

Boundary State Generation for Testing and Improvement of Autonomous Driving Systems

Matteo Biagiola, and Paolo Tonella, *Member, IEEE Computer Society*

Abstract—Recent advances in Deep Neural Networks (DNNs) and sensor technologies are enabling autonomous driving systems (ADSs) with an ever-increasing level of autonomy. However, assessing their dependability remains a critical concern. State-of-the-art ADS testing approaches modify the controllable attributes of a simulated driving environment until the ADS misbehaves. In such approaches, environment instances in which the ADS is successful are discarded, despite the possibility that they could contain hidden driving conditions in which the ADS may misbehave.

In this paper, we present GENBO (GENerator of BOundary state pairs), a novel test generator for ADS testing. GENBO mutates the driving conditions of the ego vehicle (position, velocity and orientation), collected in a failure-free environment instance, and efficiently generates challenging driving conditions at the behavior boundary (i.e., where the model starts to misbehave) in the same environment instance. We use such boundary conditions to augment the initial training dataset and retrain the DNN model under test. Our evaluation results show that the retrained model has, on average, up to $3\times$ higher success rate on a separate set of evaluation tracks with respect to the original DNN model.

Index Terms—Software Testing, Autonomous Driving

1 INTRODUCTION

While the dream of fully-autonomous vehicles (the so-called *Level 5*, as defined by the Society of Automotive Engineers (SAE) [1]) is still to come, there exist deployed systems that exhibit impressive levels of automation in the driving task (e.g., the Tesla’s autopilot is considered an advanced Level 2 system [2]). Hence, testing of autonomous driving systems (ADSs) has become a critical research topic, of vital importance for today’s and for future ADSs.

The literature on testing ADS systems is quite rich. A recent survey on ADS testing [3] analyzed 181 papers published in peer-reviewed venues between June 2015 and June 2022. Most of the papers target vision-based ADSs that use Deep Neural Networks (DNNs) to process the input images and decide the actions to take. Such systems are typically trained in a supervised fashion using a dataset of labeled images, where the labels determine the desired actions (e.g., steering angle and throttle). DNN vision-based ADSs are tested offline and/or online [4, 5]. Online testing usually involves a simulator (hence, it is often called simulation-based testing), where the DNN model drives the vehicle with the objective of keeping it within the driving lane [6, 7, 8, 9].

Existing simulation-based testing approaches modify the driving environment in which the DNN model operates. A test case is a set of values representing all (or a subset of) the controllable attributes of the environment. Such attributes determine the environment in which the ADS

operates. In particular, a test case can be a road network [7], a sequence of waypoints determining the track shape [8, 9] or a mixture of dynamic and static elements of the simulation [10, 11, 12, 13]. Examples of static elements that do not change during the simulation, are the track shape, or the positions of the buildings and trees; dynamic elements are the weather condition and the dynamics, including position and velocity, of the other actors in the environment (e.g., pedestrians and other vehicles).

Current ADS testing approaches generate new environment scenarios until the ADS under test misbehaves, discarding those where the ADS is successful. Nonetheless, even in such *failure-free* scenarios, there might exist hidden driving conditions that are left unexplored, and would potentially represent further opportunities to challenge the ADS. This calls for new ADS testing approaches designed to generate challenging driving conditions within the same environment instances. Such testing approaches would also be amenable in resource-constrained settings, where only a limited number of environment configurations (e.g., driving tracks) are available [14, 15, 16], as opposed to existing test generators that modify the simulation environment. Indeed, changing the environment in such settings is expensive (e.g., building new testing tracks) and, sometimes, impossible (e.g., changing the weather condition).

We propose GENBO (GENerator of BOundary state pairs), a novel test generator for online ADS testing. GENBO works by mutating the driving conditions of the ego vehicle the DNN model is controlling, i.e., position, velocity and orientation (a *state* hereafter), within a fixed environment scenario (including the track shape, the weather condition, etc.). GENBO makes full use of an existing, failure-free environment instance, by uncovering challenging driving conditions typically neglected by state-of-the-art approaches. In particular, GENBO employs a novel search algorithm

This article has been accepted for publication in IEEE Transactions on Software Engineering. This is the author’s version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/TSE.2024.3420816

- Matteo Biagiola and Paolo Tonella are with the Università della Svizzera italiana, Lugano, Switzerland. Street: Via Buffi 13. Postcode: 6900. E-mail: {matteo.biagiola, paolo.tonella}@usi.ch.

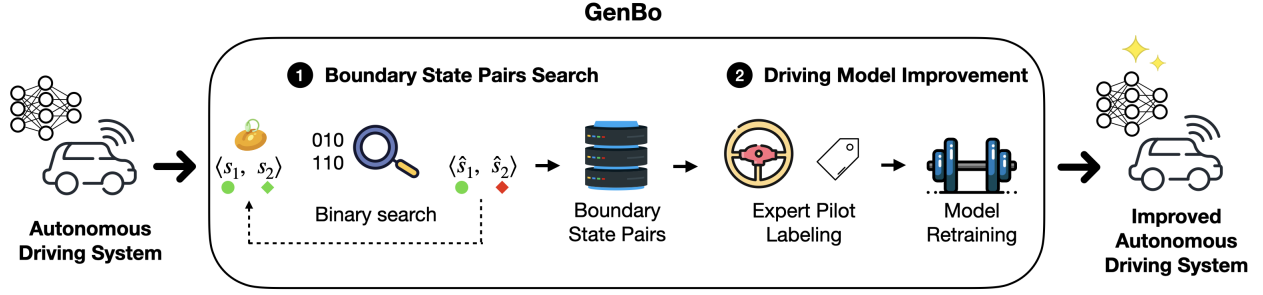


Fig. 1. Overview of our approach. Our approach takes an Autonomous Driving System (ADS) as input and features two steps, namely **1 Boundary State Pairs Search** and **2 Driving Model Improvement**. The first step **1** looks for boundary state pairs of the driving model controlling the ADS, while the second step **2** retrains the driving model using data labeled by an expert pilot driving on boundary state pairs. The output of the approach is an ADS improved by the boundary state pairs.

that evolves pairs of states in order to find *boundary state pairs*, i.e., pairs of states that are *close* to each other and that expose different (successful vs failing) behaviors of the ADS [8, 17, 18]. GENBO uses mutation operators to generate a sequence of state pairs that crosses the behavioral boundary and then applies binary search, which executes a logarithmic number of driving simulations, to efficiently reach the target boundary state pairs.

Among the boundary state pairs, we are interested in the *recoverable* ones, i.e., those where an expert pilot can avoid the failure the ADS exhibits. GENBO uses such pairs to collect a dataset of challenging driving conditions, labeled by an expert pilot. Our hypothesis is that by augmenting the original training dataset with such challenging driving scenarios we can increase the generalization capabilities of the ADS to related, but unseen, driving tracks, which contain similar boundary conditions.

We applied our approach to test the lane-keeping functionality of the Dave-2 model [19], a test subject widely used in the related literature on ADS testing [6, 8, 9, 16, 20, 21, 22, 23, 24]. In our empirical evaluation, we tested driving models from different stages of training, and we show that GENBO can find boundary state pairs even for well-trained models. Our results also show a very strong discriminative capability of boundary state pairs, as boundary state pairs of well-trained models are significantly more challenging than those of poorly trained models. Moreover, we retrained two high-quality driving models after augmenting the original training dataset with labeled examples collected from their boundary state pairs. Results show that the success rate of the retrained models on a set of evaluation tracks is on average up to $3\times$ higher with respect to that of the original model.

Our paper makes the following contributions:

Technique. A novel approach, implemented in the publicly available tool GENBO [25], which exploits a limited number of failure-free environment instances to generate challenging driving conditions for the driving model under test.

Evaluation. An empirical study showing that GENBO exposes challenging driving conditions even for well-trained driving models and that such conditions are useful to significantly improve the model.

2 APPROACH

Figure 1 shows an overview of our approach, which takes as input an ADS and performs two steps: step **1** executes a search algorithm that extracts boundary state pairs from the driving model of the ADS, i.e., its decision-making component. In particular, the search algorithm evolves a pair of nearby states characterizing the driving conditions of the vehicle on a given environment scenario. In the initial pair of states $\langle s_1, s_2 \rangle$ (also called seed state), the driving model succeeds, i.e., when it starts to drive the vehicle either in state s_1 or s_2 , it is able to successfully drive it along the driving track of the given scenario (i.e., none of the two starting states causes a failure). The objective of the search algorithm is to find a pair of *boundary states* $\langle \hat{s}_1, \hat{s}_2 \rangle$, such that in one state (say \hat{s}_1) the driving model succeeds and in the other (\hat{s}_2) it fails. Once the search finds a boundary state pair, it samples a new seed until there is search budget, to find multiple boundary state pairs for the given driving model.

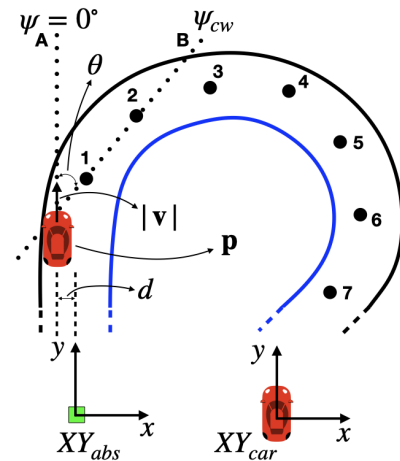


Fig. 2. Visualization of a vehicle state. A vehicle state is defined by three key properties, namely the position of the vehicle inside the driving track p , the orientation of the vehicle ψ , and the magnitude of the velocity vector $v = |\mathbf{v}|$.

Boundary state pairs identify challenging driving conditions for the driving model in a given environment scenario, which can help to improve its capabilities in other scenarios

where similar challenging conditions may occur. In the second step ② of our approach, we resort to an expert pilot to drive on the boundary state pairs of the driving model. The expert pilot labels the new camera images with the ground-truth steering angle values and outputs a labeled dataset. We use such dataset to retrain and improve the driving model of the ADS.

In our experiments, we instantiate the expert pilot with an autopilot that has global knowledge of the given environment scenario. The autopilot automates the labeling process in simulation, as it relies on precise information such as the position of the vehicle at each timestep, and the positions of the track waypoints where the vehicle drives. When the autopilot is not available, e.g., in the real world, the expert pilot is the human driver, labeling the camera images by driving on the road [19, 26].

2.1 Boundary State

The state of a vehicle driving along a track within an environment scenario (i.e., a set of variables representing the state of static and dynamic objects) is a subset of variables characterizing its motion. In particular, a test scenario in our approach is a vehicle state within a fixed environment (in our evaluation a closed driving track with an asphalt road surrounded by green grass, and sunny weather). We define the vehicle state as follows:

Definition 1 (State). A state s of a vehicle in a track T is a tuple $\langle \mathbf{p}, \psi, v \rangle$ where:

- 1) \mathbf{p} is the $\langle x, y \rangle$ absolute position of the vehicle inside T (w.r.t. the frame of reference XY_{abs});
- 2) ψ is the orientation of the vehicle;
- 3) $v = |\mathbf{v}|$ is the magnitude of the velocity vector \mathbf{v} of the vehicle.

Figure 2 shows an example of a vehicle state. The black dots in the figure indicate the waypoints of the track, while the green square indicates the origin of the track. The waypoints are evenly spaced along the track, and they are placed at the center. There are two reference systems, i.e., the absolute reference system XY_{abs} and the reference system of the vehicle XY_{car} . All the quantities that follow are defined w.r.t. XY_{abs} , except for the components of the velocity that are defined w.r.t. XY_{car} .

The vehicle has position \mathbf{p} , where the vector \mathbf{p} is centered at the center of the mass of the vehicle. It has a magnitude velocity of $v = |\mathbf{v}|$ ¹ and orientation ψ . The orientation is defined w.r.t. the y axis of XY_{abs} whose orientation is 0° . Note that acceleration cannot be included as part of the state since it is one of the variables controlled by the driving model. Consequently, at each step, the driving model determines the value of acceleration, disregarding any initial acceleration value possibly provided with the state.

Given the state of a vehicle, we define a function $d(\mathbf{p}, T)$ that measures the distance d of the vehicle from the center of the driving lane of track T (also called cross-track error or XTE [16]). It is also convenient to define a function $\theta(\mathbf{p}, \psi, T)$

that measures the angle θ between the orientation of the vehicle ψ and the orientation of the closest waypoint ψ_{cw} (the latter defining the direction of the road in a specific point in the track). In the figure, the line crossing the vehicle is parallel to the y axis of XY_{abs} , i.e., $\psi = 0^\circ$ (see the dotted line “A”). The next closest waypoint to the vehicle is waypoint “1” and its orientation is determined by the line going through the two successive waypoints (see the dotted line “B”). The angle between the orientation of the vehicle and the orientation of the closest waypoint is θ , the relative orientation of the vehicle.

Definition 2 (State Validity). A state $s = \langle \mathbf{p}, \psi, v \rangle$ of a vehicle in a track T is valid iff:

- 1) $d(\mathbf{p}, T) \leq \frac{W}{2}$, where W is the lane width of track T ;
- 2) $v \leq v_{max}$;
- 3) $\theta(\mathbf{p}, \psi, T) \leq \theta_{max}$;

The first condition predicates that the center of the mass of the vehicle must stay within the bounding box of the track (i.e., the black and blue lines in Figure 2). The second and third validity conditions regard velocity and relative orientation. In both cases, such quantities must be less than a maximum value, to be configured based on the mechanical properties of the autonomous vehicle being simulated. The three validity conditions ensure that a valid state is reachable by the driving model, as they prevent the search algorithm from exploring states that are trivially difficult for the driving model, given the physical properties of the vehicle it controls.

Definition 3 (Boundary State Pair). Given a driving model M and a track T , a boundary state pair is a pair of valid states $\langle s_i, s_j \rangle$ such that:

- 1) the driving model M , when placed in state s_i , *succeeds* at the task of driving the vehicle along track T ;
- 2) on the other hand, the driving model M , when placed in state s_j , *fails* at the task of driving the vehicle along track T ;
- 3) the two states of the pair, i.e., s_i and s_j , are distinct but close to each other, i.e., $s_i \neq s_j$ and $dist(s_i, s_j) \leq \epsilon$.

This definition introduces the operation of *placing* the driving model, and hence the vehicle, in a state s . For instance, if one of the two states in a boundary state pair is $s = \langle (10, 0.5), 0, 25 \rangle$, the simulator would place the vehicle in position $\mathbf{p} = \langle 10, 0.5 \rangle$, with an orientation of $\psi = 0^\circ$ w.r.t. XY_{abs} and an initial velocity $v = 25 \text{ km/h}$.

In each of the two states of a boundary state pair, we check if model M is able to keep the vehicle in lane for a non-negligible amount of time $t \geq t_{min}$. For instance, if the driving simulation is set at 20 *fps* and $t_{min} = 12.5 \text{ s}$, *succeeding* at the lane-keeping task means that the driving model M is able to keep the vehicle in lane for at least 250 simulation steps. In this case, we say that the state s is a *recoverable* state for the driving model M . On the other hand, *failing* at the lane-keeping task, means that the vehicle goes out-of-bound before reaching the time threshold. In this case, we say that the state s is a *non-recoverable* state for the driving model M . A boundary state pair consists of one recoverable and one non-recoverable state. In Figure 1, we indicate recoverable states with a green circle/diamond and non-recoverable states with a red circle/diamond.

1. In practice, in a rigid-body simulator, setting the initial velocity of an object has an immediate effect [27]; afterward, the velocity of the object is determined by the acceleration produced by the driving model controlling it.

Algorithm 1: Pseudocode of GENBO

Input : M , Driving model under test;
 T , Track to drive;
 S , Simulator instance;
 Tr , Reference trace;
 R , Number of restarts;
 N , Number of iterations;
 L , Length of the sequence when mutating a seed state.

Output: A , archive of boundary states.

```

1  $A \leftarrow \emptyset$ 
2 for  $num\_restarts = 1$  to  $R$  do
3   /* Initialize seed state */
4    $s_1 \leftarrow \text{SAMPLEVALIDSTATE}(Tr)$ 
5   repeat
6      $s_2, \text{valid} \leftarrow s_1.\text{MUTATE}()$ 
7   until  $\text{valid}$ 
8    $b \leftarrow \langle s_1, s_2 \rangle$ 
9    $\text{success}_1, \text{success}_2 \leftarrow \text{EXECUTE}(b, S, T, M)$ 
10  if  $\text{success}_1 \otimes \text{success}_2 \wedge b \notin A$  then
11     $A \leftarrow A \cup b$   $\triangleright$  Store collateral boundary state
12  end
13  if  $\neg \text{success}_1 \vee \neg \text{success}_2$  then
14    continue  $\triangleright$  Restart if any execution failed
15  end
16  /* Evolve seed state */
17   $\text{pairs} \leftarrow [b]$ 
18   $num\_iterations \leftarrow 1$ 
19  while  $num\_iterations \leq N$  do
20    for  $seq\_length = 1$  to  $L$  do
21       $\hat{b}, \text{valid} \leftarrow \text{GETLAST}(\text{pairs}).\text{MUTATE}()$ 
22      if  $\neg \text{valid}$  then
23        break
24      end
25       $\text{pairs}.\text{APPEND}(\hat{b})$ 
26    end
27     $idx, it \leftarrow \text{BINARYSEARCH}(\text{pairs}, S, T, M)$ 
28     $num\_iterations \leftarrow num\_iterations + it$ 
29    if  $idx > 0 \wedge \text{pairs}[idx] \notin A$  then
30       $A \leftarrow A \cup \text{pairs}[idx]$ 
31      break
32    end
33  end
34 end
35 return  $A$ 

```

The last condition for two states to form a boundary state pair is *closeness*:

$$\text{dist}(s_i, s_j) \leq \epsilon \iff \begin{cases} \|p_i - p_j\| \leq \epsilon_p \\ |v_i - v_j| \leq \epsilon_v \\ |\psi_i - \psi_j|_{360^\circ} \leq \epsilon_\psi \end{cases} \quad (1)$$

where $\epsilon = \langle \epsilon_p, \epsilon_v, \epsilon_\psi \rangle$. The first component of the distance is the Euclidean distance between the two 2D position vectors of the two states. The second distance is the difference in absolute value between the magnitude velocity vectors of the two states. The third distance is the difference between two angles measured in degrees. To ensure that the difference stays within the interval $[0, 360]$, we take the modulo on 360° (we use the notation $|\cdot|_{360^\circ}$ to indicate this). The epsilon values, defining how close two boundary states should be, are configured depending on the properties of the track and of the autonomous vehicle being simulated.

2.2 Boundary State Pairs Search

Algorithm 1 shows the high level steps of the search algorithm. The algorithm evolves one individual at a time,

where the individual is a pair of states.

The main parameters of the algorithm are the number of iterations N , the number of restarts R and the sequence length L . The number of restarts R is the number of seed states that the algorithm samples and evolves (hence, R is the maximum number of boundary states stored in the archive when the algorithm terminates). The number of iterations N corresponds to the maximum number of executions of each individual for each restart. The sequence length L is the maximum number of mutations of each individual in each iteration. This parameter needs to balance the mutation strength with the number of state pairs evaluation, avoiding overshooting the failure boundary while, at the same time, preventing the execution of too many state pairs that would unnecessarily consume the search budget. The output of the search algorithm is a set of boundary state pairs stored in the archive A .

The search algorithm consists of two main phases, i.e., the initialization of a seed state (lines 4–15) and its evolution (lines 17–33). The objective of the initialization phase is to generate a pair of states where the driving model succeeds; the subsequent evolution phase, starting from such pair of states, takes care of turning it into a boundary state pair. In the first phase, the algorithm starts by sampling a state from a reference trace Tr , by calling the `SAMPLEVALIDSTATE` function at line 4. The function also has a state validity filter, to ensure that the initial states are valid. A reference trace is a trajectory of states that we collect by running an autopilot with global knowledge of the track and of the state of the vehicle. The autopilot drives by following nominal trajectories i.e., it drives the vehicle at the center of the lane along the waypoints of the track. Alternatively, if the reference trace is not available, seed states can be sampled at random or can be obtained from a driving model M , with a downstream validity check that filters out invalid states.

The loop at lines 5–7 mutates the seed state s_1 until the resulting state s_2 is valid. As described below in Section 2.2.1, the mutation operator ensures closeness of the boundary states by construction. At line 8, the algorithm builds the individual b consisting of the two states s_1 and s_2 ². The `EXECUTE` function at line 9 places the vehicle in the two states of b , executes the two simulations, and collects the results. In particular, it returns two boolean values, success_1 and success_2 , indicating whether the driving model M succeeds at the lane-keeping task. The `if` statement at lines 10–12, checks if the boolean values of success_1 , success_2 have opposite values (XOR logical operator, indicated as \otimes), which indicates a boundary state pair has been found and stored into the archive A (line 11) after checking for duplicates (i.e., $b \notin A$). When either of the two executions, or both, fail, we restart the search (lines 13–15), while the evolution loop (lines 17–33) is performed only when in both states of the pair b the driving model succeeds.

The evolution phase starts by creating a list of pairs (line 17), initialized with the seed, and by assigning the counter variable for the following while loop (line 18). Such loop (lines 19–33) iteratively mutates the last pair in pairs for L times. The `MUTATE` function at line 21 outputs a new

2. We use the term “individual” to indicate a state pair, to follow the convention of the search-based literature where an individual is the object being evolved [28].

individual for every execution (i.e., \hat{b}), mutating both states. The function also returns a boolean variable (i.e., *valid*), indicating whether it was possible to obtain a valid state pair by mutation. If at any point during the sequence an invalid pair is generated, the algorithm breaks the **for** loop (if statement at lines 22-24) and goes directly to line 27. At this point the algorithm performs a binary search operation on the *pairs* list to find a boundary state pair. The function **BINARYSEARCH** executes the last individual and if in both its states the driving model fails, it chooses an individual in the middle of the list. Then, it proceeds recursively with the head or the tail of the list depending on whether the middle individual consists of two states where the driving model fails or succeeds, respectively. When in one state the driving model fails and in the other it succeeds, a boundary state pair is found. The function **BINARYSEARCH** returns the number of executions of individuals into the variable *it*. Moreover, it returns a positive index into the variable *idx* if it found a boundary state pair. In such a case, the algorithm adds the corresponding boundary state pair (*pairs*[*idx*]) to the archive after checking for duplicates (if statement at lines 29-32) and restarts the search. If no boundary state is found, the **while** loop at lines 19-33 restarts by mutating the last individual of the *pairs* list until the budget of *N* iterations expires.

2.2.1 State Mutation

In Algorithm 1 there are two calls to the **MUTATE** function. The first one (line 6), mutates a single state into a new one, while the second one (line 21) mutates a pair of states. Both ensure closeness of the resulting state(s) to the initial one(s).

Definition 4 (State Mutation Function). Given two valid states $s_i, s_j \in \mathbb{S}^3$ that satisfy closeness (i.e., $\text{dist}(s_i, s_j) \leq \epsilon$), a mutation is a function $\mu : \mathbb{S} \times \mathbb{S} \rightarrow \mathbb{S}$ that modifies the state $s_i = \langle \mathbf{p}, \psi, v \rangle$ such that the resulting state $\hat{s}_i = \mu(s_i, s_j) = \langle \hat{\mathbf{p}}, \hat{\psi}, \hat{v} \rangle$ has the following properties:

- 1) $d(\hat{\mathbf{p}}, T) \geq d(\mathbf{p}, T)$;
- 2) $\hat{v} \geq v$;
- 3) $|\theta(\hat{\mathbf{p}}, \psi, T)| \geq |\theta(\mathbf{p}, \psi, T)|$;
- 4) at least one of the three inequalities 1), 2), 3) is strict;
- 5) $\text{dist}(\hat{s}_i, s_j) \leq \epsilon$;
- 6) \hat{s}_i is valid.

The mutation function μ changes the position \mathbf{p} , the velocity v or the orientation ψ of the given state s_i , such that the resulting state has higher values, in absolute terms, of d , v or $|\theta|$, while preserving closeness and validity. Changes to the velocity magnitude can be made directly on the given state, while in order to change the distance from the center d and the relative orientation θ , the mutation function μ needs to change the position \mathbf{p} and/or the orientation ψ of the vehicle.

In the following, we describe the mutation function for each component of the state, i.e., position, orientation and velocity. After mutating one of these three components, chosen at random, a second and then a third mutation is applied to each of the two remaining components with probability 0.3.

3. In case of the mutation function at line 6 in Algorithm 1, $s_j = s_i$.

Mutate position. The position vector \mathbf{p} of the state s_i being mutated has two components, i.e., p_x and p_y . The position mutation operator changes either of the two components, chosen with equal probability (e.g., p_x). In order to respect the closeness condition, \hat{p}_x is chosen in the interval $(p_x - \epsilon_p, p_x + \epsilon_p)$. Then, we determine the other component of $\hat{\mathbf{p}}$ (e.g., \hat{p}_y) by satisfying the following Euclidean distance inequality:

$$\sqrt{(\hat{p}_x - q_x)^2 + (\hat{p}_y - q_y)^2} \leq \epsilon_p \quad (2)$$

where q_x and q_y are the X and Y components of the position vector of the other state in the boundary pair, s_j . To make sure that the mutation increases the current XTE d , we randomly sample position components $\langle \hat{p}_x, \hat{p}_y \rangle$ from their respective ranges and compute the XTE \hat{d} for each resulting position vector $\hat{\mathbf{p}}$. We use a finite budget to determine whether it is possible to obtain a higher XTE, while also checking that the resulting state \hat{s}_i is still valid. If the outcome of both checks is positive, we deem the mutation successful and change the position components of s_i accordingly.

Mutate orientation. Given the orientation ψ_i of state s_i , the orientation mutation operator applies a mutation such that the closeness and validity constraints hold.

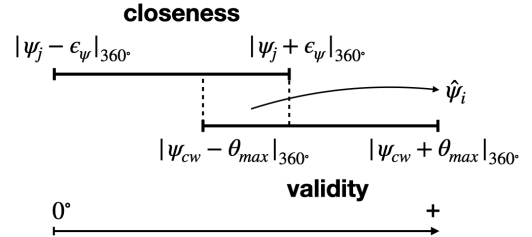


Fig. 3. Example of overlap between closeness and validity constraints concerning the “mutate orientation” function. Assuming that the orientation angle ψ increases with the x direction, the two ranges overlap, determining the new range for the mutated orientation $\hat{\psi}_i$ (delimited by the two dashed lines).

The closeness constraint regards the orientation of the other state of the pair, i.e., ψ_j , limiting the new orientation $\hat{\psi}_i$ to be ϵ_ψ -close to ψ_j . On the other hand, the validity constraint on the mutated orientation $\hat{\psi}_i$ depends on the orientation of the closest waypoint ψ_{cw} , whose angle w.r.t. the new orientation must not be greater than θ_{max} . In Figure 3 there is overlap between the two ranges, hence we choose the new orientation $\hat{\psi}_i$ in the interval determined by the higher of the two interval lower bounds (i.e., $|\psi_{cw} - \theta_{max}|_{360^\circ}$ for the example in Figure 3) and the lower of the two interval upper bounds (i.e., $|\psi_j + \epsilon_\psi|_{360^\circ}$ for the example in Figure 3).

In computing the intersection between closeness and validity intervals, we have to carefully consider that angles are defined modulo 360° . For instance, with $\epsilon_\psi = 7.2$ (i.e., $360 \cdot 2\%$), $\theta_{max} = 20^\circ$, $\psi_j = 350^\circ$ and $\psi_{cw} = 15^\circ$, the resulting overlap range would be $[355^\circ, 35^\circ]$, and we would deem that there is no overlap as the upper bound is lower than the lower bound. To properly handle such cases, we have to represent the overlap range as $[355^\circ, 360^\circ] \cup [0^\circ, 35^\circ]$. Only by doing this we correctly find that in such case an

overlap between validity and closeness ranges does exist, being the range $[355^\circ, 357.2^\circ]$.

When it is not possible to determine an overlap range, the mutation operator fails. We use a finite budget to randomly sample orientation values within the overlap range until the new relative orientation $\hat{\theta}$ is greater than the current relative orientation θ . If the search finds such a value, we replace the orientation ψ_i with new orientation $\hat{\psi}_i$ as it respects all the constraints.

Mutate velocity. The velocity mutation operator mutates the magnitude of the velocity vector. In particular the mutation needs to satisfy the following conjunction of constraints:

$$|v_i - v_j| \leq \epsilon_v \wedge v_i \leq v_{max} \wedge \hat{v}_i > v_i \quad (3)$$

where the first constraint represents closeness, the second validity, and the third the increased velocity requirement. The constraint system is solvable if $v_i < v_{max}$; we randomly choose a solution that satisfies all the constraints, \hat{v}_i , which is the mutated magnitude velocity. We then compute the components of the velocity vector \mathbf{v} on the reference system of the vehicle XY_{car} . We set $v_x = 0$, $v_y = \hat{v}_i$, since only the vertical component of the velocity is non-zero in the coordinate system of the vehicle. To compute the absolute velocity vector on the reference system XY_{abs} we carry out a rotation of axis using the orientation of the vehicle ψ [29].

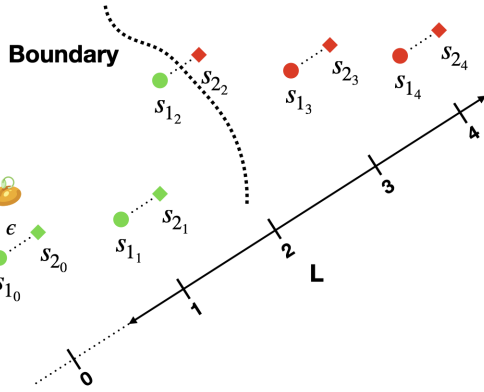


Fig. 4. Example of binary search to find a boundary state pair. The initial seed state is the pair (s_{10}, s_{20}) , with s_{20} being the more challenging state of the pair. The initial pair is consecutively mutated $L = 4$ times (difficulty increases going towards the right). Binary search efficiently finds the pair that sits on the boundary, i.e., (s_{12}, s_{22})

2.2.2 Binary Search

In Algorithm 1 at line 27, the search algorithm calls the binary search function on the sequence of mutations *pairs* generated from the seed b . Figure 4 illustrates the process on a sequence of $L = 4$ mutations. Applying consecutive increasing mutations (i.e., mutations that increase either d or v or θ) to a starting pair of states makes the pair increasingly more difficult for the driving model. We measure the difficulty of a state by using the maximum cross track error (XTE) across a given simulated scenario, which is defined as the distance between the center of mass of the vehicle and center of the lane [16, 30]. The XTE is zero when the vehicle drives at the center of the lane and it is maximum when

the vehicle goes out of bound ⁴ [31, 32] to the sequence of XTE values upon consecutive increasing mutations. Results show that there is an overall positive trend of XTE values at increasing mutation strength.

Depending on the length of the sequence, the resulting pairs will move away from the initial seed and potentially far from the boundary. The idea is to apply a binary search to look for the pair that sits on the boundary, given that the pairs in the sequence are sorted by their difficulty ⁵, although it might not be the most challenging one.

In Figure 4, the first state ($idx = 0$) is the seed, where in both states the model M is successful by construction. The last state ($idx = 4$) is the most challenging and the driving model fails in both states. Then, the binary search function executes the pair at index $idx = 2$. In this example, the model succeeds in \hat{s}_1 and fails in \hat{s}_2 . The two states in the pair are close and valid by construction, forming a boundary state pair.

2.2.3 Individual-level Mutation

In the previous sections we described how to mutate a state, but Algorithm 1 evolves individuals that are state pairs, rather than single states. Given a state pair $b = \langle s_1, s_2 \rangle$, the individual-level mutation function starts by mutating s_2 , producing a new valid state \hat{s}_2 close to s_1 . Since the pair $\langle s_1, s_2 \rangle$ consists by construction of a less challenging state s_1 and a more challenging state s_2 (see Algorithm 1), during the evolution of an individual we want to preserve such a relation between the elements of the pair, which makes it easier to cross the failure boundary with the most challenging state (\hat{s}_2) while keeping the other state (\hat{s}_1) within the boundary (see Figure 4).

Hence, we compute the difference operator $\delta_s = \hat{s}_2 - s_2 = \langle \delta_p, \delta_\psi, \delta_v \rangle$ and apply it to $s_1 = \langle p_1, \psi_1, v_1 \rangle$, obtaining $\hat{s}_1 = \langle p_1 + \delta_p, \psi_1 + \delta_\psi, v_1 + \delta_v \rangle$. Then, we check whether \hat{s}_1 is valid. In particular, the position is the most critical parameter since it also affects the relative orientation. If \hat{s}_1 is not valid, e.g., because adding the deltas resulted in a position that is out of the lane or the orientation is invalid, the individual-level mutation operator discards both states \hat{s}_2 and \hat{s}_1 , and re-attempts to mutate the pair until either both resulting states are valid or the budget expires.

2.3 Driving Model Improvement

The second step of our approach (i.e., step ②) consists of improving the driving model of the ADS. We first place the expert pilot on the non-recoverable states of the boundary state pairs of the driving model. The expert pilot labels each frame of the simulation with the right steering angle, as it can avoid the failures that affect the ADS under test.

In the next phase, we retrain the driving model from scratch by using the union of the initial dataset used for training the model, and the labeled dataset resulting from the placing the expert pilot on the boundary state pairs of the driving model.

4. We empirically validated the hypothesis of the state difficulty increasing monotonically with the mutation strength, by applying the Mann-Kendall trend test (with a significance level $\alpha = 0.05$)

5. In the presence of non-monotonic behaviors, the algorithm returns a boundary state pair (if it exists)

3 EMPIRICAL EVALUATION

To assess the existence and the practical benefits of boundary state pairs, we consider the following research questions:

RQ₁ (Existence): *Do boundary state pairs exist in the training track for well-behaving driving models?*

In our analysis we are interested in evaluating *well-behaving* driving models, i.e., models that, in nominal conditions, drive well in the driving scenario they have been trained on (in our evaluation, a closed driving track with asphalt roads, surrounded by green grass and sunny weather [8, 9]). For nominal conditions, we mean that the vehicle is placed in the starting position of the driving track, with zero velocity and oriented in the direction of the road. We say that a driving model drives well in nominal conditions if it is able to keep the lane for 1200 simulation steps (which correspond to two consecutive laps and 1 minute of driving at 20 fps).

RQ₁ aims at assessing whether such models exhibit boundary state pairs. A positive answer would imply that challenging driving scenarios exist even in a failure-free track where the driving model is well-behaving. Therefore, testing driving models does not necessarily require the manipulation of the environment (e.g., by generating new driving tracks).

Metrics. To assess the existence of boundary state pairs given a well-behaving driving model, we simply measure the number of boundary state pairs resulting from the search process.

RQ₂ (Comparison): *How do boundary state pairs of driving models of different qualities compare with each other? Do boundary state pairs discriminate between them?*

In RQ₂, we compare boundary state pairs of driving models with different degrees of performance. All the driving models respect the requirement of being able to complete the training track in nominal conditions (i.e., they are well-behaving) but they differ in terms of the amount of time they have been trained (i.e., they have different validation losses). We aim to study the extent to which boundary state pairs discriminate between high- and low-quality driving models. We expect boundary state pairs of high-quality driving models to be more challenging and extreme than those of low-quality driving models.

Metrics. We use two metrics for comparing boundary state pairs. Such metrics are system-level since the search process executes the driving model within the system (i.e., the vehicle driving along a track).

The first system-level metric we consider is *recoverability*, that measures to what extent the driving model M_A is able to recover from the boundary state pairs of the driving model M_B . For instance, let us suppose that the driving model M_B has N boundary state pairs in the training track. We measure the recovery percentage of the driving model M_A in each of them, as the number of times M_A succeeds when placed in a certain state, divided by the number of runs the model is executed in each state (to take into account the randomness of the simulation [33]). The recoverability of M_A on the boundary state pairs of M_B is the average recovery percentage across all N boundary state pairs. We consider the recoverability of M_A on both the recoverable

and non-recoverable states of M_B . Hence, recoverability is a measure of how challenging the boundary state pairs of a certain driving model are for another driving model.

The second system-level metric we consider is the *radius*. We define the radius at the state level. In particular, we consider the non-recoverable state of a boundary state pair, since it is more challenging. Given a state s and the training track T we consider three quantities: (1) the distance from the center $d(\mathbf{p}, T)$, (2) the velocity v , and (3) the relative orientation $\theta(\mathbf{p}, \psi, T)$. Our hypothesis is that such quantities determine how difficult a state is. In order to compare boundary state pairs of different driving models we need to both normalize each quantity for each state and to consider a point of reference to compare with. We define the point of reference as $\Omega : (d(\mathbf{p}, T) = 0 \text{ m}, v = 0 \text{ km/h}, \theta(\mathbf{p}, \psi, T) = 0^\circ)$, i.e., the origin. We then normalize distance, velocity and relative orientation based on their maximum values, i.e., respectively $\frac{W}{2}$ for the distance, v_{max} for the velocity, and θ_{max} for the relative orientation. Given a state s and the vector of normalized quantities \mathbf{q} , we compute the radius as the distance between \mathbf{q} and the reference (i.e., $\|\mathbf{q} - \Omega\|$) and normalize by dividing by the maximum distance. If a driving model has N boundary state pairs, the radius of the driving model is the average of all the radii for each boundary state pair. The closer the resulting radius is to 1, the better the driving model. The rationale is that a boundary state pair consists of two states that are close to each other and such that one of them is recoverable by the driving model. If the radius is high, then the boundary state pair is challenging, but nonetheless the driving model is able to recover in one of the states of the pair.

We use the Mann-Whitney rank test [34] to assess the statistical significance of the difference between the radii of different models and the Vargha-Delaney effect size (i.e., \hat{A}_{12}) to assess the magnitude of such difference [35], as previous literature suggests [36].

RQ₃ (Retraining): *How effective are boundary state pairs in improving the performance of a high-quality driving model?*

In RQ₃, we investigate the usefulness of boundary state pairs for improving a well-behaving driving model. In particular, we consider the boundary state pairs of the best model according to both model-level (i.e., validation loss) and system-level metrics (i.e., recoverability and radius). Our hypothesis is that such boundary state pairs identify challenging driving scenarios that improve the generalization capabilities of the model on a separate set of evaluation scenarios. A positive answer to this question would imply that a fixed and failure-free environment scenario contains hidden states that are useful not only to test but also to improve the ADS.

Metrics. To assess the usefulness of the boundary state pairs of a given model, we measure the success rates of the original model on a set of evaluation tracks different from the training one. We then compare them with the success rates of the model retrained with the boundary state pairs dataset on the same evaluation tracks. We use the Wilcoxon signed-rank test [37] to assess the statistical significance of the difference between the success rates of the two models on the same evaluation tracks. As for the radius, we use the Vargha-Delaney effect size (i.e., \hat{A}_{12}) to assess the magnitude of the difference.

3.1 Procedure

Our experimental procedure consists of: (1) searching for boundary state pairs of different driving models (RQ₁ and RQ₂) and computing recoverability and radius for the boundary state pairs of each of them (RQ₂); (2) improving the best driving model according to validation loss and recoverability/radius, by using its generated boundary state pairs (RQ₃).

Across all research questions, we compare our approach with a 1 + 1 evolutionary algorithm (which we indicate as (1+1)-EA, hereafter). Indeed, the state of the art in testing ADSs are search-based approaches [7, 8, 22]; among such approaches we choose (1+1)-EA as it is the most efficient. Moreover, being based on a single individual, it is directly comparable to our approach that only evolves one state pair (i.e., one individual) at a time. In a given iteration, (1+1)-EA mutates the current individual; then, between current and mutated individuals, the individual selected for the next iteration is the one with the highest XTE. To make the comparison with our approach fair, the mutation operators of (1+1)-EA are the same as with our approach.

3.1.1 Implementation and Test Object

We implemented our approach in a Python tool called GENBO, which is publicly available [25]. The test object of our study is the popular DNN-based model Dave-2 [19]. Dave-2 is a robust lane-keeping model that has been previously used in several testing works in the literature [6, 8, 9, 16, 20, 21, 22, 23, 24]. We use the Donkey CarTM open-source framework [38, 39] built with Unity [40]. The simulator has been used in previous work to test both supervised learning and reinforcement learning models [16, 41]; it is also featured in a recent survey as one of the prevalent open-source simulation platforms for online ADS testing [3].

3.1.2 Model Training

The first step to train a DNN-based driving model is to collect a dataset of labeled images captured by the camera of the vehicle driving along a track. We used the default closed track provided by the simulator, and, as expert pilot, we resorted to an autopilot with global knowledge of the track to automatically label the images with steering angle values. We decide the throttle command at runtime, both for the DNN-based driving model and the autopilot, via a linear interpolation between the minimum velocity (10 km/h) and the maximum velocity (30 km/h) so that the vehicle decreases its velocity when the steering angle increases (e.g., in a curve) [30].

As autopilot, we used a PID controller [42] and tuned the Proportional, Integral and Derivative constants for the specific track. We made the PID controller drive the vehicle for three laps of the track, collecting approximately 5k labeled images. We used an 80/20 training/validation split to train the Dave-2 model until convergence (we stopped the training after 10 epochs of no improvements of the validation loss). Furthermore, we saved a checkpoint every epoch of validation loss improvement. Of those models we kept four models we used for testing, at decreasing validation losses. The first model, i.e., M_1 , is the first well-behaving model. The last model, i.e., M_4 , is the model with the best

validation loss. We selected the two models in between, i.e., M_2 and M_3 , by looking at the validation loss. In particular, we kept a model whenever its validation loss decreased by at least 10% w.r.t. the previously considered model (i.e., M_1 and M_2 respectively).

3.1.3 Boundary State Pairs Search

We executed GENBO and (1+1)-EA for the driving models obtained in the previous step, plus the autopilot, which we expect to perform better than the DNN-based models given its global knowledge of the track. We used the autopilot to collect the reference trace Tr we use in Algorithm 1 to sample valid states. As hyperparameters for GENBO and (1+1)-EA, we used $R = 40$ number of restarts, $N = 10$ iterations for each restart, and we used $L = 3$ as sequence length for GENBO. These choices were informed by preliminary experiments, where we found that they achieve a favorable trade-off between computation time and the effectiveness of the search in finding boundary state pairs for the most robust driving models. When placing the driving model on a certain state, we consider the state recoverable if the model is able to keep the vehicle in lane for at least 12.5s which, at 20 fps, corresponds to 250 simulation steps. We chose $\epsilon_p = W \cdot 10\%$, with $W = 4m$ being the width of the lane in the track we use. We set the maximum velocity $v_{max} = 30 \text{ km/h}$, and $\epsilon_v = v_{max} \cdot 10\%$. Regarding the orientation we chose $\epsilon_\psi = 360^\circ \cdot 2\%$ and $\theta_{max} = 20^\circ$. Such values of ϵ ensure that the two states in the pair are *reasonably* close to each other, effectively representing a boundary state pair if the driving model succeeds in one of the states and fails in the other. To account for the randomness of the search algorithm [36] and of the driving simulation [33], we executed GENBO and (1+1)-EA nine times for each driving model.

To specifically target the randomness of the simulation during the search [43, 44]⁶, when the search algorithm finds a boundary state pair, we re-execute the driving model on each state of the pair three times. If the state pair is a boundary state pair the majority of the times (i.e., 2 out of 3 times), then we consider the state pair a boundary state pair and store it in the archive. At the end of the search process, we iterate over the boundary state pairs in the archive, and we compute a replication percentage across 10 repetitions, i.e., the number of times a state pair is actually a boundary state pair for the driving model under test out of 10 repetitions. For the subsequent analysis we consider as likely boundary state pairs those which have a replication percentage greater than zero. In fact, a boundary state pair is inherently flaky as the two states of the pair are *close*, hence multiple repetitions are necessary to gather evidence of a boundary state pair status, i.e., non-zero boundary state replication percentage.

Each run of the search algorithm consists of at most $N \times R = 400$ driving simulations, plus a variable number of simulations due to replications depending on the number of boundary state pairs in the archive. If we were to discard the replications, we would have 400×5 driving models $\times 9$

6. Across different executions, the ADS might receive a slightly different sequence of images due to synchronization and timing inconsistencies.

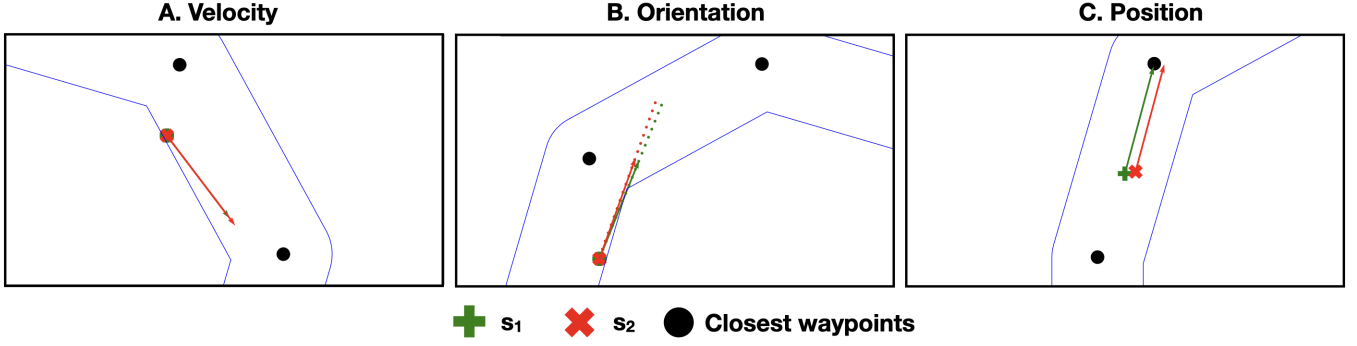


Fig. 5. Examples of boundary state pairs for the model M_{last} . The two blue lines delimit the track T , the green plus sign is the state of the pair s_1 where the driving model succeeds, the red cross indicates the state s_2 where the driving model fails, and black dots are the closest waypoints (in terms of position within T) to the pair. The green and red arrows are the velocity vectors (their length is normalized and scaled $4\times$ for visualization purposes).

repetitions, consisting of a total of $\approx 18k$ driving simulations per search algorithm (i.e., $\approx 36k$ for GENBO and (1+1)-EA).

In order to measure recoverability, we executed each driving model on the boundary state pairs of the others (both recoverable and non-recoverable), keeping track of the number of times a given model is able to recover from a certain state. Also in this case we consider a state recoverable, if the driving model is able to keep the vehicle in lane for at least 250 simulation steps.

3.1.4 Driving Model Improvement

Table 2 shows the list of tracks along with their features (Track Features macro-column). In particular, we measured the curvature of the track (Column 1), i.e., the inverse of the minimum radius of the circles going through each sequence of three consecutive waypoints [9], and the number of turns (Column 2), where a turn is a change of direction between consecutive waypoints by more than 5° . The first track, i.e., T_1 , is the training track, with a curvature of 0.29 and 4 turns; the second track, i.e., T_2 is the specular version of T_1 . To generate the other evaluation tracks, i.e., T_3 — T_9 , we randomly mutated the waypoints of T_1 until one of the following condition is true: (1) the distance between T_1 and the candidate track T_c (i.e., the Euclidean distance between the waypoints of the two tracks being compared) must be lower than a threshold; (2) the curvature of T_c must be greater than the curvature of T_1 ; (3) the number of turns of T_c must be greater than the number of turns of T_1 . The distance constraint imposes that the resulting track is similar to T_1 , while the curvature and the number of turns constraints ensure that it is more challenging to drive.

We executed the two best driving models (as they can no longer be improved by the training dataset as opposed to lower-quality models) on the evaluation tracks, T_2 — T_9 , to collect the success rates. We used 100 simulations, and we considered a simulation a success if the driving model is able to keep the vehicle in lane for 600 simulation steps (which approximately corresponds to one lap of the track at the maximum possible speed, and 30s of driving at 20 fps). For each search repetition of a search algorithm on a given driving model, we took the corresponding boundary state pairs in the archive and placed the autopilot in the non-recoverable state of each pair, in order to collect a labeled dataset for retraining. We executed the autopilot for 100

simulation steps per state and discarded the cases where the autopilot was not able to keep the vehicle in lane for that amount of time. Then, we merged the original training dataset and the labeled boundary state pairs dataset for each repetition, splitting the resulting dataset in such a way that the validation set is a superset of the validation set used for the initial training, to mitigate the possibility of regressions on the training track. We carried out retraining with the same hyperparameters of the initial training (random seed included) until convergence, obtaining, for a given driving model, nine retrained driving models, i.e., one for each search repetition (assuming that at least one boundary state pair was found by the search during that repetition). Finally, we executed each of them on the eight evaluation tracks to collect the success rates.

3.2 Results

3.2.1 Existence (RQ_1)

Table 1, macro-column RQ_1 (Existence), shows the number of boundary state pairs the search process found for each driving model, averaged across the 9 repetitions. Both for GENBO (Column 2) and (1+1)-EA (Column 3), we notice that the number of boundary state pairs decreases with the model quality, as measured by the validation loss. The only exception occurs for GENBO between M_3 and M_{last} , but with a small difference in terms of number of boundary state pairs found (the difference between the two is not statistically significant). As expected, it is challenging for the search algorithm to find boundary state pairs for the autopilot (on average 1.33 for GENBO, while (1+1)-EA does not find any in all repetitions), since it is, by construction, robust against changes of the initial conditions of the vehicle. GENBO finds on average six boundary state pairs across all driving models, against two of the (1+1)-EA approach, showing its higher effectiveness in extracting boundary state pairs, especially for the strongest driving models.

Figure 5 shows three examples of boundary state pairs for the best driving model M_{last} . Figure 5.A shows a *velocity* boundary state pair, namely two states that have the same position and orientation, but with different velocity magnitudes (respectively ≈ 24 km/h for s_1 and ≈ 27 km/h for s_2); in this case, a slight change in velocity is enough for the driving model to lose control of the vehicle, resulting

TABLE 1

Results for RQ_1 (Existence) and RQ_2 (Comparison). Bold-faced values indicate a statistically significant difference w.r.t. the radius of M_1 ; underlined values indicate a large \hat{A}_{12} . "NA" indicates that there are no boundary state pairs. Columns 2–3 show the boundary state pairs found by GENBO and (1+1)-EA, while Columns 4–5 show the radius of such points. Columns 6–16 show the recoverability metric for all driving models on the boundary state pairs of GENBO. Likewise, Columns 17–27 show the recoverability metric for (1+1)-EA's boundary state pairs. All values in the table are average across 9 repetitions.

	RQ ₁ (Existence)				RQ ₂ (Comparison)																					
	# Boundary States		Radius		Recoverability (%)																					
					GENBO										(1+1)-EA											
	GENBO	(1+1)-EA	GENBO	(1+1)-EA	M_{1R}	M_{1NR}	M_{2R}	M_{2NR}	M_{3R}	M_{3NR}	M_{lastR}	M_{lastNR}	autopilot _R	autopilot _{NR}	Avg	M_{1R}	M_{1NR}	M_{2R}	M_{2NR}	M_{3R}	M_{3NR}	M_{lastR}	M_{lastNR}	autopilot _R	autopilot _{NR}	Avg
M_1	10.7	5.22	0.63	0.61	–	–	30	20	34	26	17	5	0	0	16	–	–	48	7	13	7	0	0	NA	NA	13
M_2	7.78	2.11	0.72	0.65	99	96	–	–	90	48	65	36	4	0	55	100	100	–	–	80	27	71	29	NA	NA	68
M_3	4.78	1.00	0.75	0.69	100	99	98	84	–	–	94	54	29	18	72	100	100	100	92	–	–	100	57	NA	NA	91
M_{last}	5.89	0.78	0.73	0.62	100	98	94	75	97	62	–	–	32	4	70	100	100	100	100	100	93	–	–	NA	NA	99
autopilot	1.33	0.00	0.75	NA	100	100	100	100	100	99	100	89	–	–	99	100	100	100	100	100	100	100	100	–	–	100
Avg	6.11	1.82	0.72	0.64	100	98	81	70	80	58	69	46	16	5	–	100	100	87	75	73	57	68	46	NA	NA	–

in an out-of-bound event. Similarly, Figure 5.B shows an *orientation* boundary state pair, where the two states differ only for their value of ψ (respectively 21° for s_1 and 19° s_2); in this case, reducing the orientation towards the center of the track limits the agent's space and time to maneuver the vehicle while making the right turn. In Figure 5.C, the position of the two states is different; in s_2 the vehicle is slightly ahead (both in the x and y directions) of s_1 , and since the velocity in the two states is ≈ 30 km/h, the agent is not able to make the turn in time.

RQ_1 (Existence): Boundary state pairs exist for driving models with different performance. In particular, the number of pairs found by the search decreases with the quality of the driving model.

3.2.2 Comparison (RQ_2)

Table 1, macro-column RQ_2 (Comparison), shows the two system-level metrics, i.e., *Radius* (Columns 4–5) and *Recoverability* (Columns 6–27), we used to compare the different driving models under test. All the values in the table (except the Avg columns and row), are averages across the 9 repetitions. Regarding the radius, we observe, for both GENBO and (1+1)-EA, that the metric reaches a plateau after M_2 , suggesting that it is able to discriminate low-quality driving models (i.e., M_1) from good to high-quality driving models (i.e., M_2 – M_{last}). For GENBO the radius of the autopilot is in line with the DNN-based driving models M_2 – M_{last} (there is no statistical difference between the radii of such driving models) while it is statistically different with a large effect size w.r.t. the radius of M_1 (as is the radius of the other driving models). The trend is similar for (1+1)-EA, where there is a statistically significant difference between the radius of M_1 and the radii of both M_2 and M_3 , while, as opposed to GENBO, there is no statistical difference between the radius of M_1 and the radius of M_{last} . This is likely due

to the fact that the radius of M_{last} is estimated with very few boundary state pairs (i.e., less than one per repetition on average), making the estimate less reliable. Overall, the radius seems to be sensitive to the number of boundary state pairs, especially for higher quality models (i.e., M_2 , M_3 , and M_{last}), suggesting that more boundary state pairs adjust the radius estimate towards higher values.

Regarding recoverability, Column 6–15 shows the average recovery percentage for each driving model on the boundary state pairs found by GENBO for the remaining models across the 9 repetitions. For instance, model M_1 has an average recovery percentage of 30% on the recoverable states of M_2 (Column 8, M_{2R}), while it has an average recovery percentage of 20% on the non-recoverable states of M_2 (Column 9, M_{2NR}). Looking at the Avg row, we observe that the average recovery percentage decreases when the model quality increases. Indeed, the recovery percentage ranges from 100% on the recoverable states of M_1 (i.e., M_{1R}) to 46% on the non-recoverable states of M_{last} (i.e., M_{lastNR}). The recovery percentage further decreases when we consider the autopilot driving model. Its boundary state pairs are challenging for the DNN-based driving models, especially the non-recoverable states of the pairs, as the DNN-based driving models only recover on 5% of them on average. The recoverability metric has a similar behavior for boundary state pairs found by (1+1)-EA. Indeed, the Avg row shows a decreasing trend starting from 100% recoverability on the pairs of M_{1R} to 46% recoverability on the pairs of M_{lastNR} .

Column 16 shows the average recovery percentage of each driving model on the boundary state pairs found by GENBO of the remaining models. We observe that the recovery percentage increases from 16% of M_1 to 99% of the autopilot. The recovery percentage of M_2 is more than twice as much that of M_1 (i.e., 55%). The recovery percentage further increases when moving from M_2 to M_3 and then plateaus (the difference between M_3 and M_{last} is negligible, i.e., 2%). The same holds for the boundary

state pairs found by (1+1)-EA (see Column 27). However, the average recoverability seems to be less granular than that measured by the boundary state pairs of GENBO. In particular, the jumps between models of different qualities seem to be more pronounced (e.g., from 13% to 68% from M_1 to M_2 vs 16%–55% with GENBO); moreover, the average recoverabilities of M_{last} and the autopilot are similar (i.e., respectively 99% and 100%), while the boundary state pairs of GENBO better discriminate the two driving models (i.e., the average recoverability is respectively 70% and 99%). This difference might be due to the different number of boundary state pairs found by the two search strategies, which are then used to compute the recoverability metric; a higher number, as in the case of GENBO, seems to suggest a more fine-grained computation of recoverability.

It is interesting to notice how our novel recoverability metric behaves when assessing the relative quality of two models. For instance, let us compare M_1 with M_2 , when considering GENBO as the search strategy (a similar comparison holds when considering (1+1)-EA): M_1 has a recoverability of 30% (resp. 20%) on the recoverable (resp. non-recoverable) states of M_2 , while M_2 has a recoverability of 99% (resp. 96%) on the recoverable (resp. non-recoverable) states of M_1 . This indicates a clear superiority of M_2 over M_1 . The same pairwise comparison can be conducted for all pairs of driving models, consistently showing a very strong discriminative capability of the recoverability metric.

RQ₂ (Comparison): Boundary state pairs discriminate driving models with different qualities. In particular, the radius discriminates low-quality from high-quality models at a gross granularity, while the recoverability metric is more fine-grained and discriminates very accurately the driving quality of each pair of models considered in our study.

3.2.3 Retraining (RQ₃)

Table 2 shows the results of retraining the best driving models, i.e., M_3 and M_{last} (as they show similar performance in terms of radius and recoverability), using the respective boundary state pairs, both for GENBO and (1+1)-EA. In particular, we show the *Success Rate* (%) on the evaluation tracks of the original and retrained models (Column 5–7 for M_3 , and Columns 8–10 for M_{last}). The success rates of $M_{retr.}^3$ and $M_{retr.}^{last}$ for each driving track, are averaged across the 9 repetitions.

For both GENBO and (1+1)-EA, we observe the absence of regressions of the retrained models on the training track T_1 . In fact, the success rate on T_1 remains 100%, both for $M_{retr.}^3$ and $M_{retr.}^{last}$. For GENBO, the success rate of the retrained models improves in all the evaluation tracks where the original model has a non-zero success rate (i.e., see T_2, T_5 – T_7 for M_3 and T_3, T_5 – T_7 for M_{last}). On the other hand, we observe that the M_{last} model, when retrained with the boundary state pairs of (1+1)-EA (Column 10), has lower success rates w.r.t. the original model in T_3 (i.e., 0.143% success rate vs 56%) and T_5 (i.e., 57.29% vs 70%).

On average, the boundary state pairs of GENBO consistently cause an improvement of the success rate, going from a minimum of $2.4\times$ to a maximum of $4.7\times$ for M_3 , and

TABLE 2

Results for RQ₃ (Retraining). Best success rates are highlighted in bold. Columns 2–4 show the features of the 9 evaluation tracks, such as curvature, number of turns and distance w.r.t. the original training track T_1 . Columns 5–7 show the success rates for the M_3 driving model before retraining (Column 5) and after retraining, respectively with the boundary state pairs found by GENBO (Column 6) and (1+1)-EA (Column 7). Likewise, Columns 8–10 show the success rates for the M_{last} driving model before and after retraining. The success rate values are averaged across 9 repetitions.

	Track Features			Success Rate (%)			Success Rate (%)		
	Curv.	# Ts.	Dist.	M_3	GENBO	(1+1)-EA	M_{last}	GENBO	(1+1)-EA
					$M_{retr.}^3$	$M_{retr.}^3$		$M_{retr.}^{last}$	$M_{retr.}^{last}$
T_1^*	0.29	4	–	100.0	100.0	100.0	100.0	100.00	100.0
T_2^\dagger	0.29	4	–	16.00	75.00	56.00	0.000	0.000	41.43
T_3	0.34	5	20.13	0.000	0.000	0.000	56.00	72.00	0.143
T_4	0.32	5	19.08	0.000	0.000	0.000	0.000	0.000	11.57
T_5	0.28	5	29.31	21.00	86.56	70.60	70.00	93.00	57.29
T_6	0.31	5	33.13	0.000	19.67	0.000	1.000	16.00	5.286
T_7	0.28	5	31.43	33.00	80.67	55.60	23.00	78.00	40.43
T_8	0.39	7	29.28	0.000	0.222	0.000	0.000	1.000	0.000
T_9	0.33	5	31.18	0.000	0.000	0.000	0.000	0.000	0.000
Avg	–	–	–	8.750	32.76	22.78	18.75	32.50	19.52

* Training track † Specular version of the training track

from a minimum of $1.3\times$ to a maximum of $16\times$ for M_{last} . For both models the difference between the success rates of the original vs the retrained model is statistically significant with a small \hat{A}_{12} (if we were to remove the zeros for T_3, T_4, T_8 and T_9 for M_3 and T_2, T_4, T_8 and T_9 for M_{last} , the effect size would be large). Indeed, some of the testing tracks are very challenging for M_3 and M_{last} (i.e., success rate of 0%), and on them retraining with the boundary state pairs results in an improvement of the success rate only on a few of them (i.e., see T_6 for M_3 and T_8 for M_{last}). Overall, the success rate of the retrained models is on average, across all evaluation tracks, $3.7\times$ the success rate of the original model for M_3 and $1.7\times$ for M_{last} . Considering the boundary state pairs of (1+1)-EA, we observe an overall improvement of $2.6\times$ for M_3 , and only a slight improvement for M_{last} (i.e., $1.04\times$ on average). However, for both driving models the difference between the success rates of the original vs the retrained models is not statistically significant, suggesting that the boundary state pairs of (1+1)-EA are not effective enough to significantly improve strong driving models.

RQ₃ (Retraining): The boundary state pairs of our best driving models, when extracted by GENBO, are effective at improving the model through retraining. On average, the success rate of the retrained model on a set of evaluation tracks is between $1.7\times$ and $3.7\times$ higher than the original success rate, with a maximum improvement of $16\times$.

3.3 Threats to Validity

External Validity. Using one simulator and one ADS poses an external validity threat. To mitigate this issue we selected

a widely used ADS from the testing literature [6, 8, 9, 16, 20, 21, 22, 23, 24] (i.e., Dave-2 [19]) and we considered it at different training levels. As driving simulator we selected the popular open-source driving simulator Donkey Car™, also used in previous studies in ADS testing [16, 41]. Both the simulator and the ADS we selected represent the state-of-the-art in ADS testing as a recent survey suggests [3]. Another external validity threat is the limited number of evaluation tracks we used to measure the improvement of the retrained driving model. We addressed this threat by generating a diverse set of evaluation tracks where the original driving model displayed a wide range of success rates, i.e., ranging from high (70%) to low (0%).

Focusing on lane-keeping task, also poses an external validity threat. Although our approach might not generalize to other tasks (such as pedestrian avoidance [15] or collision avoidance [10, 13]), it only requires the formalization of the concept of state as well as the definition of the closeness and validity constraints. Our overall approach of searching for boundary state pairs and improving the driving model is generally applicable to any driving task, although the definition of the closeness and validity constraints needs to be specific to the state variables at hand.

Internal Validity. We compared all the driving models under identical settings of the hyperparameters of the search algorithm and we evaluated the improvement of the retrained model on the same evaluation tracks.

Conclusion Validity. We executed the search process multiple times with different random seeds to account for the randomness of the search algorithm. We also ran each driving simulation under the same conditions multiple times to both assess the reliability of a boundary state pair and to measure the success rate. We used rigorous statistical tests to draw our conclusions.

Reproducibility. The source code, the driving simulator, and the test objects are publicly available in our replication package [25].

4 RELATED WORK

Simulation-based Testing. Simulation-based testing is a prominent approach in the literature to test the reliability of autonomous driving systems (ADSs) [3]. Several techniques have been proposed by researchers. ASFAULT combines procedural content generation and search-based algorithms to generate virtual roads. Abdesslem et al. [12] use decision trees to guide the search towards the critical features of the environment. Calò et al. [13] introduce the concept of avoidable collisions and proposed two search-based algorithms to generate them. Riccio et al. propose DEEPJANUS [8] that use multi-objective optimization to generate challenging and diverse virtual roads; moreover, they propose DEEP-HYPERION [9, 45] that outputs an explanatory map of the road features that cause a misbehavior. SAMOTA [11] and INDAGO [41], respectively for Deep Learning (DL) and Reinforcement Learning (RL) systems, use a surrogate model to predict the output of the driving simulator in order to efficiently generate critical scenarios. Similarly, Giamattei et al. [46] propose CART that builds a causal model of the environment and queries the model to execute only the most promising test cases in the simulator. Recently,

MORLOT [10] and DEEPCOLLISION [47] use reinforcement learning (RL) to manipulate the large input space of the driving simulation, while Doreste et al. [48] use RL to train an intelligent testing agent. Our approach GENBO differs from the above in three distinct ways: (1) GENBO starts from a failure-free driving scenario to extract hidden and challenging driving conditions; (2) GENBO mutates the driving conditions of the ego vehicle instead of acting on the environment; (3) GENBO uses the boundary state pairs associated with challenging driving conditions to improve the ADS under test. Such differences make GENBO (i.e., a *driving-condition-mutating* approach) and the *scenario-mutating* approaches described above, complementary. Indeed, scenario-mutating approaches can explore challenging driving scenarios within the scenario space, while driving-condition-mutating approaches can extract challenging driving conditions within those scenarios.

Boundary Input Generation. Instead of generating individual driving scenarios that induce a failure of the ADS under test, researchers proposed search-based approaches to find boundary inputs [49], i.e., inputs that trigger different behaviors of the ADS under test. For instance, Mullins et al. [17] use an adaptive search algorithm to discover performance boundaries of the ADS. Tuncali et al. [18] employ an approach called rapidly-exploring random trees to generate a pair of configurations where a collision is avoidable/unavoidable. Riccio et al. [8] use a multi-objective search algorithm to generate a road shape where the ADS starts to misbehave. In the approach by Biagiola et al. [50], a test case is a pair of environment configurations and the objective is to find boundary pairs such that in one of them the RL agent under test can adapt to the new environment, while in the other it cannot. While such approaches search for boundary pairs of the environment, our approach focuses on the boundary pairs of the vehicle the driving model controls.

Similarly to us, Tappler et al. [51] define a boundary state as a state that precedes another non-terminal state in which the RL agent under test misbehaves. However, their algorithm to search for boundary states relies on symbolic reasoning and symbolic exploration of the execution tree, which does not scale to a complex context such as ADS testing, for which we designed a novel, ad-hoc state exploration algorithm.

5 CONCLUSION AND FUTURE WORK

GENBO extracts challenging driving conditions in a failure-free driving scenario by mutating the initial state of the ADS. Experimental results show that boundary state pairs exist even for high-quality and well-trained driving models. By retraining the ADS under test with examples collected from its boundary state pairs, we significantly improved the success rate of the retrained model on a separate set of evaluation tracks. In our future work, we plan to extend our study to multiple driving simulators to extend the generalizability of our approach and experiment with alternative search algorithms to look for boundary state pairs. We also plan to investigate other safety-critical case studies other than autonomous driving (e.g., unmanned aerial vehicles [52]), to study the failure boundary of other learning-based systems.

6 ACKNOWLEDGEMENTS

This work was partially supported by the H2020 project PRECRIME, funded under the ERC Advanced Grant 2017 Program (ERC Grant Agreement n. 787703).

REFERENCES

- [1] Society of Automotive Engineers, "Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles." https://www.sae.org/standards/content/j3016_201806, 2016. Online; accessed 18 August 2019.
- [2] Dan Croutch, "The State of Self-Driving Cars: Autonomous Advances." <https://www.techspot.com/article/2644-the-state-of-self-driving-cars/>, 2016. Online; accessed 23 April 2023.
- [3] S. Tang, Z. Zhang, Y. Zhang, J. Zhou, Y. Guo, S. Liu, S. Guo, Y.-F. Li, L. Ma, Y. Xue, and Y. Liu, "A survey on automated driving system testing: Landscapes and trends," *ACM Trans. Softw. Eng. Methodol.*, feb 2023. Just Accepted.
- [4] F. U. Haq, D. Shin, S. Nejati, and L. Briand, "Can offline testing of deep neural networks replace their online testing? a case study of automated driving systems," *Empirical Software Engineering*, vol. 26, no. 5, p. 90, 2021.
- [5] A. Stocco, B. Pulfer, and P. Tonella, "Model vs system level testing of autonomous driving systems: a replication and extension study," *Empir. Softw. Eng.*, vol. 28, no. 3, p. 73, 2023.
- [6] A. Stocco, M. Weiss, M. Calzana, and P. Tonella, "Misbehaviour prediction for autonomous driving systems," in *ICSE '20: 42nd International Conference on Software Engineering*, Seoul, South Korea, 27 June - 19 July, 2020, pp. 359–371, 2020.
- [7] A. Gambi, M. Mueller, 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, (New York, NY, USA), pp. 318–328, ACM, 2019.
- [8] V. Riccio and P. Tonella, "Model-based exploration of the frontier of behaviours for deep learning system testing," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Virtual Event, USA, November 8–13, 2020, pp. 876–888, 2020.
- [9] 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*, pp. 79–90, 2021.
- [10] F. U. Haq, D. Shin, and L. C. Briand, "Many-objective reinforcement learning for online testing of dnn-enabled systems," in *45th IEEE/ACM International Conference on Software Engineering*, ICSE 2023, Melbourne, Australia, May 14–20, 2023, pp. 1814–1826, IEEE, 2023.
- [11] 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*, pp. 811–822, 2022.
- [12] R. Ben Abdesslem, S. Nejati, L. C. Briand, and T. Stifter, "Testing vision-based control systems using learnable evolutionary algorithms," in *2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)*, pp. 1016–1026, May 2018.
- [13] A. Calò, P. Arcaini, S. Ali, F. Hauer, and F. Ishikawa, "Generating avoidable collision scenarios for testing autonomous driving systems," in *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pp. 375–386, IEEE, 2020.
- [14] L. BGR Media, "Waymo's self-driving cars hit 10 million miles." <https://techcrunch.com/2018/10/10/waymos-self-driving-cars-hit-10-million-miles/>, 2018. Online; accessed 23 April 2023.
- [15] M. Borg, R. B. Abdesslem, S. Nejati, F. Jegeden, and D. Shin, "Digital twins are not monozygotic - cross-replicating ADAS testing in two industry-grade automotive simulators," in *14th IEEE Conference on Software Testing, Verification and Validation, ICST 2021, Porto de Galinhas, Brazil, April 12–16, 2021*, pp. 383–393, 2021.
- [16] A. Stocco, B. Pulfer, and P. Tonella, "Mind the gap! a study on the transferability of virtual vs physical-world testing of autonomous driving systems," *IEEE Transactions on Software Engineering*, 2022.
- [17] G. E. Mullins, P. G. Stankiewicz, R. C. Hawthorne, and S. K. Gupta, "Adaptive generation of challenging scenarios for testing and evaluation of autonomous vehicles," *J. Syst. Softw.*, vol. 137, pp. 197–215, March 2018.
- [18] C. E. Tuncali and G. Fainekos, "Rapidly-exploring random trees for testing automated vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, pp. 661–666, IEEE, 2019.
- [19] M. Bojarski, D. D. Testa, D. Dworakowski, B. Firner, B. Flepp, P. Goyal, L. D. Jackel, M. Monfort, U. Muller, J. Zhang, X. Zhang, J. Zhao, and K. Zieba, "End to end learning for self-driving cars.," *CoRR*, vol. abs/1604.07316, 2016.
- [20] A. Stocco, P. J. Nunes, M. d'Amorim, and P. Tonella, "Thirdeye: Attention maps for safe autonomous driving systems," in *37th IEEE/ACM International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10–14, 2022*, pp. 102:1–102:12, 2022.
- [21] A. Stocco and P. Tonella, "Confidence-driven weighted retraining for predicting safety-critical failures in autonomous driving systems," *J. Softw. Evol. Process.*, vol. 34, no. 10, 2022.
- [22] 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*, pp. 45–48, IEEE, 2023.
- [23] M. Hussain, N. Ali, and J. Hong, "Deepguard: a framework for safeguarding autonomous driving systems from inconsistent behaviour," *Autom. Softw. Eng.*, vol. 29, no. 1, p. 1, 2022.
- [24] S. Wang, Z. Sheng, J. Xu, T. Chen, J. Zhu, S. Zhang, Y. Yao, and X. Ma, "ADEPT: A testing platform for simulated autonomous driving," in *37th IEEE/ACM*

- International Conference on Automated Software Engineering, ASE 2022, Rochester, MI, USA, October 10-14, 2022*, pp. 150:1–150:4, ACM, 2022.
- [25] M. Biagiola, “Replication package.” <https://github.com/testingautomated-usi/genbo>, 2024. Online; accessed July 2024.
- [26] M. Harris, “Tesla’s autopilot data deluge.” <https://spectrum.ieee.org/tesla-autopilot-data-deluge>, 2022. Online; accessed 04 January 2024.
- [27] U. Technologies, “Rigidbody velocity in unity.” <https://docs.unity3d.com/ScriptReference/Rigidbody-velocity.html>, 2023. Online; accessed July 2024.
- [28] P. McMinn, “Search-based software test data generation: a survey,” *Softw. Test. Verification Reliab.*, vol. 14, no. 2, pp. 105–156, 2004.
- [29] Wikipedia, “Rotation of axes.” https://en.wikipedia.org/wiki/Rotation_of_axes. Online; accessed April 2023.
- [30] M. Biagiola, A. Stocco, V. Riccio, and P. Tonella, “Two is better than one: digital siblings to improve autonomous driving testing,” *Empir. Softw. Eng.*, vol. 29, no. 4, p. 72, 2024.
- [31] H. B. Mann, “Nonparametric tests against trend,” *Econometrica*, vol. 13, no. 3, pp. 245–259, 1945.
- [32] M. Kendall, *Rank correlation methods*. Rank correlation methods., Oxford, England: Griffin, 1948.
- [33] A. Afzal, D. S. Katz, C. Le Goues, and C. S. Timperley, “Simulation for robotics test automation: Developer perspectives,” in *2021 14th IEEE Conference on Software Testing, Verification and Validation (ICST)*, pp. 263–274, IEEE, 2021.
- [34] 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*, vol. 18, no. 1, pp. 50–60, 1947.
- [35] 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.
- [36] A. Arcuri and L. Briand, “A hitchhiker’s guide to statistical tests for assessing randomized algorithms in software engineering,” *Software Testing, Verification and Reliability*, vol. 24, no. 3, pp. 219–250, 2014.
- [37] F. Wilcoxon, *Individual comparisons by ranking methods*. Springer, 1992.
- [38] “Donkey Car.” <https://www.donkeycar.com/>, 2021. Online; accessed July 2024.
- [39] M. E. Tawn Kramer and contributors, “Self driving car sandbox.” <https://github.com/tawnkramer/sdsandbox>, 2021. Online; accessed July 2024.
- [40] “Unity3d.” <https://unity.com>, 2019. Online; accessed July 2024.
- [41] M. Biagiola and P. Tonella, “Testing of deep reinforcement learning agents with surrogate models,” *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 3, pp. 73:1–73:33, 2024.
- [42] W. Farag, “Complex trajectory tracking using pid control for autonomous driving,” *International Journal of Intelligent Transportation Systems Research*, vol. 18, no. 2, pp. 356–366, 2020.
- [43] M. H. Amini, S. Naseri, and S. Nejati, “Evaluating the impact of flaky simulators on testing autonomous driving systems,” *Empir. Softw. Eng.*, vol. 29, no. 2, p. 47, 2024.
- [44] A. Afzal, D. S. Katz, C. L. Goues, and C. S. Timperley, “Simulation for robotics test automation: Developer perspectives,” in *14th IEEE Conference on Software Testing, Verification and Validation, ICST 2021, Porto de Galinhas, Brazil, April 12-16, 2021*, pp. 263–274, IEEE, 2021.
- [45] T. Zohdinasab, V. Riccio, A. Gambi, and P. Tonella, “Efficient and effective feature space exploration for testing deep learning systems,” *ACM Trans. Softw. Eng. Methodol.*, vol. 32, no. 2, pp. 49:1–49:38, 2023.
- [46] L. Giamattei, A. Guerriero, R. Pietrantuono, and S. Russo, “Causality-driven testing of autonomous driving systems,” *ACM Trans. Softw. Eng. Methodol.*, dec 2023. Just Accepted.
- [47] 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.
- [48] A. Doreste, M. Biagiola, and P. Tonella, “Adversarial testing with reinforcement learning: A case study on autonomous driving,” in *Proceedings of the 2024 IEEE Conference on Software Testing, Validation and Verification (To appear)*, ICST ’24, p. 12 pages, IEEE, May 2024.
- [49] M. Pezzè and M. Young, *Software testing and analysis: process, principles, and techniques*. John Wiley & Sons, 2008.
- [50] M. Biagiola and P. Tonella, “Testing the plasticity of reinforcement learning-based systems,” *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 31, no. 4, pp. 1–46, 2022.
- [51] M. Tappler, F. C. Córdoba, B. K. Aichernig, and B. Könighofer, “Search-based testing of reinforcement learning,” *arXiv preprint arXiv:2205.04887*, 2022.
- [52] S. Khatiri, S. Panichella, and P. Tonella, “Simulation-based test case generation for unmanned aerial vehicles in the neighborhood of real flights,” in *IEEE Conference on Software Testing, Verification and Validation, ICST 2023, Dublin, Ireland, April 16-20, 2023*, pp. 281–292, IEEE, 2023.



Matteo Biagiola is a Postdoc at the Software Institute of Università della Svizzera italiana (USI) in Lugano, Switzerland. He obtained his Ph.D in 2020 from Università degli Studi di Genova, Italy, in a joint collaboration with Fondazione Bruno Kessler, Trento, Italy. He is interested in software testing, with a particular focus on test generation for Web applications and learning-based systems. He serves as a reviewer for major Software Engineering conferences and journals, and was awarded several distinguished reviewer awards (i.e., ICSME 2023, ICST 2024, TOSEM 2023).



Paolo Tonella is Full Professor at the Faculty of Informatics and at the Software Institute of Università della Svizzera italiana (USI) in Lugano, Switzerland. He is Honorary Professor at University College London, UK. Paolo Tonella holds an ERC Advanced grant as Principal Investigator of the project PRECRIME. He has written over 150 peer reviewed conference papers and over 50 journal papers. In 2011 he was awarded the ICSE 2001 MIP (Most Influential Paper) award, for his paper: "Analysis and Testing of Web Applications". His H-index (according to Google scholar) is 68. He is/was in the editorial board of TOSEM, TSE and EMSE. He was Program Co-Chair of ESEC/FSE 2023, and he is Program Co-Chair of ISSA 2025. His current research interests are in software testing, in particular approaches to ensure the dependability of machine learning based systems, automated testing of cyber physical systems, and test oracle inference and improvement.