Empirical Software Engineering (2024) 29:72
https://doi.org/10.1007/s10664-024-10458-4

Two is better than one: digital siblings to improve autonomous driving testing

Matteo Biagiola[1] · Andrea Stocco[2,3] · Vincenzo Riccio[4] · Paolo Tonella[1]

Accepted: 7 February 2024

Abstract

Simulation-based testing represents an important step to ensure the reliability of autonomous driving software. In practice, when companies rely on third-party general-purpose simulators, either for in-house or outsourced testing, the generalizability of testing results to real autonomous vehicles is at stake. In this paper, we enhance simulation-based testing by introducing the notion of *digital siblings*—a multi-simulator approach that tests a given autonomous vehicle on multiple general-purpose simulators built with different technologies, that operate collectively as an ensemble in the testing process. We exemplify our approach on a case study focused on testing the lane-keeping component of an autonomous vehicle. We use two open-source simulators as digital siblings, and we empirically compare such a multi-simulator approach against a digital twin of a physical scaled autonomous vehicle on a large set of test cases. Our approach requires generating and running test cases for each individual simulator, in the form of sequences of road points. Then, test cases are migrated between simulators, using feature maps to characterize the exercised driving conditions. Finally, the joint predicted failure probability is computed, and a failure is reported only in cases of agreement among the siblings. Our empirical evaluation shows that the ensemble failure predictor by the digital siblings is superior to each individual simulator at predicting the failures of the digital twin. We discuss the findings of our case study and detail how our approach can help researchers interested in automated testing of autonomous driving software.

Communicated by: Bibi Stamatia, Bowen Xu, Xiaofei Xie and Maxime Cordy

✉ Matteo Biagiola
matteo.biagiola@usi.ch

Andrea Stocco
andrea.stocco@tum.de ; stocco@fortiss.org

Vincenzo Riccio
vincenzo.riccio@uniud.it

Paolo Tonella
paolo.tonella@usi.ch

[1] Università della Svizzera italiana (USI), Via Buffi, 13, Lugano, Switzerland

[2] Technical University of Munich, Boltzmannstraße 3 Garching near Munich, Munich, Germany

[3] fortiss GmbH, Guerickestraße 25, Munich, Germany

[4] Università degli Studi di Udine, Via Gemona 92, Udine, Italy

Published online: 17 May 2024

Keywords AI testing · Self-driving cars · Simulation-based testing · Digital twins · Deep neural networks · Autonomous vehicles.

1 Introduction

The development of autonomous vehicles (AVs) has received great attention in the last decade. As of 2020, more than $150 billions have been invested in AVs, a sum that is expected to double in the near future (Boutan 2020). AVs typically integrate multiple advanced driver-assistance systems (e.g., for adaptive cruise control, parking assistance, and lane-keeping) into a unified control unit, using a perception-plan-execution strategy (Yurtsever et al. 2020). Advanced driver-assistance systems based on Deep Neural Networks (DNNs) are trained on labeled input-output samples of real-world driving data provided by the vehicle sensory to learn human-like driving actions (Grigorescu et al. 2020).

Before deployment on public roads, AVs are thoroughly tested in the field, on private test tracks (BGR Media 2018; Borg et al. 2021; Cerf 2018; Stocco et al. 2022). While essential for fully assessing the dependability of AVs on the road, field testing has known limitations in terms of cost, safety and adequacy (Stocco et al. 2022). To overcome these limitations, driving simulators are used to generate several real-life edge case scenarios that are unlikely to be experienced during field testing, or that are dangerous to reproduce for human operators (Borg et al. 2021; Koopman and Wagner 2016). Simulation-based testing represents a consolidated testing practice, being more affordable than field testing, yet capable of exposing many bugs before deployment (BGR Media 2018; Borg et al. 2021; Cerf 2018; Stocco et al. 2022).

In this paper, we distinguish two main categories of driving simulators, namely digital twins (DTs) and general-purpose simulators (GPSims). DTs provide a software replica of *specific* real vehicles, that are digitally recreated in terms of appearance, aerodynamics, and physical interactions with the environment (Borg et al. 2021). In the context of mixed-reality testing approaches (Tang et al. 2022; U.S. Department of Transportation 2018), such as Hardware-in-the-Loop and Vehicle-in-the-Loop, the digital twin is connected to physical AV components to further increase the degree of fidelity. In this paper, we consider simulation-based testing where the digital twin is a software replica of a specific real vehicle. Developing a DT is expensive (Kothlow 2021; van Dinter et al. 2022) and can take up to five years (Infinity Simulator 2022). Hence, it remains an exclusive prerogative of big companies such as Uber (Waabi World (2022)), Waymo (Simulation City (2021)) or Wayve (Infinity Simulator (2022)). GPSim are generally designed without the need to faithfully reproduce a specific vehicle or testing scenario, as they rather offer generic APIs to run one or more AVs on virtual road tracks. GPSim such as Siemens PreScan (Software 2022) or ESI Pro-SiVIC (Group 2021) offer a more affordable alternative to the expensive DT development, and are widely used for outsourcing testing tasks to third-party companies (May 2019), for which access to, or customizations of the original DT are not feasible for each individual vehicle (Hu et al. 2023).

Despite affordability, GPSim can be affected by a *fidelity* and *reality gap*, when the simulated experience does not successfully transfer from the GPSim to the reference DT and eventually to the real AV (Hu et al. 2023). These discrepancies can lead to a distrust in simulation-based testing, as reported by recent surveys (Afzal et al. 2021; García et al. 2020; Hu et al. 2023; Tang et al. 2022). While comparative works of GPSim exist in the literature (Kaur et al. 2021; Rosique et al. 2019), cross-simulator testing for AVs is a relatively unexplored avenue for research. Only a recent study (Borg et al. 2021) investigates the use of multiple GPSim for testing a pedestrian vision detection system. The study compares a large set of test scenarios on both PreScan (Software 2022) and Pro-SiVIC (Group 2021) and reports inconsistent results in terms of safety violations and behaviors across these simulators. Consequently, using a single-simulator approach for AV testing might be unreliable, as the testing results are highly dependent on the chosen GPSim.

In this paper, we target the fidelity gap between GPSim and DT by proposing a multi-simulator approach for AV testing called *digital siblings* (DSS). Our approach involves automated test generation and a novel cross-simulator feature map analysis that combines the outcome of several simulator-specific test generators into a unified view. We use DSS as a surrogate model of the behavior of a DT. Our intuition is that agreement among multiple GPSim will increase the confidence in observing the same behavior in DT. On the other hand, in the presence of disagreements, DSS can mitigate or even eliminate the risk of choosing the worst GPSim, which would give poor simulation testing results.

In detail, our case study consists in the automatic generation of test cases, i.e., sequences of road points determining the roads where the AV drives, to test the lane-keeping component of an AV. We then use feature maps to characterize both the structure of such test cases, and the behaviors of the AV in each of them, to group failures by similarity, and to avoid reporting the same failures repeatedly. To account for the specificities of each GPSim, we execute test generation separately for each sibling. Then, we migrate the tests generated for one sibling to the other sibling. Finally, we merge failing and non failing executions based on similarity of features and estimate the overall joint failure probability.

In our case study we use DSS to test three state-of-the-art DNN lane-keeping models, i.e., Nvidia Dave-2 Bojarski et al. (2016); Chauffeur (2016), and Epoch (2016) (the last two were developed by the respective teams in the Udacity challenge competition (Udacity challenge 2020)). We consider as siblings two open-source simulators, namely Udacity (2019) and BeamNG (2022), widely used in previous studies to test lane-keeping software (Gambi et al. 2019; Jahangirova et al. 2021; Riccio and Tonella 2020; Stocco et al. 2020; Zohdinasab et al. 2021). As DT, we adopt an open-source framework (Tawn Kramer 2022) used in previous research (Stocco et al. 2022; Tang et al. 2022; Verma et al. 2021; Viitala et al. 2020; Zhou et al. 2021) featuring a virtual replica of a 1:16 scale electric AV. We evaluate DSS with both *offline* and *online* testing (Haq et al. 2021), i.e., the lane-keeping models are tested both w.r.t. the accuracy of its predictions on labeled individual inputs, and at the system-level for their capability to control the vehicle on several hundreds automatically-generated roads.

Our empirical evaluation shows that, at the model-level, the distribution of prediction errors of DSS is statistically indistinguishable from that of the DT. Overall, at the system-level, the failure probability of DSS highly correlates with the true failure probability of the DT. More notably, the quality of driving measured in DSS can predict the true failure probability of the DT, which suggests that we can use the digital siblings to possibly anticipate the failures of the lane-keeping component of the real-world AV more reliably than with a single GPSim. A practical implication of our findings for software engineers is the usage of digital siblings when testing DNN-based lane-keeping software, to increase the level of fidelity of the observed behaviors and failures. The same recommendation holds for AV testing researchers.

Our paper makes the following contributions:

– **Digital Siblings.** A novel approach for testing DNN-based lane-keeping software that generates road scenarios in multiple general-purpose simulators, and combines their testing outcomes to approximate a digital twin. This is the first solution that leverages a multi-simulator approach to overcome the simulation fidelity gap.

– **Evaluation.** An empirical study showing that the digital siblings are effective at predicting the failures of the AV under test in the digital twin for a physical scaled vehicle in the lane-keeping task.

2 Motivation and Background

In this section, we provide additional motivation for our approach, and we briefly describe the main concepts to understand the rest of the paper. In particular, we discuss the lane-keeping functionality of an AV, and we introduce evolutionary search as a tool to generate challenging test scenarios for AVs.

2.1 Motivation

In practice, test engineers use simulation platforms for testing early releases of their autonomous driving software, prior to real-world physical testing. The gap between simulated and real-world test outcomes hinders trustworthiness in the testing process. Thus, efforts must be made to provide evidence that simulation-based testing campaigns can expose real-world AV failures.

In an ideal scenario, the chosen simulation platform is able to accurately replicate the physics of the AV under test. Such high-fidelity digital twins are used by automotive companies as a proxy for their physical AVs. Under this assumption, the high-fidelity digital twin allows to safely carry out a testing campaign while saving costs and, at the same time, improving the robustness of the software.

However, high-fidelity digital twins are costly to develop and maintain, and not all manufacturers can afford them (those who can are not keen to disclose their high-fidelity digital twins, as these are valuable assets that give them a competitive advantage). Moreover, AV manufacturers outsource most of the testing processes to small/medium companies and such high-fidelity digital twins are not available to them. These companies adopt GPSims as a low-cost alternative for simulation-based testing of AVs.

The goal of our approach is to increase the *reliability* of simulation-based testing of AVs, specifically targeting environments that adopt general purpose simulators that are not designed to represent a specific AV, but rather focus on high-level scenario-based testing. To mitigate this design limitation, we propose a testing methodology employing an ensemble of GPSims. This approach involves aggregating the outcomes of multiple GPSims to mitigate the risks associated with simulator flakiness or representativeness. We combine multiple relatively low-cost simulators to obtain reliable test results as if we used a very costly dataset from the real operation or a high-cost simulator such as a high-fidelity digital twin. Our approach is particularly beneficial when these GPSims exhibit complementary behaviors, allowing them to compensate for each other's weaknesses while combining their strengths. Our research hypothesis is that the combination of complementary GPSims provides a more reliable estimation of testing outcomes than the usage of a single GPSim. In this paper, we present the initial findings supporting this hypothesis, exploring and evaluating one practical implementation of our approach using widely accessible open-source simulation platforms.

We instantiate our approach for testing the lane-keeping component of an AV, implemented with a DNN. The test cases are sequences of road points, which determine the two-lane roads where the AV is supposed to drive autonomously. To assess the benefits of our multi-simulator approach (i.e., DSS), we use the digital twin (DT) of a physical 1:16 scale electric

AV (Tawn Kramer 2022), as a surrogate for the real-world AV behaviors. Indeed, we assume having access only to multiple GPSims as, in practice, a DT is often unavailable. In our evaluation, we validate our hypothesis by comparing the extent to which both DSS and each individual sibling can predict the failures of the DNN lane-keeping component in DT, thus quantifying the reliability of testing.

2.2 Background

2.2.1 Lane-keeping

This paper focuses on testing AVs that perform the lane-keeping functionality from driving samples labeled by humans. Lane-keeping, also called lane-centering or lane-following, is an automated driving assistance feature of an AV to keep the vehicle at the center of the lane. This system can be implemented at different levels, from a warning to the driver when the vehicle crosses one of the lanes up to the driverless version, which steers the vehicle automatically when it detects a departure from the center of the lane.

In this paper we consider the driverless version since it is a crucial component for the safe deployment of AVs on public roads. Indeed, according to a report by NHTSA (2007), off-road crashes due to failures of the lane-keeping component are first in cost ($15 billion) and second in frequency. From a technical standpoint, the lane-keeping task is implemented by *behavior cloning* DNNs, which learn end-to-end from supervised expert demonstrations. The training dataset consists of driving images captured with a camera sensor mounted on board of the vehicle, appropriately labeled with the driving commands of a human driver.

We consider lane-keeping DNN models, such as NVIDIA's Dave-2 Bojarski et al. (2016), that predict the steering angle at which the car should steer to keep the vehicle in lane, given a single driving image. These models are generally trained with stochastic gradient descent (Saad 1998) on stationary datasets, with the goal of minimizing the error between the predicted and the ground-truth steering angles.

Such labels are typically an array of commands, i.e., steering, throttle and brake, although in the simplest case only the steering is provided, while the throttle is determined as a function of the steering and the velocity of the vehicle. Given the dataset, a DNN model, such as the Dave-2 model from Nvidia Bojarski et al. (2016), is trained to predict the label given an image by minimizing the Mean Squared Error (MSE) between the current prediction and the ground-truth label.

2.2.2 Evolutionary Search

Evolutionary or metaheuristic search is a class of techniques that apply randomness and heuristics to find near-optimal solutions to optimization problems Luke (2013). Such techniques are very general, since they only require evaluating how good a candidate solution is. The *goodness* of a solution is called fitness and the objective of the search algorithm is to optimize it (either maximize it or minimize it). The algorithm manipulates a solution to exploit the known parts of the search space, and creates new solutions to explore the parts that are unknown.

Search algorithms have been applied to testing problems and have been particularly effective tools for test generation (Fraser and Arcuri 2012; Panichella et al. 2017; Lukasczyk et al. 2020). In this paper, we use the MapElites search algorithm (Mouret and Clune 2015), implemented in the DeepHyperion tool (Zohdinasab et al. 2021), to generate test cases for the DNN model under test. The algorithm explores the solution feature space at large, in order to provide a comprehensive characterization of the behaviors of the driving model.

3 Multi-simulator AV Testing with Digital Siblings

The goal of our approach is to use digital siblings to test the DNN-based lane-keeping component of an AV, by generating a large set of road scenarios. Our approach takes as input a DNN lane-keeping model M, and uses an existing road generator to test its behavior, by generating roads for multiple driving simulators. The key intuition is that multiple GPSims can better approximate the driving behavior of the AV executed in DT, which we use as a proxy for the behavior of the real-world AV, as opposed to a single-simulator approach.

Our approach supports an arbitrary number of digital siblings. For simplicity of exposition, engineering effort, and evaluation, we describe and experiment it using two simulators. However, we present the most important steps of our approach, i.e., migration (step ❸) and merge (step ❹), in a generic manner that accommodates any number of siblings.

Figure 1 (top) shows an overview of our approach in which two digital siblings, namely DS_1 and DS_2, are used to test the behavior of a driving model under test M, i.e., an end-to-end DNN for lane-keeping. In the first phase, M is either trained or fine-tuned (step ❶) to run on both DS_1 and DS_2, as well as on the target platform (i.e., DT). A test generation phase (step ❷) is executed for each digital sibling, generating road scenarios for each simulator and producing two *feature maps* FM_{DS_1} and FM_{DS_2}. Feature maps group together test cases with similar feature combination values, to reduce redundancy and summarize the AV behaviors in unique feature combination (Zohdinasab et al. 2021, 2022). The value in a feature map cell, displayed in a colored heat scale, represents the average test case outcome, i.e., the behavioral

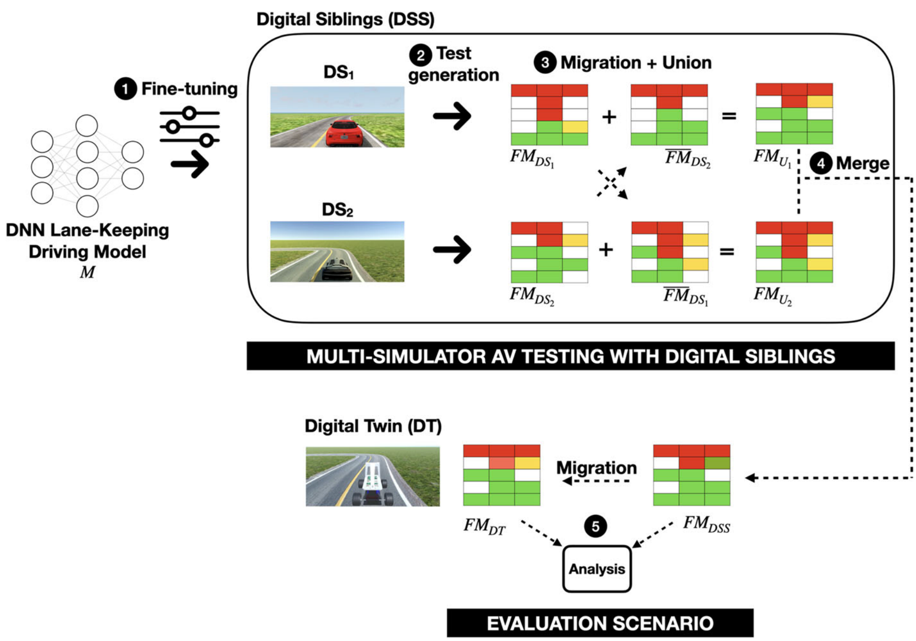

Fig. 1 Overview of our multi-simulator approach and its usage

information about the execution of M in each test scenario (e.g., the failure probability). For each simulator, the test generation algorithm produces test scenarios that are executed to assess the behavior of the driving model M under many different circumstances. Hence, the output of test generation is simulator and model dependent and the feature maps of DS_1 (FM_{DS_1}) and DS_2 (FM_{DS_2}) can be different.

The next step of our approach (step ❸) requires to *migrate* the test cases across simulators. In detail, the test cases in FM_{DS_1} are executed on DS_2, resulting in the feature map $\overline{FM}_{DS_1}$. Similarly, the test cases in FM_{DS_2} are executed on DS_1, resulting in the feature map $\overline{FM}_{DS_2}$. Then, for both DS_1 and DS_2, we compute the *union* of the two feature maps, obtaining FM_{U_1} for DS_1 and FM_{U_2} for DS_2. Both maps contain the same set of test cases, although executed on two different simulators. The final output of the digital siblings (step ❹) is obtained by *merging* FM_{U_1} and FM_{U_2} into the final feature map FM_{DSS}.

Step ❺ assesses the correlation of the FM_{DSS} map with the FM_{DT} map, to evaluate the predictive capability of the digital siblings. Figure 1 (bottom) shows an overview of the evaluation of our approach (detailed later, in Section 4). All the test cases in the final feature map FM_{DSS} are executed (i.e., migrated) on the DT, to obtain the ground truth feature map FM_{DT}.

3.1 Test Scenarios

3.1.1 Representation

We adopted an abstract representation of the road in each driving simulator so that only a sequence of road control points is needed when creating a new road in the driving scene. We follow the representation given by Riccio and Tonella (2020) who defined a two-lane road using a series of *control points* (displayed as red stars in Fig. 2). The control points are interpolated using *Catmull-Rom* splines (Barry and Goldman 1988), giving the road its final shape (yellow solid line).

Figure 2 shows the visualization of a test scenario generated at step ❷. Specifically, the road is defined using nine control points whereas the Catmull-Rom spline only goes through seven of them. This is because a spline segment (e.g., $P_2 - P_3$) is always defined by four control points (e.g., P_1, P_2, P_3, P_4). Since two of them are on either side of the endpoints of the spline segment (e.g., P_1 and P_4), the spline cannot traverse the extreme endpoints (e.g., P_1 and P_9). Hence, P_2 defines the start point of the road (depicted as a black triangle) whereas P_8 defines the end point (depicted as a black square).

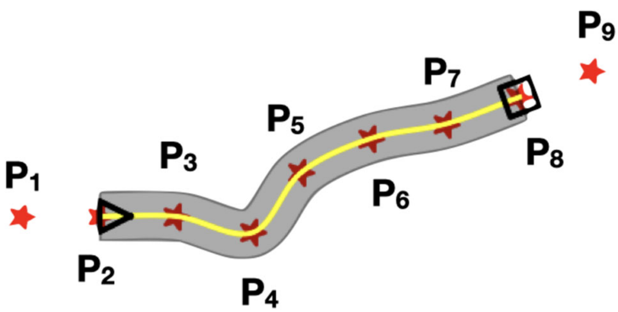

Fig. 2 Example of test scenario for a lane-keeping autonomous driving system

3.1.2 Implementation

The default initial state of each test case involves positioning the vehicle in the first drivable control point (i.e., P_2 in Fig. 2), at the center of the right lane following the road orientation.

We uniformed the 3D rendering of each simulator such that the driving scenarios have the same look and feel: a two-lane asphalt road, where the road is delimited by two solid white lines on each side and the two driving lanes are separated by a single solid yellow line. The road is placed on top of a green plane representing grass. Harmonization of the driving scenarios across simulators ensures that geometrical features are preserved for the collected driving images and that any color transformation applied to them during training preprocessing remains applicable (Bojarski et al. 2016).

3.1.3 Validity and Oracle

After interpolation, a road is deemed *valid* if it respects the following constraints: (1) the start and end points are different; (2) the road is contained within a squared bounding box of a predefined size (specifically 250×250); and, (3) there are no intersections. A test case is deemed *successful* when the vehicle drives within the right lane until the last road control point (e.g., P_8 in Fig. 2). On the contrary, a test case *failure* occurs when the vehicle drives *out of bound* (OOB).

3.2 Creating/Fine-Tuning the Driving Model

3.2.1 Data Collection

For the creation or fine-tuning of a self-driving model (step ❶), a labeled dataset of driving scenes is needed. We automate labeled data collection by resorting to *autopilots* that have *global knowledge* of the driving scenario such as the detailed road geometry and precise vehicle position. In particular, in each simulator, at each step of the simulation, the steering angle of the autopilot is computed by a Proportional-Integral-Differential (PID) controller (Farag 2020) according to the formula:

$$steering = K_P \cdot \mathrm{LP} + K_D \cdot diff_{\mathrm{LP}} + K_I \cdot total_{\mathrm{LP}} \quad (1)$$

where LP stands for *lateral position* (Stocco and Tonella 2020) (in particular, the lateral position is zero when the vehicle drives at the center of the lane). Equation (1) states that the proportional constant K_P acts on the raw error while the derivative constant K_D controls the difference between two consecutive errors and the integral constant K_I considers the total sum of the errors during the whole simulation until the current timestep. Finally, the steering value is clipped in the interval $[-1, +1]$, where -1 means steering all the way to the left and $+1$ to the right (0 means the vehicle goes straight as no steering is applied). The steering values are normalized in order to account for the different simulators that we use in our approach.

The autopilot produces a steering angle label for each image which is used to train the driving model. We aligned the frame rates of the different simulators at 20 *fps* such that, in each simulator, the autopilot collects a comparable number of labeled images. The speed of the vehicle, both for the autopilot and M, is controlled by the throttle via a linear interpolation between the minimum speed and maximum speed so that the car decreases the speed when the steering angle increases (e.g., in a curve). The following formula computes the throttle based on the speed of the vehicle and the steering:

$$throttle = 1 - steering^2 - \left(\frac{speed}{K}\right)^2 \tag{2}$$

where K is set to a predefined low value L when the measured *speed* is greater than a given maximum speed threshold, to enforce strong deceleration; viceversa, K is set to a high value H when the measured *speed* is lower than or equal to the maximum speed threshold, to reduce the deceleration component. From (2), we can see that the throttle is close to 1 (the highest possible value) when the vehicle does not steer ($steering = 0$) and the *speed* is substantially lower than the maximum allowed speed (in this case, $K = H$); when one of the two conditions is false the throttle decreases, because of either deceleration component. Similarly to the steering angle values, we clip the throttle value in the interval [0, 1].

3.2.2 Model Fine-Tuning via Hybrid Training

The next step involves training the model M using all simulators and the data collected in step ❶. Alternatively, if an existing trained model M is available for the target DT, our approach requires *fine-tuning* it for all digital siblings. In both scenarios, we use *hybrid* training based on gradient descent (Bottou and Bousquet 2007).

Hybrid training requires combining the datasets collected for different simulators/platforms into a unified dataset, making sure that each dataset is equally represented (i.e., the unified dataset contains the same number of samples from each simulator/platform specific dataset). Then, the unified dataset is split into training and validation sets (e.g., using the standard 80/20 ratio). The training pipeline is designed in such a way that each image, of dimensions 320×160, is processed according to the simulator/platform it was taken from. For example, images may be cropped differently. Depending on the vehicle size, the front part of the car may, or may not be visible in the frame captured by the camera. Another example of simulator-specific adaptation is the cropping of the above-horizon portion of the image, unnecessary for the lane-keeping task. After cropping, each image is resized to the size required for training, i.e., 320×160.

The training pipeline can be further configured to use plain synthetic virtual images from the driving simulators, or pseudo-real images resembling real-world driving images. The first configuration represents the standard practice in AV testing. In the second configuration, the reality gap due to low photo-realism is reduced by an *image-to-image* transformation that translates the driving images of each simulator into images similar to those captured by the real-world AV during on-road driving. This practice was proposed in the literature (Stocco et al. 2022) and in industry (Bewley et al. 2019) to increase the transferability of the driving model tested in simulation to the real world.

More specifically, this second configuration requires training a CycleGAN model for each driving simulator (Zhu et al. 2017). CycleGAN entails two *generators*, one that learns how to translate images from *simulated* to *real* world (sim2real) and the other that learns the opposite transformation (real2sim). During training of the model, we use the sim2real generator trained for the respective simulator to translate the corresponding training set images. During testing, the sim2real generator translates images at runtime, i.e., during the execution of the simulation. We refer to the translated images as *pseudo*-real, since they are the output of a generative process designed to resemble real images.

Figure 3 shows an example of image translation with a CycleGAN trained for each simulator. The corresponding networks translate an image of a road curve taken in the simulated

Fig. 3 Example of CycleGAN translation for the three simulators

domain (left) to an image belonging to the real domain (right)—the test track of a small scale physical AV. During training and testing of the driving model in a given simulator, we use the generator of the CycleGAN trained for such simulator.

In our evaluation (Section 4), we consider both configurations of our approach, i.e., training using either simulator or pseudo-real images. We refer to the model trained on simulator images as M_S, and the model trained on pseudo-real images as M_R.

3.3 Test Generation

While our approach is compatible with any test generation algorithm, in this paper we adopt the *MapElites* (Mouret and Clune 2015) algorithm implemented in DeepHyperion Zohdinasab et al. (2021), because the output of DeepHyperion is projected to a feature map that characterizes each generated test scenario according to its features. In other words, test cases having equivalent features (e.g., 3 turns and maximum curvature of 0.2) are grouped into the same *cell* of the feature map.

Figure 4 shows an example of feature map generated by DeepHyperion. The roads (i.e., the test cases) in the map are characterized by two structural features, i.e., the *number of turns* in the road (x axis) and the *curvature* of the road (y axis), the latter defined as the minimum radius of the circles going through each sequence of three consecutive road points (Zohdinasab et al.

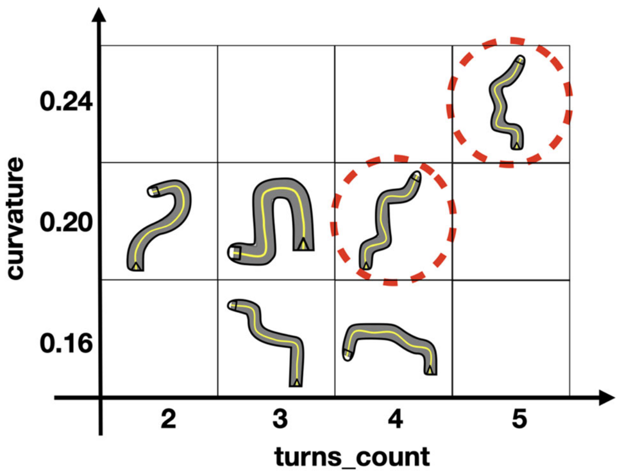

Fig. 4 Example of feature map by DeepHyperion. The two axes represent structural features of the generated roads (i.e., curvature and number of bends)

Algorithm 1 DeepHyperion algorithm

Input : M, DNN model under test;
S, Simulator instance;
P_s, Population size;
N, Number of iterations.
Output: F_m, feature map.

1 $M \leftarrow$ INITFEATUREMAP()
2 $pop \leftarrow \emptyset$
3 /* Generate Initial Population */
4 **while** $i \leq P_s$ **do**
5 $\quad t_c \leftarrow$ GENERATEINDIVIDUAL()
6 $\quad f \leftarrow$ EXECUTEINDIVIDUAL(t_c, M, S)
7 $\quad$ PLACEINDIVIDUALMAP(F_m, f, t_c)
8 $\quad pop \leftarrow pop \cup \{t_c\}$
9 **end**
10 /* Evolve Individuals */
11 **while** $i \leq N$ **do**
12 $\quad t_c \leftarrow$ SELECTINDIVIDUAL(pop)
13 $\quad \hat{t}_c \leftarrow$ MUTATEINDIVIDUAL(t_c)
14 $\quad f \leftarrow$ EXECUTEINDIVIDUAL($\hat{t}_c$, M, S)
15 $\quad$ PLACEINDIVIDUALMAP(F_m, f, $\hat{t}_c$)
16 **end**
17 **return** F_m

2021). Such features have been used in previous work and have been shown to be effective at characterizing the search space of road generators (Zohdinasab et al. 2021). Characterizing a test case based on its structural features, i.e., only based on the properties of the road, allows us to identify unique failure scenarios, i.e., failure scenarios with distinctive road properties.

During test generation, the test cases are distributed in the map according to their features. The *value* of each cell is influenced by the behavior of M when driving on the roads pertaining to a cell. The minimum *lateral distance* recorded by the simulator is used by DeepHyperion as a *fitness* of the generated test case. The lateral distance is the opposite of the lateral position, i.e., it has the highest value when the vehicle drives at the center of the lane, and it decreases as the vehicle approaches the roadside. In particular, it is negative when the model misbehaves (i.e., the vehicle goes out of bound). In Fig. 4 the two dashed-encircled cells point out two failure cells for M (i.e., cells containing roads with negative fitness).

Algorithm 1 shows the pseudocode of the DeepHyperion algorithm. It takes as input the driving model under test M, the simulator instance S and two hyperparameters, i.e., the population size P_s and the number of iterations N the search is allowed to run, i.e., the budget of the algorithm. The algorithm starts by initializing an empty feature map and population (Lines 1–2). Then, the *while* loop at Lines 4–9 fills the initial population by randomly generating an individual (Line 5) and executing it to collect its fitness value f (Line 6).

The assignment to the feature map (Line 7) is done by the procedure PLACEINDIVIDUALMAP based on the feature values of the individual t_c (to determine the coordinates of the target cell) and its fitness value. If the target cell is empty, the individual is placed in the cell. If the cell is non-empty (i.e., another test case was already generated for that cell), a *local competition* based on the value of the fitness takes place. If the fitness of the individual in the cell is greater than the fitness of the candidate individual, the individual in the cell gets replaced with the candidate individual. Otherwise, no replacement is carried out, which also holds if the individual in the cell already has a negative fitness. The selection function ensures that the search space of the features is explored at large, while the local competition on the individual cells keeps only the lowest performing individuals (i.e., potential misbheaviours) at the end of the generation in order to guide the search towards misbehaviors with unique feature values.

The *while* loop at Lines 11–16 evolves the initial population of individuals. First, an individual is selected (Line 12) and mutated (Line 13), i.e., the control points of the road are changed in order to form a new individual $\hat{t}_c$ with different features. Such individual is then executed (Line 14) and placed in the map (Line 15). The algorithm terminates after a number N of iterations (Line 16).

Algorithm 1 returns a feature map with a single individual for each cell, i.e., the one with the lowest fitness (Line 17). In order to further explore the search space, we run DeepHyperion multiple times for each digital sibling to generate multiple feature maps. Then, we combine such maps by considering the *bounds* of each feature map axis in all the runs (i.e., minimum and maximum value), and place each generated individual in the combined map, whose bounds are the lowest (resp. highest) bound values across maps. In this way, there are potentially multiple individuals in each cell, and the value of a cell represents the metric of interest averaged over all individuals in that cell (see FM_{DS_1} and FM_{DS_2} in Fig. 1). For instance, considering the failure probability, the value of a cell represents the number of times the model under test failed over the number of all individuals in the cell (a failure occurs when the fitness of an individual is negative).

3.4 Migration and Union

The test generation step produces two feature maps FM_{DS_1} and FM_{DS_2}, for DS_1 and DS_2, respectively (in general, N feature maps, i.e., FM_{DS_1}, ..., FM_{DS_N}). The next step of our approach (i.e., step ❸, see Fig. 1) consists of *migrating* the test cases in FM_{DS_1} to DS_2 (producing $\overline{FM}_{DS_1}$) and viceversa (producing $\overline{FM}_{DS_2}$). In general, migrating the test cases in FM_{DS_i} (with $i = 1, \dots, N$) to DS_j (with $j \neq i$), would produce $\overline{FM}_{DS_{ij}}$. For instance, if $N = 3$, migrating the test cases in FM_{DS_2} to the other siblings, would produce $\overline{FM}_{DS_{21}}$ when migrating to DS_1, and $\overline{FM}_{DS_{23}}$ when migrating to DS_3. Such operation consists of instantiating the abstract (control point based) road representation of the test case being migrated, such that it respects the dimensionality constraints, and it can be supplied as input to the target simulator.

After migration, for both DS_1 and DS_2 (in general, DS_1, ..., DS_N), we consider the *union* of their maps. We consider the bounds of each feature in the two maps, and we place the respective test cases in a new unified map according to their coordinates, producing the map FM_{U_1} for DS_1 (i.e., $FM_{DS_1} + \overline{FM}_{DS_2}$) and the map FM_{U_2} for DS_2 (i.e., $FM_{DS_2} + \overline{FM}_{DS_1}$). In general, $FM_{U_i} = FM_{DS_i} + \sum_{j \neq i} \overline{FM}_{DS_{ji}}$. For instance, if $N = 3$, $FM_{U_2} = FM_{DS_2} + (\overline{FM}_{DS_{12}} + \overline{FM}_{DS_{32}})$. Hence, the two maps, or N maps in general, contain the same tests that fill the same cells at the same coordinates.

The value of each cell in the union maps FM_{U_1}, FM_{U_2} is recomputed from the individuals assigned to them. For the failure probability, if a given cell in FM_{DS_1} has n_1/N_1 failing individuals, while the corresponding cell in $\overline{FM}_{DS_2}$ has n_2/N_2 failing individuals, the failure probability value of the cell in the union map FM_{U_1} will be $(n_1+n_2)/(N_1+N_2)$. In general, for a given cell in FM_{U_i}, the failure probability is computed as $(n_1+\dots+n_i+\dots n_N)/(N_1+\dots+N_i+\dots+N_N)$. When a quality of driving metric is computed instead of a failure probability, the union map contains the average of the respective quality of driving metrics:

$qm = (qm_1 + qm_2)/2$, where qm_1, qm_2 are the quality of driving metrics found in the same cell in the two feature maps being united (FM_{DS_1}, $\overline{FM}_{DS_2}$, or FM_{S_2}, $\overline{FM}_{S_1}$), while qm is the resulting quality of driving metric, in the union map (FM_{U_1} or FM_{U_2}). In general, for a given cell in FM_{U_i}, the quality metric is computed as $(qm_1 + \cdots + qm_i + \ldots qm_N)/N$.

3.5 Merge

The final step of the approach (i.e., step ❹ in Fig. 1) requires to *merge* the two union maps FM_{U_1} and FM_{U_2} into FM_{DSS} (in general, N union maps $FM_{U_1}, \ldots, FM_{U_N}$). The objective of the merge operation is to combine the testing output of the two digital siblings. Since we aim to use the digital siblings to approximate the behavior of M on DT and predict its failures, the merge operator privileges *agreements* between the maps of the two digital siblings, i.e., only cells in the maps that have a hot color (e.g., a high failure probability) will produce a hot color in the merged cell. Indeed, such tests are likely to represent simulator-independent misbehaviors of the model under test, which are critical for the safety of the system. Specifically, if the failure probability of FM_{U_1} is $fp_1 = n_1/N_1$ and that of FM_{U_2} is $fp_2 = n_2/N_2$, in the merged map the failure probability will be the product, $fp = fp_1 \times fp_2$ (in general, the failure probability of a given cell in DSS would be $fp = fp_1 \times \cdots \times fp_i \times \cdots \times fp_N$). When a quality of driving (resp. lack of quality of driving) metric is computed instead of a failure probability, the merged map will conservatively contain the maximum (resp. minimum) of the respective quality of driving metrics. In particular, $qm = \max\{qm_1, qm_2\}$ (resp. $qm = \min\{qm_1, qm_2\}$), where qm_1, qm_2 are the quality of driving metrics found in the same cell in FM_{U_1} and FM_{U_2} respectively, while qm is the resulting quality of driving metric in the merged map. In general, the quality metric of a given cell in DSS would be $qm = \max\{qm_1, \ldots, qm_i, \ldots, qm_N\}$, and the lack of quality of driving of a given cell would be $qm = \min\{qm_1, \ldots, qm_i, \ldots, qm_N\}$. By giving priority to failures (resp. quality of driving degradations) that occur in both siblings and are hence very likely to be relevant for the target platform, this choice better accommodates the limited testing budget available for production/field testing (BGR Media 2018; Borg et al. 2021; Cerf 2018; May 2019; Stocco et al. 2022).

3.6 Evaluation Scenario

While our approach assumes that DT is not available in practice, to evaluate whether the DSS can approximate the behavior of M and predict its failures when executed on DT, we migrate all the tests in the digital siblings feature map (i.e., FM_{DSS}) to an actual DT, which is used to obtain the ground truth map FM_{DT} (see "Evaluation Scenario" in Fig. 1 (bottom)). The two maps being compared contain the same tests in the same cells, but the values of the cells might differ, depending on the behavior of M in the different simulators. Thus, we analyze and compare the two feature maps FM_{DSS} and FM_{DT}, to assess the capability of DSS at predicting the failures of the model when executed on DT.

4 Case Study

The goal of the empirical study is to evaluate whether two digital siblings (DSS) can better approximate the *behavior* of a driving model and predict its failures on a digital twin (DT), w.r.t. using only one general-purpose simulator (GPSim). We rely on DT only to evaluate the benefits of our multi-simulator approach, as a proxy for the behaviors of the AV in the real world, since DT is often unavailable in practice. In our empirical study, we focus on testing a lane-keeping DNN model by generating road scenarios. To this aim, we consider the following research questions:

RQ$_1$ (Offline Evaluation) *How do the offline prediction errors by the DSS compare to those of the DT?*

We first test our hypothesis at the model-level. For all simulators, we compute the errors between the model predictions and each autopilot ground truth labels on a stationary driving images dataset. We compare the error distributions of each individual simulator with the DT, as well as their combination as digital siblings.

With RQ$_1$ we aim to assess whether a correlation between the offline predictions exists at the model-level, which can be useful for developers to gain trust about their DNN model prediction accuracy, prior to running system-level tests.

RQ$_2$ (Failure Probability) *How does the failure probability of the DSS compare to that of the DT?*

In RQ$_2$ we test the model at the system-level, specifically the hypothesis that combining the failure probabilities of the two digital siblings provides a better predictor of the ground truth failure probability of the model executed on DT w.r.t. using a single simulator. A positive answer to RQ$_2$ would imply that a multi-simulator approach can predict, and possibly anticipate, the failures of the DNN-based lane-keeping model on DT, which are expected to be accurate proxies of the AV real-world failures.

RQ$_3$ (Quality of Driving) *How does the quality of driving of the DSS compare to the failure probability of the DT?*

By considering only the failure probability, we might overlook the correlation between real failures on DT and near-failures on DSS—test cases in which the model exhibits a degraded driving quality without necessarily going off-road. Thus, with RQ$_3$, we also assess whether finer-grained driving quality metrics can predict the ground truth failure probability of the lane-keeping model on DT.

4.1 Test Object and Simulators

4.1.1 Study Object

We considered three self-driving architectures, i.e., Dave-2 Bojarski et al. (2016); Chauffeur (2016) and Epoch (2016). Such architectures were used in previous studies on AV testing in the literature (Stocco et al. 2022, 2020; Tang et al. 2022; Jahangirova et al. 2021; Stocco et al. 2022; Stocco and Tonella 2020, 2021; Zohdinasab et al. 2021; Panichella et al. 2021; Gambi et al. 2022; Biagiola et al. 2023), and the respective models feature different number of parameters. The Dave-2 model has 2.8M parameters, Chauffeur has 100k parameters while Epoch has 26M parameters (we used a reduced version of the Epoch model to reduce training and inference time (Stocco et al. 2022)).

Architecturally, Dave-2 consists of five convolutional layers, followed by three fully-connected layers Bojarski et al. (2016). Chauffeur has six convolutional layers each followed by a dropout and a max pooling layer (except the last one) (Chauffeur 2016). Epoch has three convolutional layers and one fully-connected layer, which makes up for most of the parameter count of the model (Epoch 2016).

4.1.2 Digital Siblings (DSS)

We implemented and investigated the effectiveness of DSS using the simulators (BeamNG 2022) and (Udacity 2017). We chose them as digital siblings because: (1) they support training and testing of a DNN that performs lane-keeping, including Dave-2, Chauffeur and Epoch; (2) they are often used as simulator platforms for AV testing, as highlighted by a recent survey on autonomous driving testing (Tang et al. 2022); (3) they are potentially complementary because they are developed with different technologies/game engines, and they are characterized by different physics implementations (e.g., rigid vs soft-body dynamics); (4) they are publicly available under open-source or academic-oriented licenses, hence customizable.

BeamNG (2022) is a framework specialized in autonomous driving developed by BeamNG GmbH. The framework is released under an academic-oriented license, and it has been downloaded 5.5*k* times as of January 2023. From a technical standpoint, BeamNG features a *soft-body dynamics* simulation based on a spring-mass model. Such a model is composed of nodes (mass points) that are connected by beams (springs), i.e., weightless elements that allow accurate vehicle deformation and other aerodynamic properties Gambi et al. (2019).

Udacity (2017) is developed with Unity3d (2021), a popular cross-platform game engine. The project has been publicly released in 2016 by the for-profit educational organization Udacity, to allow people from all over the world to access some of their technology and to contribute to an open-source self-driving car project. As of January 2023, the simulator has 3.7*k* stars on GitHub. From a technical standpoint, Udacity is based on the Nvidia PhysX engine (Nvidia PhysX 2022), featuring discrete and continuous collision detection, ray-casting, and *rigid-body dynamics* simulation.

4.1.3 Digital Twin (DT)

We use the Donkey Car™ open-source framework (Donkey Car 2021) as digital twin for our study. This platform has been used for AV testing research with physical self-driving cars in physical environments (Stocco et al. 2022; Viitala et al. 2020; Zhou et al. 2021). The framework includes open hardware to build 1:16 scale radio-controlled cars with self-driving capabilities, a Python framework for training and testing DNN models with lane-keeping functionalities using supervised or reinforcement learning, and a simulator in which the real-world Donkey Car is faithfully modeled. This was assessed by a recent work (Stocco et al. 2022) reporting that, for three lane-keeping models, the steering angle distribution of the AV model driving in the real-world environment is statistically indistinguishable from the steering angle distribution of the AV model driving in the digital twin.

In the rest of the section, we refer to BeamNG as DS_1, Udacity as DS_2, the combined digital siblings as DSS, and DonkeyCar as DT.

4.2 Procedure

4.2.1 CycleGAN Models

Data Collection We collected 15*k* simulated images, 5*k* for DS_1 and DS_2 by running the autopilots on a set of randomly generated roads. Moreover, we collected 5*k* real-world images (Stocco et al. 2022) by manually driving the physical twin of the DT on a physical road track in our lab.

Training We trained three CycleGAN models, one for each simulator, with the obtained training sets (5*k* virtual images and 5*k* real-world images). Each model was trained for 60 epochs using the default hyper-parameters of the original paper (Zhu et al. 2017). We saved a checkpoint model every 5 epochs, and we ultimately chose the one that achieved the best neural translations (in terms of visual quality) using a test set of $\approx 8k$ simulated images for each simulator, representing a test road driven from beginning to the end (Stocco et al. 2022). While a quantitative assessment of the output of CycleGAN is still a major challenge (Borji 2019; Lambertenghi and Stocco 2024) and out of the scope of this paper, the driving capability of the lane-keeping model, as the experimental evaluation shows, represents an implicit validation of the CycleGAN model's ability to retain all essential features needed for an accurate steering angle prediction.

4.2.2 Driving Models

Data Collection For all simulators (i.e., DS_1, DS_2 and DT), we collected a training set by running the autopilots on a set of randomly generated roads (this set is different from the one used to train the CycleGAN). To ensure having non-trivial driving scenarios and appropriate labels for challenging curves, the maximum angle of a curve was set to be less than or equal to 270°. In particular, for our training set, we generated 25 roads with 8 control points (Zohdinasab et al. 2021). To collect a balanced dataset where left and right curves are equally represented, each road was driven by the autopilot in both directions, i.e., from the start point to the end point and from the end point to the start point. The autopilot drove successfully the totality of the roads on all simulators; our training set comprises $\approx 70k$ images, equally distributed across the simulators.

Training For each self-driving architecture we trained two models, one by using the plain simulated images (M_S) and the other by translating the images of each simulator into *pseudo*-real images (M_R) using the respective CycleGAN generator.

We followed the guidelines by Bojarski et al. Bojarski et al. (2016) to train AV autopilots. We used custom hyperparameters for each self-driving architecture, and the Adam optimizer (Kingma and Ba 2014) to minimize the mean squared error (MSE) between the predicted steering angles and the ground truth value. For all models, we set a learning rate of 10^{-4} and a batch size of 128. We used 50 epochs for Dave-2 and Chauffeur (only for the M_R model) and 500 epochs for Epoch and the M_S model of Chauffeur. We used an early stopping of 10 epochs for the models where the number of training epochs was 50 and an early stopping of 20 epochs otherwise.

We evaluated the performance of the trained lane-keeping models on DT, as it is the target simulator we want to approximate using the digital siblings. We collected a labeled dataset by running the autopilot on DT on 25 randomly generated roads each with 8 control points and

Table 1 Offline and online performance on the test set of the lane-keeping models on DT

	M_S		M_R	
	MSE	Success rate	MSE	Success rate
Dave-2 Bojarski et al. (2016)	0.08	0.84	0.07	0.96
Chauffeur (2016)	0.07	0.72	0.07	0.92
Epoch (2016)	0.09	0.52	0.07	0.96
Avg	0.08	0.69	0.07	0.95

a maximum angle of 270°, i.e., the same road parameters as the training set. We computed the mean squared error (MSE) between the steering angle prediction of the model on each image and the steering angle of the autopilot. Table 1 shows the MSE of all models on the first and third columns; on average, the MSE is low for both the models trained using simulated images (i.e., M_S), and the models trained using real images (i.e., M_R). We also measured the success rate of each model by driving it on the 25 randomly generated roads, and counting the number of times the model was able to arrive at the end of the road without going out of bound. Overall, each model is able to successfully complete the majority of the generated roads. Most notably, M_R models are able to complete more than 90% of the test set roads.

4.2.3 Offline Evaluation

We collected a labeled dataset for offline evaluation by generating 20 roads (i.e., 10 roads driven in both directions) with the same parameters as the training set. The images collected for the *offline* evaluation dataset amount to ≈26*k*, considering all simulators.

4.2.4 Test Generation

After training M_S and M_R for each self-driving architecture, we executed DeepHyperion *twice* to generate tests using the two digital siblings DS_1 and DS_2. We chose a population size of 20 individuals and a number of search iterations respectively equal to 150 for M_S and 100 for M_R, as we observed from preliminary experiments that this choice of hyperparameters allows an extensive coverage of the feature maps. For both M_S and M_R and each digital sibling in each self-driving architecture, we repeated test generation five times to diversify the exploration of the search space and to collect multiple test cases for each cell in the feature maps. Overall, across all runs and driving models, DeepHyperion generated 10,260 tests for both siblings.

Concerning the simulations, for all simulators, we set the maximum speed for the vehicle to 30 km/h (Zohdinasab et al. 2021). When testing M_R in a given simulator, we engineered the testing pipeline to load the appropriate sim2real CycleGAN generator to translate the simulated image generated by BeamNG/Udacity into pseudo-real images *in real-time during driving*. For each executed test case, we collected the lateral position of the vehicle for each simulation step as well as its lateral distance. The former determines the quality of driving of the model (Jahangirova et al. 2021), while the latter is the fitness of the test case.

4.2.5 Migration and Union

For the initial (FM_{DS_1}, FM_{DS_2}) and for the union (FM_{U_1}, FM_{U_2}) feature maps, we compute the failure probability as the number of tests with a negative fitness divided by the total number of tests in the respective cell. To evaluate the quality of driving, we adopted the maximum lateral position (i.e., the distance between the center of the vehicle and the center of the lane (Stocco and Tonella 2020)) experienced during the test case execution. Previous work showed that such metric is effective at characterizing the degradation in the quality of autonomous driving (Jahangirova et al. 2021), since the lower the value of such metric, the higher is the quality of driving (thus, it actually measures *lack* of quality of driving). When considering the quality of driving, the value of each cell in a feature map represents the average of the maximum lateral positions of each test case in that cell. Furthermore, we normalized the maximum lateral position values in the interval [0, 1] before taking the union.

4.2.6 Merge

Merging the maps of the two digital siblings requires a different treatment for failure probability and quality of driving. Regarding the failure probability, the merge operator that ensures a conservative aggregation of two values is the *product*. Regarding the lack of quality of driving, the conservative merge operator is the *minimum*, since the quantities to merge are not probabilities. In fact, by taking the minimum we get a high lack of driving quality only when both simulators exhibit high values for such a metric.

4.3 Metrics

4.3.1 RQ$_1$ (Offline Evaluation)

We computed the prediction errors given by the difference between the predictions of the model (M_R) on images of the offline evaluation dataset (see Section 4.2), and the corresponding ground truth labels given by the autopilot. We binned the prediction errors of the model on each simulator and built the respective *probability density* (i.e., the number of errors in each bin is divided by the total number of prediction errors) such that different distributions could be compared.

Then, we computed the *distance* between each digital sibling distribution, as well as their combination, and the DT using the *Wasserstein* distance (Arjovsky et al. 2017) (also known as the *earth mover's distance*). Given two one-dimensional distributions A and B, the Wasserstein distance $W(A, B)$ is defined by the following formula (Ramdas et al. 2017):

$$W(A, B) = \int_{\mathbb{R}} |CDF_A(x) - CDF_B(x)| dx \tag{3}$$

where CDF is the *cumulative distribution function* of a distribution. In other words, the Wasserstein distance between two distributions is defined as the difference between the area formed by their cumulative distribution functions.

We assess whether the difference between two distributions is statistically significant using the Wilcoxon test (Conover 1999) applied to the density functions of the two error distributions to compute the p-value (with threshold $\alpha \leq 0.05$). We also perform power analysis (with statistical power $\beta \geq 0.8$) on the prediction errors to check whether a non-significant p-value is due to a low data sample size or to the difference being statistically insignificant.

4.3.2 RQ$_2$ (Failure Probability) and RQ$_3$ (Quality of Driving)

For RQ$_2$, we computed the pairwise *Pearson correlation* between maps along with the corresponding p-value. In particular, correlations are computed between each union feature map of each digital sibling (FM_{U_1}, FM_{U_2}) and the feature map of the DT (FM_{DT}), and between FM_{DSS} and FM_{DT}. For RQ$_3$, the setting is equivalent to that of the failure probability but considering quality of driving maps, comparing DS$_1$, DS$_2$ and DSS against the ground truth DT.

To evaluate the capabilities of the digital siblings (individually or jointly) to predict failures on DT, we computed the area under the curve Precision-Recall (AUC-PRC) at increasing thresholds, for both RQ$_2$ and RQ$_3$. This requires the discretization of failure probabilities into binary values (failure vs non-failure) for the ground truth (i.e., DT): we consider a cell in the DT feature map to be a failure cell if the associated failure probability is > 0.0. AUC-PRC is more informative than the AUC-ROC metric (i.e., the area under of the curve of the Receiver Operating Characteristics) when dealing with imbalanced (Saito and Rehmsmeier 2015) datasets, which is the case of our study (the number of failures in the feature maps is lower than the number of non-failures with an average 10 to 20% ratio).

4.4 Results

4.4.1 Offline Evaluation (RQ_1)

Table 2 reports the results for our first research question. The first column shows the simulators being compared. Columns 2–5 report the Wasserstein distance between the prediction error densities of the corresponding simulators, and the *p*-value concerning the statistical significance of the differences between the two densities, for M_S and M_R.

For M_S (Columns 3–4), our results show that, for Dave-2, the distance between the steering angle errors obtained for the combined digital siblings DSS and the errors obtained for DT is lower than the distance of DS_1 (0.03776 vs 0.046) and higher than the distance of DS_2 (0.02648). The distribution of the steering angle errors of DS_2 is statistically different from the errors of DT (i.e., *p*-value $0.02 < 0.05$), while the distribution of the steering angle errors of DSS is statistically indistinguishable from the errors of DT (i.e., *p*-value $0.053 > 0.05$ and power > 0.8). This behavior is also consistent for Epoch, with the exception that the distribution of the prediction errors for DS_2 is statistically indistinguishable from that of DT. However, the distance between DSS and DT is lower than the distance of DS_1 from DT, with a statistically indistinguishable distribution of prediction errors w.r.t. DT. For Chauffeur, the combined digital siblings DSS have the only distribution of errors that is equivalent to that of DT, and its distance to it is the lowest considering the individual digital siblings.

Regarding M_R (Columns 5–6), our results show that, for Dave-2, the distance between the steering angle errors obtained for the combined digital siblings DSS and the errors obtained for DT is *2.8 times lower* than the distance of each simulator taken individually (as a percentage,

Table 2 Results for RQ_1

		Offline evaluation (RQ_1)			
		M_S		M_R	
		distance	p-value	distance	p-value
Dave-2 Bojarski et al. (2016)	DS_1 vs DT	0.04669	0.101	0.03250	0.011
	DS_2 vs DT	0.02648	0.020	0.02187	0.078
	DSS vs DT	**0.03776**	0.053†	**0.00951**	0.088†
Chauffeur (2016)	DS_1 vs DT	0.03989	0.023	0.04625	0.011
	DS_2 vs DT	0.02641	0.047	0.02145	0.078†
	DSS vs DT	**0.01208**	0.394†	**0.01843**	0.334†
Epoch (2016)	DS_1 vs DT	0.06030	0.011	0.03374	0.016
	DS_2 vs DT	**0.01634**	0.078†	0.02318	0.078†
	DSS vs DT	0.02726	0.053†	**0.00989**	0.256†

† *power > 0.8*
Bold-faced values indicate the best approach

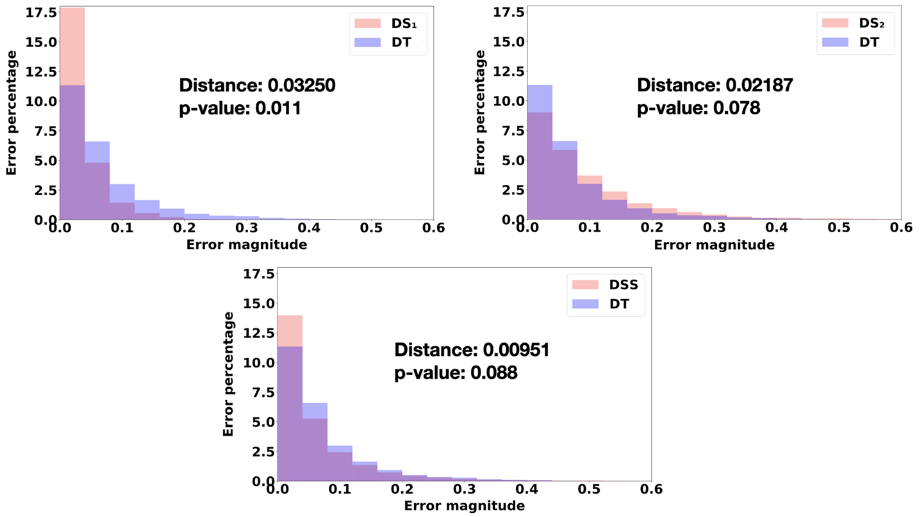

Fig. 5 Distributions of prediction errors of Dave-2 M_R in the two digital siblings, i.e., DS_1 and DS_2, their combination (DSS) and DT. Best viewed in color

the distance of DSS is respectively 70% and 56% smaller than the distance of the two individual siblings, DS_1, DS_2). The statistical test confirms that the error distributions of DSS and DT are statistically indistinguishable (p-value > 0.05 and power > 0.8), which is not the case for the error distributions of DS_1 (p-value < 0.05). Likewise, for all the other self-driving architectures, the digital siblings DSS have the lowest distance to DT w.r.t. the individual siblings and their distribution is always statistically indistinguishable from that of DT.

Figure 5 offers a visual explanation of these scores for the Dave-2 model.[1] The subplots compare the steering angle error distributions, respectively, of DS_1, DS_2 and DSS (shown in light red) with that of DT (shown in light blue). The x-axis of each subplot represents the magnitude of the prediction errors of the model M_R w.r.t. the predictions of the autopilot, while the y-axis indicates their percentage for each bin.

From the plots we can see that, overall, at the model-level, M_R makes prediction errors with small magnitudes on DS_1, DS_2 and DSS (i.e., most of the errors are between 0.0 and 0.3). On the digital sibling DS_1 (i.e., BeamNG), M_R has a high agreement with the autopilot, as most errors have a low magnitude. It has numerous small errors (< 0.2), while it has only a negligible portion of the distribution being above 0.2. The agreement with DT is low as M_R *under-approximates* the true error distribution on DT: M_R on DT has fewer errors with low magnitude and has a longer tail of errors greater than 0.2 (even greater than 0.3 in some cases). Differently, on the digital sibling DS_2 (i.e., Udacity), the error distribution has a longer tail than that on DT. Indeed, M_R executed on DS_2 *over-approximates* the errors it would have on DT, as the errors observed on DS_2 have higher magnitude than those observed on DT.

The error distribution of the model on DSS shows why it is appropriate to combine the outcome of two simulators. At the model-level, DSS better approximates the true error

[1] We report the plots for the other lane-keeping models in our replication package Replication package (2023).

distribution of the model on DT, by providing an intermediate error between DS_1 and DS_2 for both M_S and M_R.

> **$\mathbf{RQ}_1$**: Overall, at the model-level, the digital siblings produce a steering angle error distribution that is statistically indistinguishable from the true steering angle error distribution of the model on the digital twin. Considering all the models, in 5 out of 6 cases, the digital siblings are better at approximating the distribution of prediction errors of the digital twin than each individual sibling.

4.4.2 Failure Probability (RQ_2)

Table 3 shows the Pearson correlation (r), the p-value, and the AUC-PRC for the comparison between DS_1, DS_2, DSS and DT, respectively. The analysis is reported separately for M_S (Columns 3–5) and M_R (Columns 6–8).

Concerning M_S—i.e., the model driving with simulated driving scenes— the failure probabilities for Dave-2 have a high positive correlation with the true failure probability of DT ((Column 3). All such correlations are statistically significant for DSS, as well as for each individual sibling DS_1 and DS_2 (p-values < 0.05, see Column 4). Likewise, the correlations are high and statistically significant for the other lane-keeping models (Epoch features slightly lower correlations).

However, for Dave-2 the correlation of DSS is 9% higher than the best individual correlation (i.e., DS_1) and 21% higher than the worst individual correlation (i.e., DS_2). In terms of failure prediction, DSS have the highest AUC-PRC value, 4% higher than DS_1 and 33% higher than DS_2.

This also happens with Epoch, where the correlation of DSS is slightly higher than that of the best sibling DS_1 (i.e., 0.571 vs 0.561) and 33% higher than that of the worst sibling DS_2. Regarding failure prediction on DT, DSS are 3% better than the best sibling. In the case of Chauffeur, DS_1 has the best results both in terms of correlation and failure prediction.

Table 3 Results for RQ_2

		Failure probability (RQ_2)					
		M_S			M_R		
		r	p-value	AUC-PRC	r	p-value	AUC-PRC
Dave-2 Bojarski et al. (2016)	DS_1 vs DT	0.650	10^{-11}	0.654	0.391	10^{-4}	**0.403**
	DS_2 vs DT	0.583	10^{-8}	0.512	0.377	10^{-4}	0.306
	DSS vs DT	**0.710**	10^{-13}	**0.684**	**0.457**	10^{-5}	0.398
Chauffeur (2016)	DS_1 vs DT	**0.733**	10^{-16}	**0.774**	0.417	10^{-4}	0.481
	DS_2 vs DT	0.588	10^{-10}	0.715	0.337	10^{-3}	0.300
	DSS vs DT	0.700	10^{-14}	0.742	**0.422**	10^{-4}	**0.496**
Epoch (2016)	DS_1 vs DT	0.561	10^{-8}	0.599	0.469	10^{-5}	0.586
	DS_2 vs DT	0.428	10^{-5}	0.604	**0.521**	10^{-7}	0.565
	DSS vs DT	**0.571**	10^{-8}	**0.622**	0.450	10^{-5}	**0.641**

Bold-faced values indicate the best approach

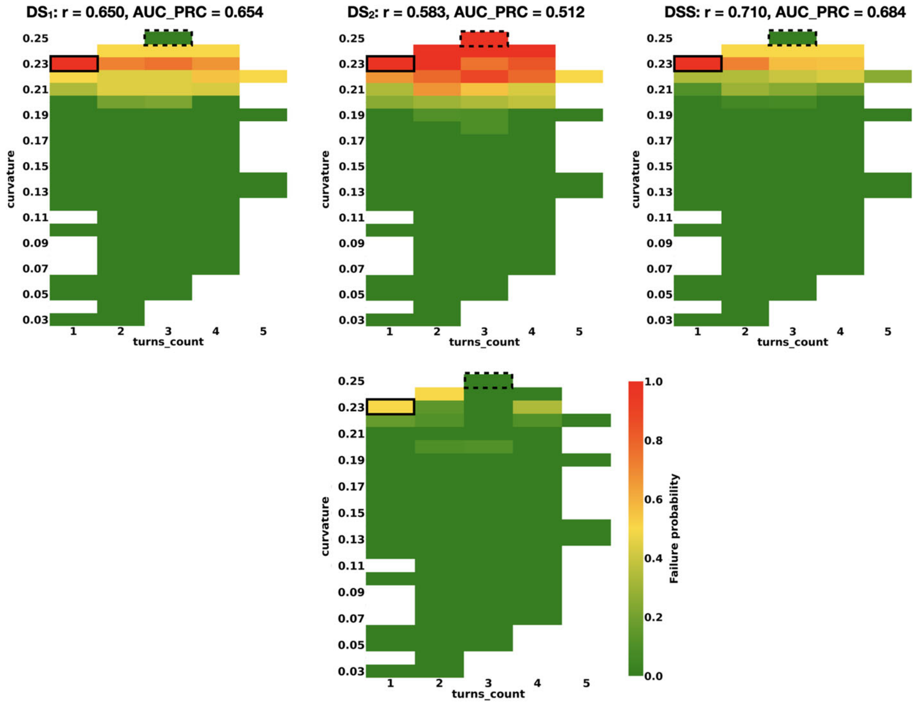

Fig. 6 Feature maps representing the failure probability of Dave-2 M_S on the two digital siblings, DS_1 and DS_2, their combination (DSS) and on DT. Solid line cells represent a true failure predicted by DSS while dashed line cells represent a false positive of DS_2. Best viewed in color

However, DSS are better than the worst of the two siblings DS_2 both in terms of correlation and failure prediction.

Figure 6 shows the feature maps related to M_S of Dave-2.[2] The first three feature maps represent the failure probability of DS_1, DS_2 and DSS, respectively. The last feature map represents the ground truth failure probability of DT. The color of each cell ranges from green (i.e., non-failure, or failure probability = 0) to red (i.e., failure probability = 1). Let us analyze a *false positive* case. The test cases at coordinates (3, 0.25), whose corresponding cells are highlighted with a dashed line, represent road tracks having three curves and a maximum curvature of 0.25. In DT, this cell is green, i.e., all test cases for M_S driving on DT succeed. On the other hand, M_S has contrasting behaviors when the same test cases are executed on DS_1 or DS_2. These test cases did not exhibit any failure in DS_1, whereas they did trigger failures in DS_2. This disagreement is canceled out when combining the two digital siblings with the product operator and the cell is green in the DSS map. As such, digital siblings are conservative w.r.t. failures, as a failure is reported only when both digital siblings are in agreement. This can be noticed for test cases at coordinates (1, 0.23), which represent road tracks having one curve with a maximum curvature of 0.23—an instance of a *true positive* case (the corresponding cells in each map are highlighted with a solid line). Both DS_1 and DS_2 have a failure probability of 1 and, as a consequence, the DSS map also does. On DT, M_S has also a high failure probability (0.5), which confirms the high effectiveness of the DSS framework at approximating the true failure probability of DT.

[2] We report the plots for the other lane-keeping models in our replication package Replication package (2023).

Concerning the failure probability for M_R—i.e., the model driving with pseudo-real driving scenes, for Dave-2 and Chauffeur, DSS are better than each individual sibling in terms of correlation with DT. For Dave-2, DS_1 better predicts the failures of DT, while for Chauffeur, the digital siblings are better than each individual sibling. Interestingly, for Epoch, DS_2 better correlates with DT but the AUC-PRC value of DSS is the higher than the individual siblings.

> **RQ**$_2$: At the system-level, in four cases out of six, the failure probability of the digital siblings better correlates with the true failure probability of the digital twin w.r.t. each individual sibling. In four cases out of six, the failures obtained on the digital siblings are a better predictor of the ground truth failures experienced on the digital twin.

4.4.3 Quality of Driving (RQ$_3$)

Table 4 shows the Pearson correlation (r), the p-value, and the AUC-PRC for the comparison between DS_1, DS_2, DSS and DT, respectively. The comparison considers the correlation between the quality of driving metric experienced in DS_1, DS_2, DSS and the failure probability of the model on DT, as well as the prediction of failures from the quality of driving metric. The analysis is reported separately for both M_S (Columns 3–5) and M_R (Columns 6–8) models.

For M_S, the correlation between DSS and DT is lower than the best individual correlation for all the lane-keeping models (0.553 of DSS vs 0.621 of DS_1 for Dave-2, 0.792 of DSS vs 0.798 of DS_1 for Chauffeur, and 0.491 of DSS vs 0.511 of DS_1 for Epoch). For Dave-2, the DSS correlation is 22% higher than the worst individual correlation (0.553 of DSS vs 0.429 of DS_2); percentages are similar for Chauffeur and Epoch. For AUC-PRC, DSS and DS_1 have the same predictive power both for Dave-2 and Chauffeur (i.e., respectively 0.659 and 0.940), while for Epoch the DSS prediction is slightly better than that of DS_1. Thus, DSS mitigate the risk of relying on the testing results of a low-quality GPSim (i.e., DS_2).

Concerning M_R, we observed a similar trend, i.e., the correlation of DS_1 with DT are higher than the correlations of DSS with DT, although DSS always have a better correlation than the worst of the two siblings, i.e., DS_2, for all lane-keeping models. The digital siblings DSS better predict the failures of DT for Dave-2 and are equivalent to DS_1 for Chauffeur. For Epoch, the best predictor of the failures of DT is DS_2, although the digital siblings are only 9% worse.

Figure 7 shows the four feature maps related to the quality of driving of the M_R Dave-2 model on the two digital siblings and the failure probability of M_R on DT.[3] We can observe that the feature map of DS_1 and the feature map of DSS are similar. As a consequence, the two correlations are similar (0.396 of DS_1 vs 0.379 of DSS). On the other hand, the feature map of DS_2 is quite different from the failure probability map of DT, which causes the correlation to be low (0.287). We can observe that all siblings are able to capture the failure of the DT at coordinates (1, 0.23) (see the corresponding cells highlighted with a solid line). On the other hand, the test cases at coordinates (4, 0.24) triggered failures only in DS_2, and DSS correctly predict that in DT such tests will not cause a failure.

[3] We report the plots for the other lane-keeping models in our replication package Replication package (2023).

Table 4 Results for RQ_3

		Quality of driving (RQ_3)					
		M_S			M_R		
		r	p-value	AUC-PRC	r	p-value	AUC-PRC
Dave-2 Bojarski et al. (2016)	DS_1 vs DT	**0.621**	10^{-10}	**0.659**	**0.396**	10^{-4}	0.513
	DS_2 vs DT	0.429	10^{-5}	0.496	0.287	10^{-3}	0.351
	DSS vs DT	0.553	10^{-8}	**0.659**	0.379	10^{-4}	**0.626**
Chauffeur (2016)	DS_1 vs DT	**0.798**	10^{-21}	**0.940**	**0.399**	10^{-4}	**0.460**
	DS_2 vs DT	0.625	10^{-11}	0.791	0.260	0.025	0.359
	DSS vs DT	0.792	10^{-21}	**0.940**	0.382	10^{-4}	**0.460**
Epoch (2016)	DS_1 vs DT	**0.511**	10^{-7}	0.592	**0.554**	10^{-8}	0.608
	DS_2 vs DT	0.355	10^{-4}	0.541	0.389	10^{-3}	**0.715**
	DSS vs DT	0.491	10^{-6}	**0.594**	0.529	10^{-7}	0.651

Bold-faced values indicate the best approach

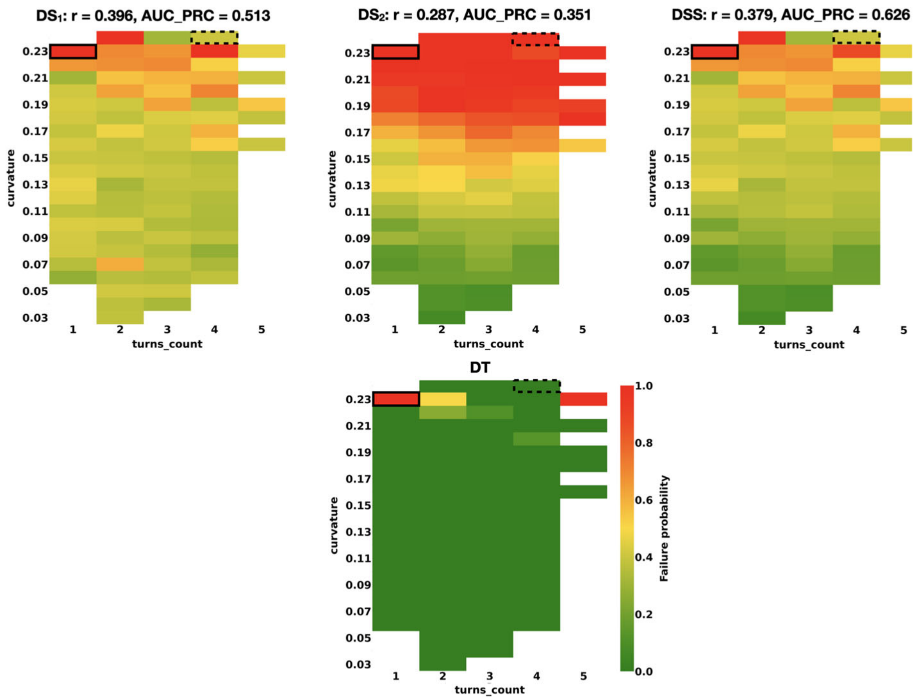

Fig. 7 Feature maps representing the quality of driving of Dave-2 M_R (i.e., the maximum lateral position) on the two digital siblings, DS_1 and DS_2, their combination (DSS) and the failure probability on DT. Solid line cells represent a true failure predicted by DSS, while dashed line cells represent a false positive of DS_2. Best viewed in color

RQ$_3$: At the system-level, for most lane-keeping models, the quality of driving of the digital siblings correlates with the failure probability of the digital twin. This correlation is either equivalent to that of the best digital sibling or falls within the range of the two siblings. In five cases out of six, the quality of driving in the digital siblings has a failure prediction capability w.r.t. the digital twin, which is equal or higher than the best individual sibling. As a result, digital siblings reduce the risk associated with relying on the least reliable simulator.

5 Discussion

GPSims Complementarity When choosing candidate GPSims, our approach requires that the simulators exhibit some degree of complementarity (i.e., different physics engines), while still supporting the same encoding of test inputs. Therefore, the selected GPSims must meet the following conditions. Firstly, the simulators must be equipped with appropriate API interfaces that allow the instantiation of analogous test cases. In our context, both Udacity and BeamNG support a sequence of road points as input to instantiate the two-lane roads where the AV drives. Secondly, the simulators need to support communication with the DNN-based systems under test. In the case of a DNN-based lane-keeping AV, the simulators should be able to capture images from the vehicle's on-board camera and execute throttle steering commands to drive the vehicle. Finally, the selected simulators should implement different physics engines. Specifically, Udacity implements soft-body dynamics, while BeamNG uses a rigid-body dynamics engine.

The worst case occurs when the two siblings disagree and the over-approximating sibling (e.g., predicting a failure) is not compensated by the under-approximating sibling (see Fig. 6). In most cases, we empirically observed that by predicting a failure only when there is agreement, the digital siblings are equivalent to the best of the two siblings (see RQ$_3$). However, for the Epoch model, when considering the failure probabilities of the M_R model, the correlation of the digital siblings is slightly worse than the worst sibling, i.e., DS$_1$ (specifically, 0.450 of DSS vs 0.469 of DS$_2$). Despite the lowest correlation, the digital siblings have the highest capabilities of detecting the failures of DT.

Simulated and Pseudo-real Models We experimented with both simulated (M_S) and real-world models (M_R) as such setting is representative of the current industrial testing practices described by the NHTSA U.S. Department of Transportation (2018). From the feature maps in Figs. 6 and 7, we can observe that the driving quality of M_S is superior w.r.t. M_R (the failure probabilities in the feature map of DT are higher), presumably because it is easier for a DNN to process plain artificial images from a simulator, rather than the images collected by a real-world camera during driving.

5.1 Threats to Validity

5.1.1 Internal Validity

We compared all simulators under identical parameter settings. One threat to internal validity concerns our custom implementation of DeepHyperion within the simulators. We mitigated this threat by faithfully replicating the code available in the replication package of the paper (DeepHyperion 2022). Another threat may be due to our own data collection phase and training of the lane-keeping models, which may exhibit many misbehaviors if trained inadequately. We mitigated this threat by training and fine-tuning a model which was able to drive on the majority of the training set roads consistently on all simulators.

5.1.2 External Validity

We considered only a limited number of DNN models and simulators, which poses a threat in terms of the generalizability of our results. We tried to mitigate this threat by choosing three popular real-world DNN models, which achieved competitive scores in the Udacity challenge (2020). Their diversity in terms of both size and architectural structure determines different driving behaviors and increases the generalizability of our results. We considered two open-source GPSims, and we chose DonkeyCar as DT, as it was used as a proxy for full size self-driving cars also in previous studies (Stocco et al. 2022, 2023; Verma et al. 2021; Viitala et al. 2020; Zhou et al. 2021). Generalizability to other GPSims or DTs would require further studies.

Our proposal focuses on testing the DNN-based lane-keeping component of an AV, by generating a large set of road scenarios. Although there are works in the literature that modify other environment objects such as weather conditions, pedestrian and other vehicles' dynamics (Ben Abdessalem et al. 2018; Haq 2022; Borg et al. 2021), we chose to generate road scenarios to test the lane-keeping behavior of the DNN in isolation, avoiding the interference of other tasks, such as obstacle and pedestrian avoidance. Further studies are needed to assess the generalizability of our multi-simulator approach to driving tasks different from lane-keeping. On this regard, feature maps are a flexible tool to encode different characteristics of a test case (e.g., the intensity of the rain or the number of vehicles in the driving scenario), by adding new dimensions for each new desired feature.

5.1.3 Construct Validity

Threats to construct validity may come from selecting inappropriate metrics to measure the agreement of the siblings with DT. To address this threat we assessed such agreement from two points of view, i.e., at the model-level (RQ_1), by measuring the distance between the two distributions under analysis and testing the statistical significance of the difference, and at the system-level, by measuring failure probability and quality of driving. Overall, our results show that the digital siblings are better at predicting the behavior of the lane-keeping model under test on DT.

6 Related Work

6.1 Digital Twins for AV Testing

Digital twins are used by researchers to reproduce real-world conditions within a simulation environment for testing purposes (Barosan et al. 2020; Yun and Park 2021; Kapteyn et al. 2020; San 2021; Almeaibed et al. 2021).

Yun and Park (2021) test an object recognition system using the GTA videogame. In particular, they exploit the realism of the game engine to collect data for training an object recognition system for both collision avoidance and lane-departure prevention. Barosan et al.

(2020) describe a digital twin for testing an autonomous truck. No testing was performed using the digital twin to assess the faithfulness of the simulator at reproducing real-world failures. Almeaibed et al. (2021), analyze the safety and security of digital twins and propose a general framework to address such issues during development. Kapteyn et al. (2020), propose a probabilistic graphical model to link the digital twin with its physical replica. The formal definition ensures that the calibration of the digital twin and its update with real-world data is principled and scalable. Similarly, San (2021) rely on the same mathematical tool to formalize the update of the digital twin with the goal of using it throughout the whole lifecycle of its physical replica, i.e., from the design to the operation phase. Veledar et al. (2019) propose a multi-metrics approach for security and safety validation for the design of a digital twin for autonomous driving.

Such works mostly focus on the design of the digital twin and its update during the development of the physical replica. Differently, in our paper we investigate testing transferability between digital siblings, i.e., a multi-simulator approach considering both simulated and pseudo-real images as input to the DNN.

6.2 Empirical Studies

Simulation platforms are often decoupled from the real world complexities (Afzal et al. 2021), which confirmed the need for real-world testing of cyber-physical systems. Our work is the first to propose the usage of a multi-simulator approach, called digital siblings, to mitigate the fidelity gap in the field of autonomous driving testing.

Concerning comparative studies across simulators, to the best of our knowledge, the only study that empirically compares the same AV on different simulation platforms is by Borg et al. (2021). The authors investigate the use of multiple GPSim for testing a pedestrian vision detection system. The study compares a large set of test scenarios on both PreScan Software (2022) and Pro-SiVIC Group (2021) and reports low agreement between testing results across the two simulation platforms. No assessment is performed of their correlation with a digital twin or a physical vehicle. In our paper, we take a step ahead, and we show how the (dis)agreements can be leveraged to mitigate the fidelity gap: by combining the predictions of two general-purpose simulators we successfully covered the gap with a DT for a scaled physical vehicle. In another work, (Amini et al. 2023) evaluates the degree of flakiness affecting two widely-used open-source AV simulators and five diverse test setups, showing that test flakiness in AV is a common issue and can significantly impact the test results obtained by randomized algorithms.

Other studies compare model-level vs system-level testing metrics within a simulation environment (Haq et al. 2021). In our empirical work, we focused on the difference between general-purpose and digital twin driving simulators. We use offline and online testing to measure the gap between single- and multi-simulator approaches at approximating a digital twin, a previously unexplored topic. Our proposition is also meant to prevent the flakiness occurring within a single simulation platform, by relying on an ensemble of simulators.

6.3 AV Testing Approaches

Most approaches use *model-level testing* (i.e., offline testing of single image predictions) to test DNN autopilots under corrupted images (Tian et al. 2018; Kong et al. 2020) or GAN-generated driving scenarios (Zhang et al. 2018), without however testing the self-driving software in its operational domain. In our work, we assess the effectiveness of our digital siblings with model-level testing in terms of prediction error distributions, but we also consider online testing at the system-level.

Another model-level testing approach is by Talwar et al. (2020). Their focus is to test the generalizability on real-world data of multiple object detection models trained on simulated images. On the other hand, we use an Image-to-Image translation architecture Zhu et al. (2017) to translate simulated images into real-world images both to evaluate the lane-keeping model offline and to test it online at the system-level.

Concerning *system-level testing* for AVs, researchers proposed techniques to generate scenarios that cause AVs to misbehave Stocco et al. (2020); Gambi et al. (2019); Stocco and Tonella (2021); Stocco et al. (2022); Moghadam et al. (2022); Zhang et al. (2018); Grewal et al. (2024); Kim et al. (2022); Zhong et al. (2021); Li et al. (2020); Jha et al. (2019); Cheng et al. (2023). Among the existing test generators, in this work we adopted DeepHyperion by Zohdinasab et al. (2021), a tool that uses illumination search to extensively cover a map of structural input features, which allowed us to easily group identical or equivalent failure conditions occurring in the same feature map cell. Haq (2022) use ML regressors as surrogate models to mimic the simulator's outcome.

These works only consider single-simulator approaches to testing. Their generalizability to a multi-simulator approach, such as the digital siblings proposed in this paper, or to cross-simulator testing, is overlooked in the existing literature.

7 Conclusions and Future Work

In this paper, we propose a multi-simulator approach named digital siblings, to improve simulation-based testing of the lane-keeping component of an autonomous vehicle. In our approach, we test the autonomous driving software by generating road scenarios in two general-purpose simulators, to better approximate the behavior of the lane-keeping model on a digital twin. We combine the testing outputs of the model on the two simulators in a conservative way, giving priority to the agreements on possible failures, where it is more likely to observe the same failing behavior on the digital twin.

At the model level, our results show that the digital siblings approximate the model predictions on the digital twin better than each individual simulator. At the system-level, the digital siblings are able to predict the failures of the model on the digital twin better than each single simulator.

In our future work we plan to extend our case study to more than two general-purpose simulators, and to study different ways to combine them based on the characteristics of each simulator and those of the digital twin.

Acknowledgements We thank BeamNG GmbH for providing us the license for the driving simulator.

Funding Open access funding provided by Università della Svizzera italiana. This work was partially supported by the H2020 project PRECRIME, funded under the ERC Advanced Grant 2017 Program (ERC Grant Agreement n. 787703).

Data Availability The software artifacts and our results are publicly available Replication package (2023).

Declarations

Conflicts of interests/Competing interests The authors declared that they have no conflict of interest.

References

Afzal A, Katz DS, Le Goues C, Timperley CS (2021) Simulation for robotics test automation: Developer perspectives. In: 2021 14th IEEE conference on software testing, verification and validation (ICST). IEEE, pp 263–274

Almeaibed S, Al-Rubaye S, Tsourdos A, Avdelidis NP (2021) Digital twin analysis to promote safety and security in autonomous vehicles. IEEE Commun Stand Mag 5(1):40–46. https://doi.org/10.1109/MCOMSTD.011.2100004

Amini MH, Naseri S, Nejati S (2023) Evaluating the impact of flaky simulators on testing autonomous driving systems

Arjovsky M, Chintala S, Bottou L (2017) Wasserstein generative adversarial networks. In: International conference on machine learning. PMLR, pp. 214–223

Barosan I, Basmenj AA, Chouhan SGR, Manrique D (2020) Development of a virtual simulation environment and a digital twin of an autonomous driving truck for a distribution center. Software architecture. Springer, Cham, pp 542–557

Barry PJ, Goldman RN (1988) A recursive evaluation algorithm for a class of catmull-rom splines. SIGGRAPH Comput, Graph

BeamNG.research (2022) BeamNG GmbH. https://www.beamng.gmbh/research

Ben Abdessalem R, Nejati SC, Briand L, Stifter T (2018) Testing vision-based control systems using learnable evolutionary algorithms. In: 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE)

Bewley A, Rigley J, Liu Y, Hawke J, Shen R, Lam VD, Kendall A (2019) Learning to drive from simulation without real world labels. In: 2019 International conference on robotics and automation (ICRA). IEEE, pp 4818–4824

BGR Media L (2018) Waymo's self-driving cars hit 10 million miles. https://techcrunch.com/2018/10/10/waymos-self-driving-cars-hit-10-million-miles

Biagiola M, Klikovits S, Peltomaki J, Riccio V (2023) Sbft tool competition 2023-cyber-physical systems track. In: 16th IEEE/ACM international workshop on Search-Based And Fuzz Testing, SBFT

Bojarski M, Del Testa D, Dworakowski D, Firner B, Flepp B, Goyal P, Jackel LD, Monfort M, Muller U, Zhang J, Zhang X, Zhao J, Zieba K (2016) End to end learning for self-driving cars. CoRR **abs/1604.07316**

Borg M, Abdessalem RB, Nejati S, Jegeden FX, Shin D (2021) Digital twins are not monozygotic–cross-replicating adas testing in two industry-grade automotive simulators. In: ICST '21. IEEE

Borji A (2019) Pros and cons of gan evaluation measures. Comput Vision Image Understand 179:41–65

Bottou L, Bousquet O (2007) The tradeoffs of large scale learning. In: Proceedings of NIPS '07

Boutan E (2020) Autonomous driving market overview. https://medium.com/swlh/autonomous-driving-market-overview-b8c71d81c072

Cerf VG (2018) A comprehensive self-driving car test. Commun ACM 61(2)

Cheng M, Zhou Y, Xie X (2023) Behavexplor: Behavior diversity guided testing for autonomous driving systems. In: Proceedings of the 32nd ACM SIGSOFT International Symposium on Software Testing and Analysis, pp 488–500. Association for Computing Machinery. https://doi.org/10.1145/3597926.3598072

Conover WJ (1999) Practical nonparametric statistics, vol. 350. John Wiley & Sons

DeepHyperion (2022) Replication package. https://github.com/testingautomated-usi/DeepHyperion

U.S. Department of Transportation NHTSA (2007) Pre-crash scenario typology for crash avoidance research

U.S. Department of Transportation UD (2018) A framework for automated driving system testable cases and scenarios. https://rosap.ntl.bts.gov/view/dot/38824/dot_38824_DS1.pdf

Donkey Car (2021). https://www.donkeycar.com/

Farag W (2020) Complex trajectory tracking using pid control for autonomous driving. Int J Intell Transp Syst Res 18(2):356–366

Fraser G, Arcuri A (2012) Whole test suite generation. IEEE Trans Softw Eng 39(2):276–291

Gambi A, Jahangirova G, Riccio V, Zampetti F (2022) SBST tool competition 2022. In: 2022 IEEE/ACM 15th international workshop on Search-Based Software Testing (SBST). IEEE, pp 25–32
Gambi A, Maul P, Mueller M, Stamatogiannakis L, Fischer T, Panichella S (2019) Soft-body simulation and procedural generation for the development and testing of cyber-physical systems. Tech. rep, BeamNG
Gambi A, Mueller M, Fraser G (2019) Automatically testing self-driving cars with search-based procedural content generation. In: Proceedings of ISSTA '19
García S, Strüber D, Brugali D, Berger T, Pelliccione P (2020) Robotics software engineering: A perspective from the service robotics domain. In: Proceedings of ESEC/FSE '20. pp 593–604
Grewal R, Tonella P, Stocco A (2024) Predicting safety misbehaviours in autonomous driving systems using uncertainty quantification p 12
Grigorescu S, Trasnea B, Cocias T, Macesanu G (2020) A survey of deep learning techniques for autonomous driving. J Field Robot 37(3):362–386
Group E (2021) Esi prosivic. https://myesi.esi-group.com/downloads/software-downloads/pro-sivic-2021.0
Haq FU, Shin D, Briand LC (2022) Efficient online testing for dnn-enabled systems using surrogate-assisted and many-objective optimization. In: 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022. ACM, Pittsburgh, PA, USA, May 25-27, 2022, pp 811–822. https://doi.org/10.1145/3510003.3510188
Haq FU, Shin D, Nejati S, Briand L (2021) Can offline testing of deep neural networks replace their online testing? Empir Softw Eng
Hu X, Li S, Huang T, Tang B, Chen L (2023) Sim2real and digital twins in autonomous driving: A survey
Jahangirova G, Stocco A, Tonella P (2021) Quality metrics and oracles for autonomous vehicles testing. In: Proceedings of 14th IEEE International conference on software testing, verification and validation, ICST '21. IEEE
Jha S, Banerjee SS, Tsai T, Hari SKS, Sullivan MB, Kalbarczyk ZT, Keckler SW, Iyer RK (2019) Ml-based fault injection for autonomous vehicles: A case for bayesian fault injection. In: 2019 49th annual IEEE/IFIP international conference on dependable systems and networks (DSN), pp. 112–124. https://api.semanticscholar.org/CorpusID:195776612
Kapteyn MG, Pretorius JVR, Willcox KE (2020) A probabilistic graphical model foundation for enabling predictive digital twins at scale. CoRR abs/2012.05841
Kaur P, Taghavi S, Tian Z, Shi W (2021) A survey on simulators for testing self-driving cars. CoRR abs/2101.05337. arXiv:2101.05337
Kim S, Liu M, Rhee JJ, Jeon Y, Kwon Y, Kim CH (2022) Drivefuzz. In: Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security. ACM. https://doi.org/10.11452F3548606.3560558
Kingma DP, Ba J (2014) Adam: A method for stochastic optimization. arXiv:1412.6980
Kong Z, Guo J, Li A, Liu C (2020) Physgan: Generating physical-world-resilient adversarial examples for autonomous driving. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp 14,254–14,263
Koopman P, Wagner M (2016) Challenges in autonomous vehicle testing and validation. SAE Int J Transp Safety
Kothlow C (2021) The power of a multi-purpose digital twin. https://blogs.sw.siemens.com/simcenter/the-power-of-a-multi-purpose-digital-twin/
Lambertenghi SC, Stocco A (2024) Assessing quality metrics for neural reality gap input mitigation in autonomous driving testing p 12
Li G, Li Y, Jha S, Tsai T, Sullivan M, Hari SKS, Kalbarczyk Z, Iyer R (2020) Av-fuzzer: Finding safety violations in autonomous driving systems. In: 2020 IEEE 31st international symposium on software reliability engineering (ISSRE), pp. 25–36. https://doi.org/10.1109/ISSRE5003.2020.00012
Lukasczyk S, Kroiß F, Fraser G (2020) Automated unit test generation for python. In: International symposium on search based software engineering. Springer, pp 9–24
Luke S (2013) Essentials of metaheuristics, vol. 2. Lulu Raleigh
May C (2019) Why automotive companies outsource software development services. https://medium.datadriveninvestor.com/why-automotive-companies-outsource-software-development-services-54a806458b4?gi=9d9b4f45e9ba
Moghadam MH, Borg M, Saadatmand M, Mousavirad SJ, Bohlin M, Lisper B (2022) Machine learning testing in an adas case study using simulation-integrated bio-inspired search-based testing
Mouret JB, Clune J (2015) Illuminating search spaces by mapping elites. arXiv:1504.04909
Nvidia PhysX (2022) https://developer.nvidia.com/physx-sdk
Panichella A, Kifetew FM, Tonella P (2017) Automated test case generation as a many-objective optimisation problem with dynamic selection of the targets. IEEE Trans Softw Eng 44(2):122–158

Panichella S, Gambi A, Zampetti F, Riccio V (2021) SBST tool competition 2021. In: 2021 IEEE/ACM 14th international workshop on Search-Based Software Testing (SBST). IEEE, pp 20–27

Ramdas A, García Trillos N, Cuturi M (2017) On wasserstein two-sample testing and related families of nonparametric tests. Entropy 19(2):47

Replication package (2023) https://github.com/testingautomated-usi/maxitwo

Riccio V, Tonella P (2020) Model-based exploration of the frontier of behaviours for deep learning system testing. In: Proceedings of ESEC/FSE

Rosique F, Navarro PJ, Fernández C, Padilla A (2019) A systematic review of perception system and simulators for autonomous vehicles research. Sensors 19(3). https://doi.org/10.3390/s19030648

Saad D (1998) Online algorithms and stochastic approximations. Online Learn

Saito T, Rehmsmeier M (2015) The precision-recall plot is more informative than the roc plot when evaluating binary classifiers on imbalanced datasets. PloS one 10(3):e0118,432

San O (2021) The digital twin revolution. Nat Comput Sci 1(5):307–308

Software SDI (2022) Simcenter prescan. https://www.plm.automation.siemens.com/global/en/products/simcenter/prescan.html

Stocco A, Nunes PJ, d'Amorim M, Tonella P (2022) Thirdeye: Attention maps for safe autonomous driving systems. In: Proceedings of 37th IEEE/ACM international conference on automated software engineering, ASE '22. IEEE/ACM

Stocco A, Pulfer B, Tonella P (2022) Mind the gap! A study on the transferability of virtual vs physical-world testing of autonomous driving systems. IEEE Trans Softw Eng. https://ieeexplore.ieee.org/document/9869302

Stocco A, Pulfer B, Tonella P (2023) Model vs system level testing of autonomous driving systems: A replication and extension study. Empir Softw Eng

Stocco A, Tonella P (2020) Towards anomaly detectors that learn continuously. In: Proceedings of 31st International Symposium on Software Reliability Engineering Workshops, ISSREW 2020. IEEE

Stocco A, Tonella P (2021) Confidence-driven weighted retraining for predicting safety-critical failures in autonomous driving systems. J Softw: Evol Process. https://doi.org/10.1002/smr.2386

Stocco A, Weiss M, Calzana M, Tonella P (2020) Misbehaviour prediction for autonomous driving systems. In: Proceedings of 42nd International Conference on Software Engineering, ICSE '20. ACM

Talwar D, Guruswamy S, Ravipati N, Eirinaki M (2020) Evaluating validity of synthetic data in perception tasks for autonomous vehicles. In: 2020 IEEE international conference on Artificial Intelligence Testing (AITest). IEEE, pp 73–80

Tang S, Zhang Z, Zhang Y, Zhou J, Guo Y, Liu S, Guo S, Li Y, Ma L, Xue Y, Liu Y (2022) A survey on automated driving system testing: Landscapes and trends. arXiv:2206.05961, https://doi.org/10.48550/arXiv.2206.05961

Tawn Kramer ME contributors (2022) Donkeycar. https://www.donkeycar.com/

Team Chauffeur (2016) "Steering angle model: Chauffeur.". https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/chauffeur

Team Epoch (2016) "Steering angle model: Epoch.". https://github.com/udacity/self-driving-car/tree/master/steering-models/community-models/cg23

Team U (2019) Udacity's self-driving car simulator. https://github.com/tsigalko18/self-driving-car-sim

Team U (2020) Udacity self-driving car challenge. https://github.com/udacity/self-driving-car/

Tian Y, Pei K, Jana S, Ray B (2018) Deeptest: Automated testing of deep-neural-network-driven autonomous cars. In: Proceedings of ICSE '18. ACM

Udacity (2017) A self-driving car simulator built with Unity. https://github.com/udacity/self-driving-car-sim. Online; accessed 18 August 2019

Unity3d (2021) https://unity.com

van Dinter R, Tekinerdogan B, Catal C (2022) Predictive maintenance using digital twins: A systematic literature review. Inf Softw Technol

Veledar O, Damjanovic-Behrendt V, Macher G (2019) Digital twins for dependability improvement of autonomous driving. In: Systems, software and services process improvement: 26th European Conference, EuroSPI 2019, Edinburgh, UK, September 18–20, 2019, Proceedings 26. Springer, pp 415–426

Verma A, Bagkar S, Allam NVS, Raman A, Schmid M, Krovi VN (2021) Implementation and Validation of Behavior Cloning Using Scaled Vehicles. In: SAE WCX digital summit. SAE international. https://doi.org/10.4271/2021-01-0248

Viitala A, Boney R, Kannala J (2020) Learning to drive small scale cars from scratch. CoRR abs/2008.00715. arXiv:2008.00715

Waabi World (2022) https://waabi.ai/waabi-world/

Waymo Simulation City (2021) https://waymo.com/blog/2021/06/SimulationCity.html

Wayve (2022) Introducing wayve infinity simulator. https://wayve.ai/blog/introducing-wayve-infinity-simulator/
Yun H, Park D (2021) Simulation of self-driving system by implementing digital twin with gta5. In: 2021 International Conference on Electronics, Information, and Communication (ICEIC). pp 1–2. https://doi.org/10.1109/ICEIC51217.2021.9369807
Yurtsever E, Lambert J, Carballo A, Takeda K (2020) A survey of autonomous driving: Common practices and emerging technologies. IEEE Access 8:58443–58469
Zhang M, Zhang Y, Zhang L, Liu C, Khurshid S (2018) Deeproad: Gan-based metamorphic testing and input validation framework for autonomous driving systems. In: Proceedings of ASE '18
Zhong Z, Kaiser G, Ray B (2021) Neural network guided evolutionary fuzzing for finding traffic violations of autonomous vehicles
Zhou H, Chen X, Zhang G, Zhou W (2021) Deep reinforcement learning for autonomous driving by transferring visual features. In: 2020 25th International Conference on Pattern Recognition (ICPR). https://doi.org/10.1109/ICPR48806.2021.9412011
Zhu JY, Park T, Isola P, Efros AA (2017) Unpaired image-to-image translation using cycle-consistent adversarial networks. In: Computer vision (ICCV), 2017 IEEE international conference on
Zohdinasab T, Riccio V, Gambi A, Tonella P (2021) Deephyperion: exploring the feature space of deep learning-based systems through illumination search. In: Proceedings of ISSTA '21
Zohdinasab T, Riccio V, Gambi A, Tonella P (2022) Efficient and effective feature space exploration for testing deep learning systems. ACM Trans Softw Eng Methodol

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Matteo Biagiola Matteo Biagiola is a Postdoctoral Researcher at the Software Institute of Universitá della Svizzera italiana (USI) in Lugano, Switzerland. He obtained his Ph.D degree in 2020 from Universitá degli Studi di Genova, Genova, Italy, in a joint collaboration with Fondazione Bruno Kessler, Trento, Italy. Software testing is his main research interest, with a particular focus on test generation for Web applications and learning-based systems. He serves as a reviewer for Software Engineering conferences (e.g., ICSME 2024, ICST 2024, ESEM 2024) and journals (e.g., TOSEM, TSE and EMSE).

Andrea Stocco Andrea Stocco is an Assistant Professor at the Technical University of Munich at the Chair of Software Engineering for Data-intensive Applications of the School of Computation, Information and Technology. He is also the head of the Automated Software Testing unit at fortiss. His research interests include software testing and empirical software engineering, with particular emphasis on misbehavior prediction for machine learning-based systems, and automated repair, robustness and maintainability of test suites for web applications. He is the recipient of the Paper Award at the 16th International Conference on the Quality of Information and Communications Technology (QUATIC 2023) and the Best Student Paper Award at the 16th International Conference on Web Engineering (ICWE 2016). He serves on the program committees of top-tier software engineering conferences such as ICSE, FSE and ICST, and reviews for numerous software engineering journals including TSE, EMSE, TOSEM, JSS, and IST.

Vincenzo Riccio Vincenzo Riccio is an Assistant Professor at University of Udine, Italy. Previously, he was a Postdoctoral Researcher with the Software Institute of Universitá della Svizzera Italiana (USI) in Lugano, Switzerland. He obtained his Ph.D degree from Università degli Studi di Napoli “Federico II”, Italy. His current research is focused on test automation for machine learning-based applications. He serves as a reviewer for Software Engineering conferences (e.g., ESEM 2024, FSE 2024, ISSTA 2024) and journals (e.g., TOSEM and TSE). He is part of the organizing committee of workshops (DeepTest and SBFT) and conferences (SANER and SSBSE). He is Guest Editor of the EMSE journal’s special issue on Innovations in Software System Testing with Deep Learning.

Paolo Tonella Paolo Tonella is Full Professor at the Faculty of Informatics and at the Software Institute of Universitá 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 66. He is/was in the editorial board of TOSEM, TSE and EMSE. He is Program Co-Chair of ESEC/FSE 2023. 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.