the given time budget. In summary, in the Left-Turn route, minimizing the distance between the EV and the VIF does not frequently lead to collisions, since most of the collisions are caused by the VIF stopping in the middle of the two lanes due to a crash with a static obstacle. Moreover, random search is particularly effective in this route, as it tends to generate straight trajectories that frequently lead the VIF to collide with the static obstacle, causing a collision with the EV.

4.4 Discussion

The extension study highlights both the advantages and the limitations of using RL for testing ADS in the settings of Haq *et al.* [13]. We show that DQN is able to learn an effective policy to trigger significantly more ADS violations than both Q and RANDOM in 2 out of 3 routes. DQN is also more efficient than compared approaches in finding violations, which is critical in the context of ADS testing where the tests execution is highly time-consuming. In addition, the routes in which DQN outperformed other techniques are the most meaningful from a testing point of view, as they are those where it is difficult to find violations by random exploration. Indeed, throughout the entire budget, RANDOM finds, on average, only one violation in the Straight route, less than one in Right-Turn route, and eleven violations in the Left-Turn route.

Overall, our extension study supports the claim that a DRL agent such as DQN can converge to an effective policy, significantly improving effectiveness and efficiency over random search. This result enables the design of novel RL-based techniques to test ADS in complex driving simulators like CARLA. However, this requires researchers to carefully formulate the RL problem, including the definition of states and actions spaces, and the reward function. Indeed, our extension study pinpoints the limitations of the replicated study. We identified three issues in the RL formulation by Haq *et al.*, that need to be addressed to fully exploit the RL framework as a testing tool for ADS.

First, the two (dense and sparse) components of the reward function are not always connected with each other, making them sometimes inadequate to effectively guide the agent in complex scenarios such as the Left-Turn route. A possible alternative is the one proposed by Lu *et al.* [23], who use the probability of collision as a reward function. The collision probability accounts for lateral and longitudinal distances, as well as for additional metrics such as the speed and positions of all the actors in the testing scenario.

The second issue concerns the definition of the state space of the RL agent. Indeed, in the formulation by Haq *et al.* [13] the state includes absolute coordinates, which unnecessarily increases the dimensionality of the state space. For instance, two configurations of the environment that are similar in terms of relative positions of the EV and VIF vehicles but occur in different locations of the route, are encoded as different states. This reduces the possibility for the agent to reuse previously acquired knowledge, potentially increasing the training time. One way to overcome this issue is to encode relative variables in the state space such as ego-centric polar coordinates, or lane-centric coordinates [20]. This can increase the generalization capabilities of the RL agent in those situations that require the same behavior.

The third issue is related to the definition of the action space. The actions are discrete in the setting by Haq *et al.*, and they are designed to slightly perturb the dynamics of actors in the environment, such as slightly increasing/decreasing the throttle of the VIF, or making it slightly steer left/right. The effects of such small changes are delayed, as multiple small perturbations are needed to create a meaningful change. This slows down the learning process for the agent, as it needs to assign credit to the actions that lead to a high reward, despite their delayed effect. A possible solution to speed up learning, is to use a mixture of two common strategies. The first strategy is using an Observation Time Period (OTP) [23] (also called *frame skipping* in the RL literature [2, 26]), which consists of applying an action and waiting for it to produce an effect on the environment, by *pausing* the RL agent for a number of simulation steps (in the current setting, the RL agent acts at each simulator step). This way, the reward computed for a certain action comprises multiple simulation steps, giving more precise feedback to the agent. The second strategy consists of creating a layer of abstraction between the decision-making policy of the agent, and the low level actions needed to control dynamic actors in the environment (i.e., low level controls of throttle and steering angle for the VIF) [19]. In this setting, the actions available to the agent would be

meta-actions, such as "change lane", "overtake", and "stay idle". The low level controller, running at a higher frame rate than the RL policy, takes care of translating the meta-actions to the actual commands, leaving the RL agent the responsibility to make the most important and relevant decisions.

In summary, results on the DV requirement show that complex scenarios demand a thorough reformulation of the problem to allow Reinforcement Learning agents to be effective. On the positive side, they also show that the DQN agent may be more effective and efficient in the most meaningful and interesting scenarios (i.e., those where it is difficult to expose failures). Based on these findings, we further investigate, in the next section, to what extent they apply to other safety requirements under the setting of Haq *et al.*

4.5 Other requirements

4.5.1 Formulation

We now study DQN performance on two other requirements: distance from the center of the lane (R1/DCL, DCL hereinafter) and distance from pedestrians (R3/DP, DP hereinafter). The problem formulation is the same as for the DV requirement (Section 4.2), apart from the obviously different termination condition: an episode ends when the ego vehicle exceeds its lane boundaries (for DCL) or collides with a pedestrian (for DP). Clearly, distances from the center of the lane and from the pedestrian are now used for reward computation.

We compare DQN effectiveness and efficiency against the RANDOM baseline, in finding violations of the two requirements We execute 20 repetitions with a 4-hours budget for all the techniques on the three routes, totaling 960 hours of computation.

4.5.2 Distance from center of lane

Figure 10 shows the box plots of the number of violations of the DV requirement found by DQN and RANDOM across the 20 repetitions in the three routes. We observe that in two routes (Straight and Right-Turn), violations are very difficult to expose: DQN and RANDOM have a median number of violations equal to 0, while DQN exhibits a slightly higher mean. Specifically, in the Straight route, DQN found one violation in 3 out of 20 repetitions, compared to one violation found by RANDOM in just one repetition. In the Right-Turn route, DQN is the only technique that finds a violation (also including the techniques considered in the replication study), in 2 out of 20 repetitions. In the Left-Turn route, where violations are easier to expose, DQN and RANDOM find on average the same number of violations (i.e., approximately 4). To statistically compare the results, we ran the Wilcoxon test [41], at $\alpha = 0.05$ significance level, which did not detect a significant difference between the two techniques in any of the routes (i.e., p-values = 3.17E-01, 6.86E-01, 1.57E-01,

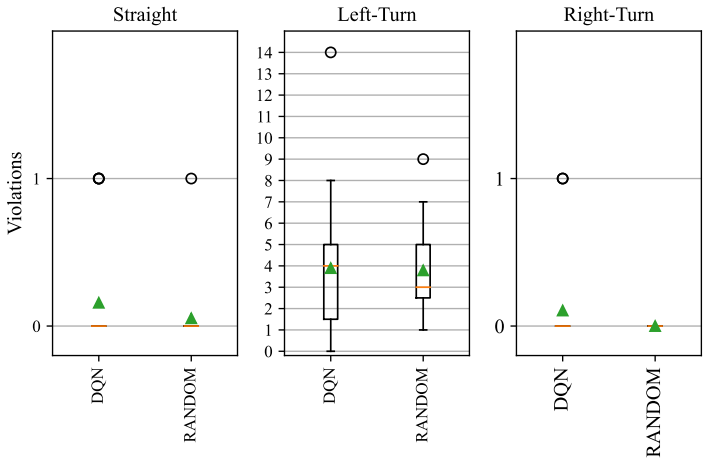

Fig. 10 Box plots of the number of violations of the DCL requirement.

respectively for Straight, Left-Turn and Right-Turn). Despite the lack of statistical significance, the trend seems to be in favor of DQN, which triggers on average more violations in the most difficult routes.

As for efficiency, Figure 11 shows the average number of violations of DQN and RANDOM over time. While in the Left-Turn route the two techniques appear equivalent, DQN outperforms RANDOM in the other two routes. On average, the former requires half the budget to find more violations than the latter. The Wilcoxon test at $\alpha = 0.05$ on the AUC of the two techniques, fails to reject the null hypothesis (i.e., p-values = 4.65E-01, 9.56E-01, 1.79E-01, respectively for Straight, Left-Turn and Right-Turn routes). Again, despite the lack of statistical significance, DQN shows a positive trend in finding more violations quicker than RANDOM, especially in the Right-Turn route.

For further performance comparison, Figure 12 plots the average reward of the two techniques, computed within an execution window of 10 minutes. It confirms that in the Straight and Right-Turn routes DQN obtains a higher reward, while the two techniques seem to be equivalent in the Left-Turn route.

Finally, Figure 13 plots the distribution of actions selected by the two techniques. It shows that, in contrast to RANDOM's uniform distribution, DQN privileges a certain set of actions and converges to a non-uniform policy.

4.5.3 Distance from pedestrians

Figure 14 shows the box plots of the number of violations of the DP requirement found by DQN and RANDOM across the 20 repetitions in the three routes. DQN outperforms RANDOM only in the Left-Turn route, while it finds fewer violations in other routes. These results, in line with the experiments on other requirements, highlight the superiority of RL agents in finding violations in scenarios where violations are difficult to expose randomly (i.e., the Left-Turn route in this case). For statistical comparison, we ran the Wilcoxon test

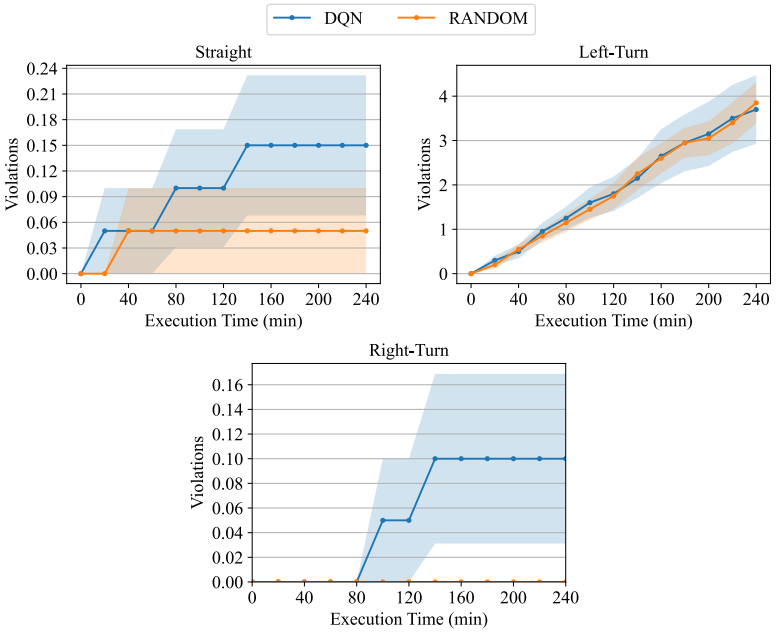

Fig. 11 Average number of violations of the DCL requirement over time.

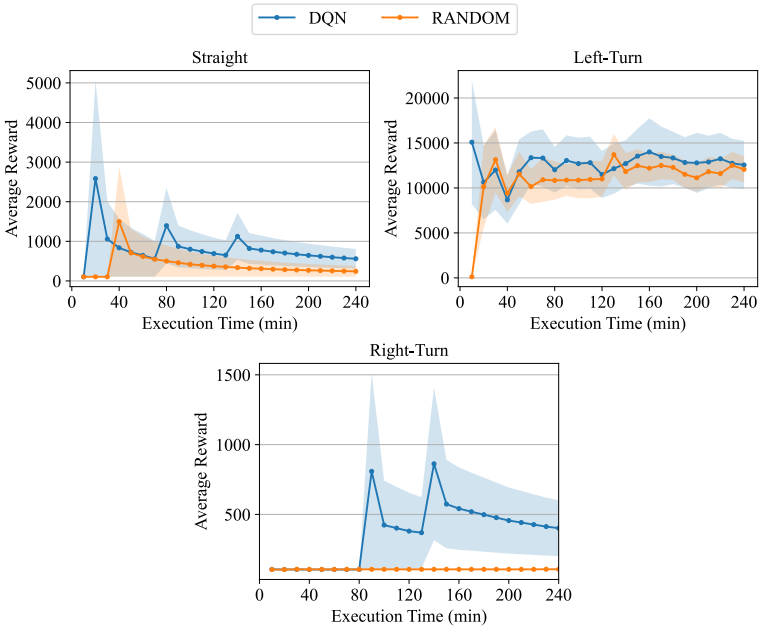

Fig. 12 Average reward over time (DCL requirement).

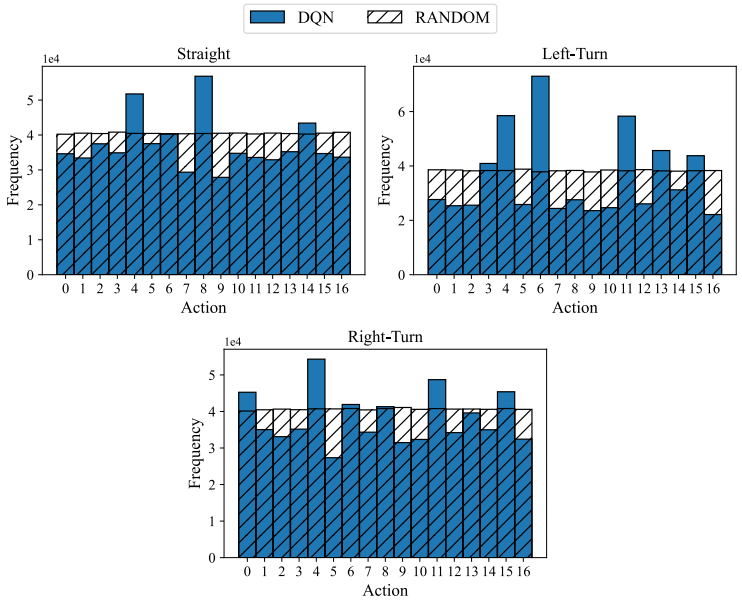

Fig. 13 Actions distributions (DCL requirement).

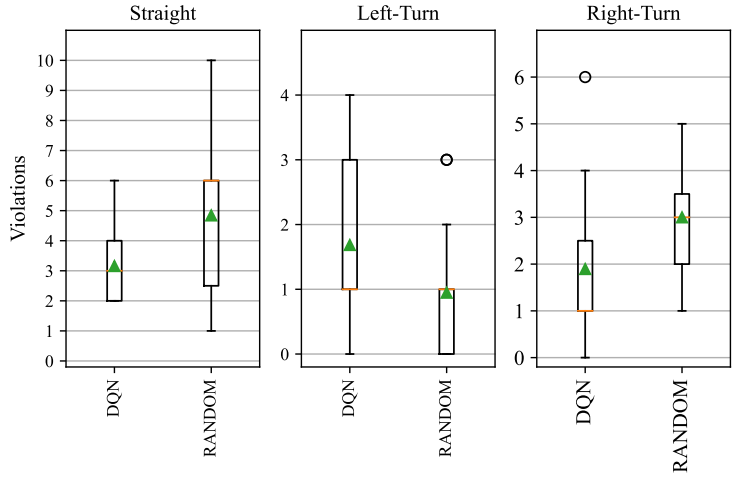

Fig. 14 Box plots of the number of violations of the DP requirement.

at $\alpha = 0.05$ significance level, which detected a significant difference for the Straight and Right-Turn routes (p-values = 4.59E-03, 5.05E-02 [12], 2.15E-02, respectively for Straight, Left-Turn and Right-Turn routes).

[12] Power analysis requires a sample of ≈140 repetitions to reach the conventional statistical power threshold $\beta = 0.8$, which would cost weeks of computation time.

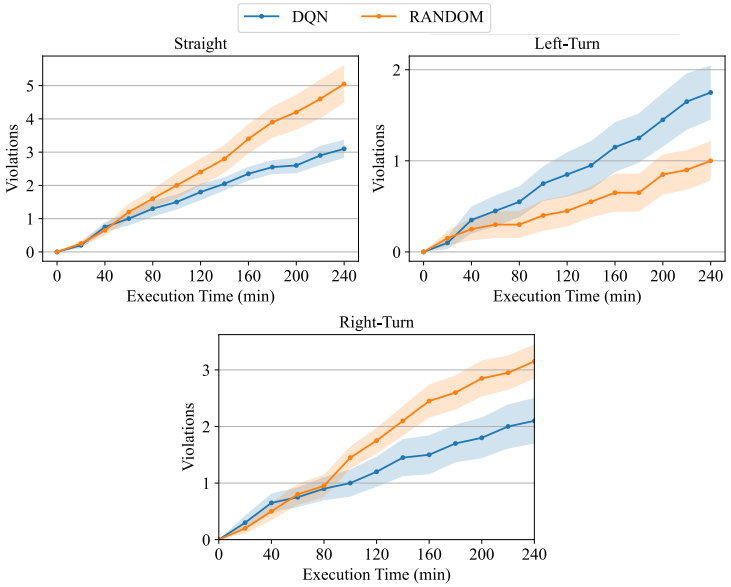

Fig. 15 Average number of violations of the DP requirement over time.

As for efficiency, Figure 15 plots the average number of violations over time found by DQN and RANDOM. It confirms that RANDOM finds violations quicker than DQN in the Straight and Right-Turn routes, while the contrary happens in Left-Turn. Indeed, in Left-Turn DQN requires about one third less time (80 minutes) than RANDOM to find the first violation. The Wilcoxon test at significance level $\alpha = 0.05$ for the AUC of the two techniques confirms the difference in the Straight and Right-Turn routes (p-values = 4.20E-02, 1.14E-01, 4.40E-02, respectively for Straight, Left-Turn and Right-Turn).

Finally, Figure 16 and Figure 17 plot, respectively, the average reward of DQN and of RANDOM in a 10 minutes window, and the distribution of the selected actions across repetitions. Figure 16 shows that in the Straight and Right-Turn routes DQN obtains lower reward values. Figure 17 shows that DQN converges to a subset of actions with respect to RANDOM, though in the Straight and Right-Turn routes this does not lead to more violations.

4.5.4 Discussion

The experiments on additional requirements confirmed both the positive and negative findings obtained on the DV requirement. As autonomous systems become more reliable, RL shows potential as a technique for online testing, particularly in scenarios where violations are challenging to identify.

On the other hand, RL is not always effective, especially in scenarios where violations are easily exposed by random search. To address these limitations, a reformulation of the testing problem in the RL framework is in order.

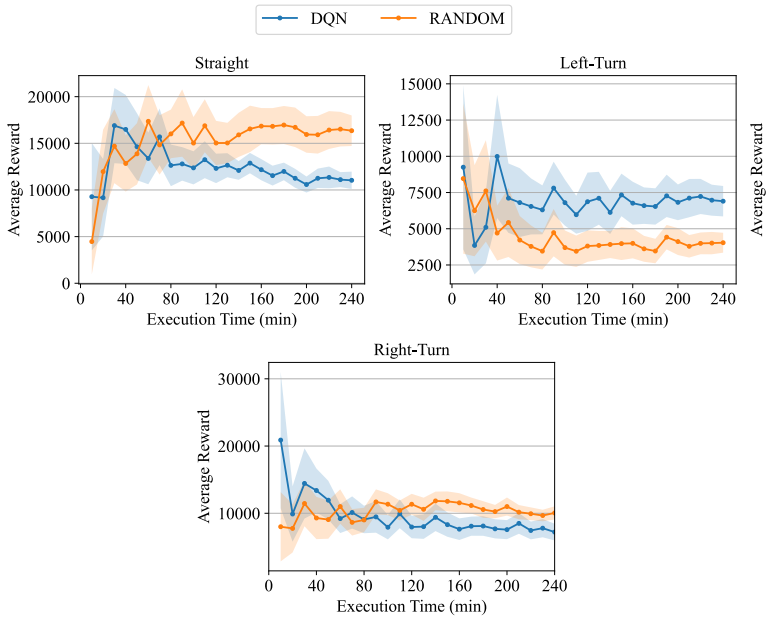

Fig. 16 Average reward over time (DP requirement).

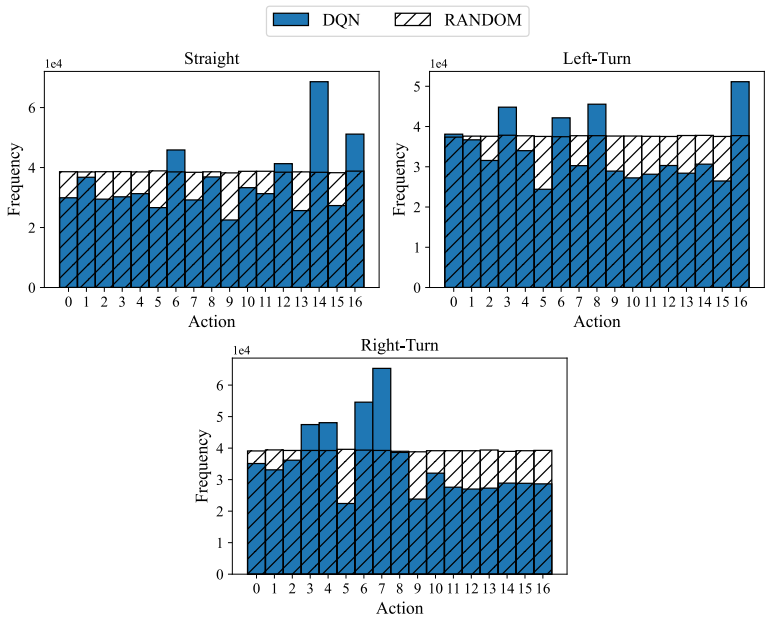

Fig. 17 Actions distributions (DP requirement).

5 Related work

Online testing of ADSs has been widely investigated, with many approaches proposed to generate scenarios that cause ADS misbehaviors [3,4,11,12,15,21, 25,29,38]. Among them, search-based techniques showed particularly effective. Before MORLOT, Haq *et al.* [12] introduced SAMOTA, a technique utilizing surrogate models to predict the outcome of a test case without executing it. A test case is a static configuration of the environment, including factors like road type and weather parameters. Riccio and Tonella [29] proposed DeepJanus to generate frontier inputs, i.e., similar input pairs that cause the ADS to misbehave for one input while working correctly for the other. Calò *et al.* [4] used search-based techniques to identify *avoidable* collision scenarios, where a collision would not have occurred with an ADS reconfiguration. They first search for a collision and then for a proper ADS configuration to avoid it.

These proposals do not deal with sequences of dynamic interactions, required to manipulate objects during simulations (e.g., another vehicle). In this context, the use of RL for online testing of ADSs is gaining increasing interest. Besides Haq *et al.* [13], a relevant work is the one by Lu *et al.* [23]. Similarly to MORLOT, they propose a learning technique (DeepCollision) that dynamically changes the environmental conditions to find collisions with vehicles, pedestrians, and static obstacles (corresponding to violations *V2*, *V3*, and *V4* in Section 3.1). DeepCollision uses DQN as RL agent to select actions from a set of 52 options to control weather, time of the day, and behavior of actors (e.g., pedestrian crossing the road and vehicle switching lane). The state is defined by a set of 12 variables including traffic lights color, EV kinematics (position, speed, and rotation), and weather conditions. The authors employ the Apollo ADS [1] and the LGSVL simulator [30]. Unfortunately, LGSVL is unmaintained since 2022, and the cloud servers are no longer operational.

Other techniques use Adaptive Stress Testing (AST), a method initially employed by Lee *et al.* [18] to test an aircraft collision avoidance system. AST formulates the problem of finding the most likely failure scenarios as a Markov decision process, which can be solved by RL agents. Koren *et al.* [16] explore the application of AST to find collisions in pedestrian crossing scenarios by extending it with deep RL. Corso *et al.* [5] also use AST, focusing on the reward formulation to find diverse and avoidable scenarios[13] as failing scenarios.

Sharif and Marijan [31] define a multi-agent environment in which a set of deep RL agents are trained with adversarial inputs. The goal is to find ADS failure states in which the EV goes off road or collides with obstacles. ADS robustness is then improved by retraining it with adversarial inputs.

The advances in applying reinforcement learning for online testing of ADSs are undeniable. However, the proposed techniques involve highly intricate and varied simulators, defining complex RL environments that require thorough

[13] They consider some scenarios to be unavoidable (e.g., a pedestrian causing a collision with a stopped ADS).

study for drawing accurate conclusions. Our work highlights the significance of replication studies in this context, which have not been conducted thus far.

6 Conclusions

Scientific research on testing core software components - driven by artificial intelligence - of Autonomous Driving Systems has shown that Reinforcement Learning can be highly beneficial in this difficult and time-consuming task. However, replication studies still lack in this field, despite their importance to establish well-grounded scientific evidences in empirical software engineering.

We have replicated a recent study which showed the superiority of RL, combined with many objective search, with respect to random testing for ADSs. While not confirming the results in the original study, the replication has provided insights on how to design RL algorithms tailored to the specific domain.

We have thus extended the original study, showing that deep RL, with an agent designed to better fit the characteristics of the state and action spaces of the RL problem for testing ADSs, can outperform random testing in effectiveness and efficiency in covering their functional and safety requirements. Further empirical studies are needed to address for more complex scenarios.

Data availability

For the sake of Open Science, we provide all artifacts for replication and all results, available at: `https://doi.org/10.6084/m9.figshare.24794544`.

Acknowledgements

This work was partially supported by the EU H2020 project PRECRIME, funded under the ERC Advanced Grant 2017 Program (ERC Grant Agreement n. 787703). It has also received funding from the EU H2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 871342 "uDEVOPS". This material is supported by the Google Research Credits program with the award GCP291784478.

References

1. Apollo, B.: `https://github.com/ApolloAuto/apollo` (2017)
2. Bellemare, M., Veness, J., Bowling, M.: Investigating contingency awareness using atari 2600 games. In: Proceedings of the AAAI Conference on Artificial Intelligence, vol. 26 (1), pp. 864–871. AAAI Press, Palo Alto, CA, USA (2012)
3. Ben Abdessalem, R., Nejati, S., C. Briand, L., Stifter, T.: 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). DOI 10.1145/3180155.3180160

4. Calò, A., Arcaini, P., Ali, S., Hauer, F., Ishikawa, F.: Generating avoidable collision scenarios for testing autonomous driving systems. In: IEEE 13th International Conference on Software Testing, Validation and Verification (ICST), pp. 375–386. IEEE (2020)
5. Corso, A., Du, P., Driggs-Campbell, K., Kochenderfer, M.J.: Adaptive stress testing with reward augmentation for autonomous vehicle validation. In: Intelligent Transportation Systems Conference (ITSC), pp. 163–168. IEEE (2019). DOI 10.1109/ITSC.2019.8917242
6. Dell'Anna, D., Aydemir, F.B., Dalpiaz, F.: Evaluating classifiers in se research: the ecser pipeline and two replication studies. Empirical Software Engineering **28**(1), 3 (2022). DOI 10.1007/s10664-022-10243-1
7. Dosovitskiy, A., Ros, G., Codevilla, F., Lopez, A., Koltun, V.: CARLA: An open urban driving simulator. In: 1st Annual Conference on Robot Learning, *Proceedings of Machine Learning Research*, vol. 78, pp. 1–16. JMLR, Cambridge, MA, USA (2017)
8. Dunn, O.J.: Multiple comparisons using rank sums. Technometrics **6**(3), 241–252 (1964). DOI 10.1080/00401706.1964.10490181
9. Favarò, F.M., Nader, N., Eurich, S.O., Tripp, M., Varadaraju, N.: Examining accident reports involving autonomous vehicles in california. PLoS one **12**(9), e0184952 (2017)
10. Friedman, M.: The use of ranks to avoid the assumption of normality implicit in the analysis of variance. Journal of the American Statistical Association **32**(200), 675–701 (1937). DOI 10.1080/01621459.1937.10503522
11. Gambi, A., Müller, M., Fraser, G.: 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), pp. 318–328. ACM (2019). DOI 10.1145/3293882.3330566
12. Haq, F.U., Shin, D., Briand, L.: Efficient Online Testing for DNN-Enabled Systems using Surrogate-Assisted and Many-Objective Optimization. In: 44th International Conference on Software Engineering (ICSE), pp. 811–822. ACM (2022). DOI 10.1145/3510003.3510188
13. Haq, F.U., Shin, D., Briand, L.: Many-Objective Reinforcement Learning for Online Testing of DNN-Enabled Systems. In: IEEE/ACM 45th International Conference on Software Engineering (ICSE), pp. 1814–1826. IEEE (2023). DOI 10.1109/ICSE48619.2023.00155
14. Hayes, C.F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L.M., Dazeley, R., Heintz, F., Howley, E., Irissappane, A.A., Mannion, P., Nowé, A., Ramos, G., Restelli, M., Vamplew, P., Roijers, D.M.: A practical guide to multi-objective reinforcement learning and planning. Autonomous Agents and Multi-Agent Systems **36**(1), 26 (2022). DOI 10.1007/s10458-022-09552-y
15. Klischat, M., Althoff, M.: Generating critical test scenarios for automated vehicles with evolutionary algorithms. In: IEEE Intelligent Vehicles Symposium (IV), pp. 2352–2358. IEEE (2019). DOI 10.1109/IVS.2019.8814230
16. Koren, M., Alsaif, S., Lee, R., Kochenderfer, M.J.: Adaptive stress testing for autonomous vehicles. In: Intelligent Vehicles Symposium, pp. 1–7. IEEE (2018). DOI 10.1109/IVS.2018.8500400
17. Leaderboard, C.A.D.: CARLA leaderboard. `https://leaderboard.carla.org/leaderboard/` (2020). Accessed: November 5, 2024
18. Lee, R., Kochenderfer, M.J., Mengshoel, O.J., Brat, G.P., Owen, M.P.: Adaptive stress testing of airborne collision avoidance systems. In: 34th Digital Avionics Systems Conference, pp. 6C2–1–6C2–13. IEEE/AIAA (2015). DOI 10.1109/DASC.2015.7311450
19. Leurent, E.: An environment for autonomous driving decision-making. `https://github.com/eleurent/highway-env` (2018)
20. Leurent, E.: A survey of state-action representations for autonomous driving (2018). URL `https://hal.science/hal-01908175`
21. Li, G., Li, Y., Jha, S., Tsai, T., Sullivan, M., Hari, S.K.S., Kalbarczyk, Z., Iyer, R.: Av-fuzzer: Finding safety violations in autonomous driving systems. In: IEEE 31st International Symposium on Software Reliability Engineering (ISSRE), pp. 25–36. IEEE (2020). DOI 10.1109/ISSRE5003.2020.00012
22. Lindsay, R.M., Ehrenberg, A.S.C.: The design of replicated studies. The American Statistician **47**, 217–228 (1993). DOI 10.1080/00031305.1993.10475983

23. Lu, C., Shi, Y., Zhang, H., Zhang, M., Wang, T., Yue, T., Ali, S.: Learning configurations of operating environment of autonomous vehicles to maximize their collisions. IEEE Transactions on Software Engineering **49**(1), 384–402 (2023). DOI 10.1109/TSE.2022.3150788
24. Maes-Bermejo, M., Gallego, M., Gortázar, F., Robles, G., Gonzalez-Barahona, J.M.: Revisiting the building of past snapshots — a replication and reproduction study. Empirical Software Engineering **27**(3), 65 (2022). DOI 10.1007/s10664-022-10117-6
25. Majumdar, R., Mathur, A.S., Pirron, M., Stegner, L., Zufferey, D.: Paracosm: A language and tool for testing autonomous driving systems. CoRR **abs/1902.01084** (2019). URL `http://arxiv.org/abs/1902.01084`
26. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A.A., Veness, J., Bellemare, M.G., Graves, A., Riedmiller, M., Fidjeland, A.K., Ostrovski, G., et al.: Human-level control through deep reinforcement learning. Nature **518**(7540), 529–533 (2015)
27. Novielli, N., Calefato, F., Lanubile, F., Serebrenik, A.: Assessment of off-the-shelf SE-specific sentiment analysis tools: An extended replication study. Empirical Software Engineering **26**(4), 77 (2021). DOI 10.1007/s10664-021-09960-w
28. Prakash, A., Chitta, K., Geiger, A.: Multi-modal fusion transformer for end-to-end autonomous driving. In: IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 7073–7083. IEEE (2021). DOI 10.1109/CVPR46437.2021.00700
29. Riccio, V., Tonella, P.: 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 (ESEC/FSE), pp. 876–888. ACM (2020). DOI 10.1145/3368089.3409730
30. Rong, G., Shin, B.H., Tabatabaee, H., Lu, Q., Lemke, S., Možeiko, M., Boise, E., Uhm, G., Gerow, M., Mehta, S., Agafonov, E., Kim, T.H., Sterner, E., Ushiroda, K., Reyes, M., Zelenkovsky, D., Kim, S.: Lgsvl simulator: A high fidelity simulator for autonomous driving. In: 23rd International Conference on Intelligent Transportation Systems (ITSC), pp. 1–6. IEEE (2020). DOI 10.1109/ITSC45102.2020.9294422
31. Sharif, A., Marijan, D.: Adversarial deep reinforcement learning for improving the robustness of multi-agent autonomous driving policies. In: 29th Asia-Pacific Software Engineering Conference (APSEC), pp. 61–70. IEEE (2022). DOI 10.1109/APSEC57359.2022.00018
32. Shull, F.J., Carver, J.C., Vegas, S., Juristo, N.: The role of replications in empirical software engineering. Empirical Software Engineering **13**(2), 211–218 (2008). DOI 10.1007/s10664-008-9060-1
33. da Silva, F.Q.B., Suassuna, M., França, A.C.C., Grubb, A.M., Gouveia, T.B., Monteiro, C.V.F., dos Santos, I.E.: Replication of empirical studies in software engineering research: a systematic mapping study. Empirical Software Engineering **19**(3), 501–557 (2014). DOI 10.1007/s10664-012-9227-7
34. Stocco, A., Pulfer, B., Tonella, P.: Model vs system level testing of autonomous driving systems: a replication and extension study. Empirical Software Engineering **28**(3), 73 (2023). DOI 10.1007/s10664-023-10306-x
35. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. A Bradford Book, Cambridge, MA, USA (2018)
36. Tang, S., Zhang, Z., Zhang, Y., Zhou, J., Guo, Y., Liu, S., Guo, S., Xue, Y., et al.: A survey on automated driving system testing: Landscapes and trends. ACM Transactions on Software Engineering and Methodology **32**(5), 1–62 (2023)
37. US Department of Transportation, N.H.T.S.A.: Summary report: Standing general order on crash reporting for automated driving systems. https://www.nhtsa.gov/sites/nhtsa.gov/files/2022-06/ADS-SGO-Report-June-2022.pdf (2022)
38. Tuncali, C.E., Fainekos, G., Ito, H., Kapinski, J.: Simulation-based adversarial test generation for autonomous vehicles with machine learning components. CoRR **abs/1804.06760** (2018). URL `http://arxiv.org/abs/1804.06760`
39. Vescan, A., Pintea, A., Linsbauer, L., Egyed, A.: Genetic programming for feature model synthesis: a replication study. Empirical Software Engineering **26**(4), 58 (2021). DOI 10.1007/s10664-021-09947-7
40. Watkins, C.J.C.H.: Learning from delayed rewards. Ph.D. thesis, King's College, Cambridge, UK (1989)

41. Wilcoxon, F.: Individual Comparisons by Ranking Methods, pp. 196–202. Springer New York, New York, NY (1992). DOI 10.1007/978-1-4612-4380-9_16. URL https://doi.org/10.1007/978-1-4612-4380-9_16
42. Zohdinasab, T., Riccio, V., Gambi, A., Tonella, P.: 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 (ISSTA), pp. 79–90. ACM (2021). DOI 10.1145/3460319.3464811