



Causality-driven Testing of Autonomous Driving Systems

LUCA GIAMATTEI, ANTONIO GUERRIERO, ROBERTO PIETRANTUONO, and STEFANO RUSSO, DIETI, Università degli Studi di Napoli Federico II, Italy

Testing Autonomous Driving Systems (ADS) is essential for safe development of self-driving cars. For thorough and realistic testing, ADS are usually embedded in a simulator and tested in interaction with the simulated environment. However, their high complexity and the multiple safety requirements lead to costly and ineffective testing. Recent techniques exploit many-objective strategies and ML to efficiently search the huge input space. Despite the indubitable advances, the need for smartening the search keep being pressing. This article presents CART (*Causal-Reasoning-driven Testing*), a new technique that formulates testing as a causal reasoning task. Learning causation, unlike correlation, allows assessing the effect of actively changing an input on the output, net of possible confounding variables. CART first infers the causal relations between test inputs and outputs, then looks for promising tests by querying the learnt model. Only tests suggested by the model are run on the simulator. An extensive empirical evaluation, using Pylot as ADS and CARLA as simulator, compares CART with state-of-the-art algorithms used recently on ADS. CART shows a significant gain in exposing more safety violations and does so more efficiently. More broadly, the work opens to a wider exploitation of causal learning beside (or on top of) ML for testing-related tasks.

CCS Concepts: • **Software and its engineering** → **Software testing and debugging**;

Additional Key Words and Phrases: Self-driving cars, autonomous vehicles, AI testing, search-based software testing, causal reasoning

ACM Reference format:

Luca Giamattei, Antonio Guerriero, Roberto Pietrantuono, and Stefano Russo. 2024. Causality-driven Testing of Autonomous Driving Systems. *ACM Trans. Softw. Eng. Methodol.* 33, 3, Article 74 (March 2024), 35 pages. <https://doi.org/10.1145/3635709>

1 INTRODUCTION

1.1 The ADS Testing Challenges

Autonomous Driving Systems (ADS) for self-driving cars are learning-enabled systems based on **Deep Neural Networks (DNNs)**, capable of sensing the environment and making decisions to drive the car safely with little or no human driver input [83]. Their spread is expected to increase in the upcoming years: a recent report projects the self-driving cars market size to surpass USD

This project has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 871342 "uDEVOPS".

Authors' addresses: L. Giamattei, A. Guerriero, R. Pietrantuono, and S. Russo, DIETI, Università degli Studi di Napoli Federico II, Via Claudio 21, 80125, Napoli, Italy; e-mails: {luca.giamattei, antonio.guerriero, roberto.pietrantuono, stefano.russo}@unina.it.



This work is licensed under a Creative Commons Attribution-NonCommercial-ShareAlike International 4.0 License.

© 2024 Copyright held by the owner/author(s).

1049-331X/2024/03-ART74

<https://doi.org/10.1145/3635709>

65 million dollars by 2030, expanding growth at a rate of 13.38% over 2022 to 2030 [1]. For people to justifiably trust ADS (and for companies to sustainably develop them), a fundamental challenge is how to effectively and efficiently test their behaviour.

A primary focus of researchers is on testing decision-making DNNs as standalone components [70], based on datasets obtained without involving the DNNs under test. This is referred to as *model-level testing* [83] or *offline testing* [32]. Several such strategies have been proposed, including: DeepXplore [66], DeepRoad [100], DeepTest [89], and DeepGauge [48]. Though important, model-level testing inherently misses the ability to spot system-level misbehaviours; e.g., negligible DNN mispredictions can accrue over time and expose failures despite high DNN accuracy [83], [32], [31].

For more realistic tests, *system-level testing*, also called *online testing*, is applied rather than (or beside) *model-level testing* [4, 24, 30, 51, 91]. In this case, the DNNs are embedded into a simulated driving environment and tested in a closed-loop mode in interaction with the environment. System-level testing is able to account for the effect that the DNN predictions have on the environment and on the behaviour of the whole system. However, despite recent efforts [3, 5, 24, 30, 32, 51, 71, 83, 91], it remains challenging to generate safety-critical test scenarios in an efficient way [30]. The space of all possible tests for the whole system, considering the input from all sensors such as cameras, LiDAR, and GPS, is very large. In addition, the difficulty in exploring this space is exacerbated by the need of testing multiple safety requirements at the same time, such as distance from objects, from pedestrians, from the center of the lane, which entails a multi- or many-objective search. Moreover, the evaluation of every generated test scenario is expensive: the use of high-fidelity simulators indeed reduces the cost of a real-world large-scale testing process (and is of course safer), but running a single test scenario takes several minutes and is computationally expensive. The combination of these challenges requires a testing strategy that intelligently explores the space of all possible tests and generates only the most promising ones to expose critical behaviours.

1.2 Contribution

To address the above challenges, we propose **CART (CAusal-Reasoning-driven Testing)**, a testing technique that *i)* learns the causal relations between test inputs and outputs, represented in a causal model, and then *ii)* uses causal inference to query the model to estimate the outcome of hypothetical tests without actually executing them, thus drastically reducing the testing cost.

CART exploits the ability of causal learning and causal inference to go beyond the conventional association-based paradigm featured by **Machine Learning (ML)**. For testing an ADS at scale, knowledge needs to be inferred proactively: ML learns (input-output) patterns from observations and predictions tell, based on the learnt distributions, what is the expected output when we *observe* a certain input; with causality, we learn the cause-effect relations, and predictions tell what is the expected output when we *actively set* a certain input variable (hence, when we change its distribution) [64, 65]. This feature, we believe, better reflects the reasoning of a human tester: when we conceive a test case, we are ultimately trying to predict *what input makes the system fail?* This is much more than just searching for failure patterns. With that question, we do not really mean *what input is more correlated to failure?* but rather *what input causes the system to fail?* In other words, we look for causal explanation, not for correlations. CART tries to imitate this tester's reasoning ability of pre-visioning the effect of possible changes of an input on safety requirements, and then largely amplifies this ability with the power of computation.

Following this view, CART acts in two phases. From an initial set of tests or from historical driving data, it builds and iteratively refines a causal model, capturing the cause-effect relations between inputs and outputs via **Causal Structure Discovery (CSD)** algorithms. Then, it uses the *do*-calculus for **Causal Inference (CI)** [64] to query the model for estimating the effect on safety

requirements of a hypothetical change of a given input variable – namely, of a hypothetical new test case. Only the most promising tests will be actually executed on the simulator.

We extensively experiment CART to assess its ability of generating effective and efficient test scenarios to detect safety violations, comparing it with state-of-the-art strategies for many-objective search used in a recent ICSE work on ADS testing [30]. We use a high-fidelity driving simulator, CARLA [17], along with an advanced ADS, Pylot [26], both representative and widely-used choices in ADS testing [19],[30],[56].

The main contributions of this article are:

- A new technique for effective and efficient testing of ADS, exploiting causal reasoning.
- An empirical evaluation of CART in comparison with state-of-the-art strategies. Results show the potential of exploiting causal inference to boost testing performance, highlighting a significant gain of CART in exposing more safety violations and doing so more efficiently.
- At a higher level, a conceptual leap compared to existing strategies is the use of causal reasoning embedded into the testing process. Causal reasoning has great potential to support test automation and to automatically suggest (or corroborate the engineer’s belief about) the most promising failure-exposing input variables, by enabling the simulation of *what happens if* scenarios. As a side effect, CART yields an interpretable representation of the causal relations between the involved variables, which can also be easily adjusted by domain expert: this is a useful asset even beyond testing, as the model can be queried to support several quality assurance activities.
- The release of a publicly available replication package, including the implementation of CART and a set of test scenarios, which can be reused by other researchers.¹

The rest of the paper is structured as follows: Section 2 gives background notions on causal inference and causal structure discovery. Section 3 reviews the related work. Section 4 presents the CART strategy. Section 5 and 6 report about the empirical evaluation. Section 7 discusses the threats to validity, while Section 8 concludes the paper.

2 BACKGROUND

2.1 Causal Inference

Causality can be defined as the influence by which an event contributes to the production of other events [57]. Causal Inference (CI) aims to estimate the impact of a change of a certain variable over an outcome of interest. It is fundamentally different from ML, as the latter aims at uncovering patterns in observed data that connect inputs and output. For decades researchers have tried to explain causality through statistical/ML methods identifying associations between variables (e.g., correlation, regression), which however cannot distinguish between cause and effect. These are limited to what Pearl and Mackenzie called the first out of the three rungs of the ladder of causation [65], that is *association*, solely based on observations.

ML allows answering questions of this form: given the same context in which we learned the model (namely, the same data distribution), *what output X do we expect if the input W happens to be equal to w ?* This query spots possible correlations between variables. Causal reasoning targets causation, addressing questions as: *What output X do we expect if we actively set W to w (thus, if we change the distribution)?* And: *What would have happened to X if we had set W to w ?* For these questions, called, respectively, *interventions* and *counterfactuals* (rung 2 and 3 of the ladder

¹The replication package is available on Figshare at: <https://doi.org/10.6084/m9.figshare.21937121>

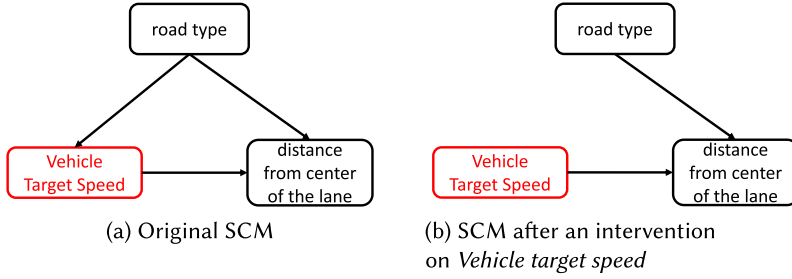


Fig. 1. Example of Structural Causal Model and intervention.

of causation), ML falls short, because it predicts solely based on what has been *seen*. With causal questions, we can proactively envision what happens (or would have happened) if we *do* (or *had done*) an intervention (i.e., set W to w). It goes beyond observation.

To run these queries, a CI engine requires a causal model to represent relationships between involved variables. **Graphical Causal Models (GCMs)** are a widely used solution. A GCM is a causal **Direct Acyclic Graph (DAG)**, where nodes are random variables and edges capture the hypothesis that the two connected variables would be associated if all other variables were fixed while the tail variable is varied [25]. The most general case of a GCM is a probabilistic causal model, which uses (conditional) stochastic models to represent causal relations, with each node X_i characterized by the conditional distribution given its parents $Pa(X_i)$ (for non-root nodes) or simply by a distribution (for root nodes). In a **Functional Causal Model (FCM)**, the value of each variable X_i is assumed to be a deterministic function of its parents $Pa(X_i)$ and of the unmeasured disturbance U_i ($X_i = f(Pa(X_i), U_i)$). These are a non-linear non-parametric generalization of linear **Structural Equation Models (SEMs)**. A GCM using an FCM for conditional distributions is called a **Structural Causal Model (SCM)** [64]. Specifically:

Definition 2.1. Structural Causal Model (SCM). An SCM is a Directed Acyclic Graph $\mathcal{G} = (X, \mathcal{E})$, where nodes $\in X$ are random variables and edges $\in \mathcal{E}$ are the causal relationships between them. Causal relationships are described as a collection of structural assignments $X_i := f_i(Pa(X_i), U_i)$ that define the (endogenous) random variables X_i as a function of their parents $Pa(X_i)$ and of (exogenous) independent random noise variables U_i .

Figure 1(a) shows an excerpt of an SCM from our dataset, where *vehicle target speed* causally affects the distance from lane center, and *road type* affects both speed and lane center distance.

In causal inference, we are interested in the distribution of an outcome variable X_i after setting another variable X_k to a certain value x (i.e., *doing* an intervention). Pearl introduced the *do-operator*, a mathematical representation of physical intervention, written as $P(X_i | do(X_k = x))$ [64]. An intervention $do(X_k = x)$ changes the SCM graph (hence the distribution), by removing the causal relations with its predecessors (i.e., deleting the $Pa(X_k) \rightarrow X_k$ arrows, see Figure 1(b)), meaning that X_k is no longer affected by any other variable.

Definition 2.2. Intervention distribution. The probability $P(X_i | do(X_k = x))$ over an SCM is the distribution entailed by the SCM obtained by replacing the definition $X_k := f_k(Pa(X_k), U_k)$ with $X_k := x$.

A *do* intervention changes the data generative process, thus: $P(X_i | do(X_k = x)) \neq P(X_i | (X_k = x))$. In ML-based inference, from the joint distribution of the two variables X_i and X_k , we aim at inferring the conditional distribution $P(X_i | X_k)$ (i.e., what is the outcome X_i if X_k happens to be

equal to x). In CI, we are interested in the distribution of the outcome we would encounter if we set X_k to a particular value x . The *do*-calculus (along with estimation methods [57]) supports the inference of type $P(X_i|do(X_k = x))$ [64]. Through three main rules, it basically allows expressing a *do* operation in terms of conditional distributions of a set of related variables, properly identified by graph patterns (e.g., *back-door*, *front-door*, *instrumental variable*). This supports both interventional and counterfactual queries. For instance, in Figure 1, let us assume we want to assess the effect of *vehicle target speed* V on the *distance* D from the center of the lane. In this case, *road type* R is said to be a *confounder*: in fact, if we observe a correlation between V and D this may be well due to the influence of R on both, and we may falsely conclude that the V causes D . This is called a *back-door* pattern. If we generated test cases based just on this observed correlation $P(D|V)$, we might fail to expose violations of the distance from the lane center requirement by just changing the vehicle target speed and ignoring the road type. If we, instead, ask for the causal effect by doing an intervention (obtaining the graph in Figure 1(b), namely $P(D|do(V))$), we obtain the real effect of V on D , net of R . By the *do*-calculus rules, we come to marginalizing over R to get the desired effect: $P(D|do(V = v)) = \sum_r P(D|V = v, R = r) \cdot P(R = r)$. Our aim is exactly to generate tests exploiting such real cause-effect relations.

Note that back-door is the simplest case: more complex patterns can entail far longer chains of transformations that are not easy to derive without *do*-calculus. Its rules not only dramatically simplify the transformations, but have also been shown to be *complete* [35]: if a causal effect is identifiable, there exists a sequence of applications of the three rules that transforms the causal “*do*” formula into a formula containing only observational quantities (i.e., conditional probabilities).

In summary, given an SCM, we can run a *do*-query specifying one (or multiple) interventions, such as $P(X_i|do(X_k = x, X_j = y))$ and get the answer by these steps:

- (1) In the graph, a *do*-intervention on X_i (a.k.a. *treatment*) corresponds to cutting all incoming edges to X_i (as the other variables will no longer affect X_i), yielding a new SCM. The problem is thus how to estimate the conditional distribution $P'(X_i|X_k, X_j) \neq P(X_i|X_k, X_j)$ in the new graph, given the observations. The *do*-calculus leverages methods to identify the causal effects correctly in the new graph, by conditioning on a set of additional variables linked to X_i exploiting some patterns in the graph (e.g., the *back-door*, *front-door*, *mediators*, *instrumental variables* identification methods [36]). This step corresponds to the identification of the *estimand*, just like it is done in the back-door example above. In summary, identifying the estimand consists in finding the mathematical formula that generates the answer to a causal query, given the data. Note that it may not always be possible to identify the estimand from the model: for example imagine that *road type*, that has an effect on both *vehicle target speed* (V) and *distance from the center of the lane* (D), is not measurable; then the query $P(D|do(V))$ cannot be answered. In that case, the model should be refined by adding new knowledge or by making simplifying assumptions (e.g., by using a measured variable claimed to be related to the unmeasured one).
- (2) Once the estimand is identified, a variety of statistical methods can be used to estimate the model’s parameters from observed data [57], such as: propensity-based stratification [72], propensity score matching [7], inverse propensity weighting [92], regression discontinuity [88], two-stage least square [87], and generalized linear models [57]. This gives the *estimate* for the identified estimand.
- (3) By the structural equations (see Definition 2.1) in the new parameterized SCM, we can quantify the impact of the intervention looking at the expected outcome on units assigned to the treatment ($E[X_i|do(X_k = x)]$), as well as at the difference in the expected outcomes between units assigned to the treatment and units not treated (known as **Average Treatment**

Effect, ATE).² Other metrics of interest can be the **Individual Treatment Effect (ITE)**, **ATE on Treated (ATT)**, or **Conditional Average Treatment Effect (CATE)** [57], all derivable from the post-intervention SCM.

A valuable alternative to the analytical derivation of the effect estimate is to use simulation. Simulation-based inference samples from the post-intervention distributions. This gives a new data sample under the intervention, which can be used to estimate the post-intervention expected outcome by just looking at the statistics of the sample. This also allows an easy derivation of confidence intervals.

Causal inference is enabled by many software tools that have been recently developed, mainly in R [22, 34, 53, 60, 74] or Python [9, 16, 76]; a detailed description of the most notable ones has been carried out by Nogueira et al. [57]. The most complete tool designated is DoWhy [76], developed by Microsoft, which covers all the process for causal inference (*modelling* a causal problem, *identifying* a target estimand, *estimating* causal effect based on identified estimand, and also possibly running a series of *refutation* tests on the estimate used to increase the confidence in the estimate). DoWhy offers a wide number of estimation methods and supports both the analytical and the simulation-based inference through an extension called DoWhy-GCM [6]. This extension provides an easy and automatic way to answer causal questions, such as simulating the impact of interventions, computing counterfactuals, estimating average causal effects, and attributing distributional changes. CART uses DoWhy-GCM for causal inference.

2.2 Causal Structure Discovery

Causal inference requires a model (e.g., a GCM) capturing the causal relations between the variables. There are generally two ways for building such a model: by explicit interventions (e.g., controlled experiments), in which we manipulate some variables and see the effect on the others; or from already-available data, by observing the variations of the variables of interest without manipulating them. The former strategy is more accurate, but it requires controlled experiments that may be expensive or even technically impossible to run. It is therefore often necessary to discover the causal relations from observations.

Causal Structure Discovery (CSD) algorithms infer the causal structure, represented as a GCM, from observed data, under the assumption that causality can be detected from statistical dependencies. Roughly speaking, the input of these algorithms is a dataset of observations, and the output is a DAG along with the structural equations linking variables to their direct causes. CSD algorithms rely, to a different extent, on various subsets of assumptions, the main ones being the *Causal Markov*, *Faithfulness*, and *Sufficiency* assumptions, to derive correspondences between the (conditional) independence in the probability distribution and the causal connectivity relationships in the generated DAG [20]. The first two conditions are the most important ones and respectively state that: *i*) every vertex X in the graph G is probabilistically independent of its non-descendants given its parents; *ii*) if a variable X is independent of Y given a conditioning set Z in the probability distribution,³ then X is *d-separated*⁴ from Y given Z in the DAG (in other words, the statistical dependence between variables estimated from the data does not violate the independence defined by any causal graph that generates the data [29]). Markov and faithfulness conditions are sufficient

²For instance, if X_k is binary, $ATE = E[X_i | do(X_k = 1)]E[X_i | do(X_k = 0)]$ [57].

³Roughly, X and Y are independent conditional on a set of variable Z if knowledge about X gives no extra information about Y once you have knowledge of Z .

⁴Let X, Y , and Z be disjoint subsets of all the vertex in the DAG. Z *d-separates* X and Y just in case every path from a variable in X to a variable in Y contains at least one vertex X_i such that either: *i*) X_i is a *collider* (i.e., the arrows converge on X_i in the path), and no descendant of X_i (including X_i) is in Z ; or *ii*) X_i is not a collider, and X_i is in Z .


to define an equivalence structure over directed acyclic graphs, where graphs that are in the same Markov equivalence class have the same (conditional) independence structure (an inherent limitation of CSD algorithms is that they are able to identify structures only up to Markov equivalence class). Sufficiency requires that for a pair of observed variables, all their common causes must also be observed in the data (and modeled in the graph).

CSD algorithms can be categorized into constraint-based, score-based, and FCM-based [25].

Constraint-based algorithms use conditional independence tests on observed data to identify a set of edge constraints [79]. These algorithms have the benefit of being generally applicable, despite the fact that they are based on the strong assumption of causal faithfulness and may therefore require large sample sizes to perform well [25]. Typical (conditional independence) constraint-based algorithms are PC, FCI [79] and its improvement RFCI [13].

Score-based algorithms optimize a score assigned to candidate graphs, exploiting adjustment measures such as the Bayesian Information Criterion, which approximates the posterior probability of the model given the data (assuming a uniform prior probability distribution over the DAG space) [52]. They relax the faithfulness assumption by replacing conditional independence tests with the goodness-of-fit tests. They are computationally expensive, as they enumerate (and score) every possible graph among the given variables. Well-known representatives are GES [11] and its successor FGES [68], which use parallelization to optimize performance. GFCE is another well-known algorithm that combines FCI and FGES [58].

Algorithms based on FCM determine the causal direction of edges, identifying the true causal structure out of all the graphs within a Markov equivalence class. The most noticeable example, working for continuous variables, is LinGaM, proposed by Shimizu *et al.* [77], where the model is assumed to be linear and non-Gaussian. Under the causal Markov assumption, acyclicity and a linear non-Gaussian parameterization (i.e., each variable is determined by a linear function of the values of its parents and an additive non-Gaussian noise term), it has been proved that the causal structure can be uniquely determined [80]. FCM-based algorithms have the advantage of not relying on the faithfulness assumption and of learning substantially more about the causal structure, sometimes even determining a unique model. However, to relax faithfulness, they introduce other assumptions that can be controversial or difficult to test [52], and generally require larger sample size to be accurate. An extensive discussion of CSD algorithms can be found in [25, 29, 57].

The implementation of CSD algorithms is provided by various tools/libraries, the most common ones being [57]: `pcaIlg` [39], `bnlearn` [73], and `Tetrad` [69]. The first two are well-known  libraries while the last is a Java tool. `Tetrad` also provides the source code enabling the development of other libraries on top of it, such as `pycausal`⁵, which wraps `Tetrad` functionalities in Python. `CART` uses `pycausal` for causal structure discovery.

3 RELATED WORK

Model-level (or *offline*) testing of ADS has been used extensively for testing the individual DNNs by either using adversarial examples (e.g., corrupted images) [48, 49, 66, 100, 102] and by using **Generative Adversarial Networks (GANs)** to perturb the input [42, 62, 95, 99, 100]. This type of testing is supported by coverage criteria such as neuron coverage [48, 66], combinatorial coverage [50], and “discrepancy” between training/validation data and test data [40, 98]. Besides exposing mispredictions, a different goal is to sample a minimal subset from the operational inputs set that, once manually labelled, can closely assess the accuracy [10, 28, 45].

⁵<https://zenodo.org/record/3592985>

Model-level testing does not test the whole system in its environment. Recently, *system-level* (or *online*) testing has been more widely investigated. On this line, researchers proposed solutions to generate scenarios that cause the system to misbehave [4, 24, 30, 32, 51, 71, 83, 91].

Gambi et al. present AsFAULT [24], a genetic algorithm combined with procedural content generation – a technique employed in video games for the automatic creation of virtual environments [90] – to generate virtual roads causing the ego vehicle to depart from the center of the lane.

DeepJanus [71] is a search-based tool that generates frontier inputs, i.e., similar input pairs that cause the ADS to mispredict for one input and work fine for the other one.

Tuncali et al. present SIM-ATAV [91] that combines combinatorial testing with requirements falsification; they use covering array as combinatorial test generation approach for discrete variables, and a falsification approach using uniform random search or, again, a search-based algorithm (simulated annealing) to search over continuous variables.

Majumdar et al. propose Paracosm [51], a simulation-based testing language, associated with a tool, that generates tests using random sampling for discrete parameters, and deterministic quasi-Monte Carlo methods [55] for continuous parameters to achieve high diversity.

Klishat and Althoff [41] propose an approach for testing of motion planning algorithms in ADS; it automatically generates critical scenarios based on a minimization of the solution space of the vehicle under test via evolutionary algorithms. Calò et al. [8] also use search-based techniques aiming at finding *avoidable* collision scenarios (i.e., scenarios in which the collision would not have occurred with a reconfiguration of the ADS). They first search for a collision and then for an alternative configuration of the ADS which avoids it.

Li et al. [44] present AV-FUZZER, a framework aiming at finding single-objective safety violations by perturbing driving maneuvers of traffic participants (i.e., acceleration/deceleration, following lane, and making lane change). Metamorphic testing is also used in combination with equivalent partition testing for system-level ADS testing [63].

Other studies focus on system-level testing, but not with the objective of generating test cases. Stocco et al. [83] compare virtual and physical-world system-level testing; in [85], the same authors propose an oracle for mispredictions detection; in [84], the authors use knowledge inferred from field execution to predict misbehaviours; Haq et al. compare online and offline testing [32].

The above studies pursue testing efficiency by trying to tackle the large-state-space challenge. None of them focuses on the additional challenges brought by the need of considering many safety requirements at the same time, which entails a many-objective search. Abdessalem et al. [4] use ML models (decision trees) combined with the NSGA-II search-based multi-objective algorithm to guide the search – three objectives are considered. The same authors [5] are the first ones to formulate the ADS testing problem as a many-objective search. Luo et al. propose EMOOD, an approach that uses an evolutionary algorithm to generate test scenarios to expose as many combinations of requirements violations as possible [47]. Recently, Haq et al. [30] adopt the same formulation, and address the expensive test case evaluation challenge too. They propose SAMOTA, a technique that uses surrogate models (based on regression and radial basis function networks) to predict the outcome of a test case without actually executing it. Similarly to SAMOTA but out of the ADS testing domain, the use of surrogate models to address computationally expensive optimization problems has been widely studied [38], with recent studies combining global and local search [103] and also using the most uncertain candidates in addition to the best predicted ones [46, 93]. This is close to our work, as the objective is to exploit some learnt relationships between test input and output, and use them to generate only interesting (i.e., safety-violating) tests. The key advancement of CART is to inject causal reasoning into the test generation process. For what is discussed, this is a more versatile tool, since causation is more informative than correlation, and learning cause-effect relations enables a tester to spot more precisely input values more likely

to cause a safety violation. Before CART, causality has been applied in software engineering in a few works [78], on metamorphic testing [12, 59], on fault localization [2, 27, 43, 86], and on performance analysis [37, 81]. The next section describes how CART exploits this feature.

4 CART

4.1 Problem definition and notation

The goal of online testing of ADS is to generate a minimal set of driving scenarios to be run on a simulator – called *test scenario* hereafter – that cause the system to violate multiple safety requirements. This is a many-objective optimization problem. A driving scenario is built using a set of variables describing various driving conditions, such as the road type, the presence of other vehicles, trees, buildings, the weather conditions, the speed – these are the input variables of the test scenario. The ADS drives the ego vehicle in that scenario and, at the end of the execution, we check if (one or more) safety violations occurred or not, by inspecting a set of output variables, such as the distance from the center of the lane, the distance from other vehicles or from pedestrians. Figure 1 constitutes a simplified example with two input variables (i.e., “Vehicle target speed” and “Road type”) and a single requirement (i.e., “Distance from center of the lane”). In this example, the goal is to find the combinations of the two inputs that cause a violation of the requirement.

More formally, let us use the following notation:

- $V = X \cup Y = \{x_1, \dots, x_m; y_1, \dots, y_n\}$ is the set of $m + n$ variables to define a test scenario, where X is the set of input variables and Y is the set of output variables.
- $s_k = (\mathbf{x}_k; \mathbf{y}_k) = (x_{k_1}, \dots, x_{k_m}; y_{k_1}, \dots, y_{k_n})$ is the k -th *test scenario* (or *test*) where \mathbf{x}_k (input) and \mathbf{y}_k (output) are the values of the corresponding variables in X and Y taken in the k -th test.
- D is a *database* containing all the executed tests.
- $Thr = (\tau_1, \dots, \tau_n)$ is the set of thresholds associated to output variables in Y , which allows determining if a safety violation occurred or not.
- $R = \{r_1, \dots, r_n\}$ is the set of safety requirements associated with the n output variables and the corresponding thresholds.⁶ A safety requirement r_j is violated when $y_j < \tau_j$ at any time during the simulation. Without loss of generality, we assume that the lower the values of output variables, the closer the requirement violation (e.g., if a “distance from vehicles” output gets close to a threshold of 0.5 meters, a safety requirement is close to being violated). When dealing with variables that are more critical when their value increases (such as “distance from the center of the lane”), its opposite is considered.
- $C \subseteq R$ ($U \subseteq R$) is the subset of safety requirements that have been *covered* (respectively: *uncovered*). Covering a requirement means that at least one test violates it.

Testing can be targeted at covering as many requirements as possible (i.e., maximize the proportion of covered safety requirements, $|C|/|R|$), or at exposing as many safety violations as possible, regardless of requirements already covered. The latter is because testers might want to have multiple diverse tests violating the same requirements that highlight different conditions in which the violation occurs.

4.2 The CART algorithm

Given the set of requirements to violate R , our algorithm addresses the many-objective optimization problem by a classical weighted-sum approach [14, 15], namely combining the many

⁶For simplicity, we assume that a safety requirement is associated with one measured output variable; clearly, the notation can be extended to deal with safety requirements defined on combinations of output variables. In this case $|R| < |Y|$, and a derived output variable expressing the desired combination is considered in the output variables set (e.g.,: $y'_j = f(y_1, \dots, y_n)$, with the associated threshold τ'_j).

objectives into a single one, hereafter called *fitness* and expressed by a fitness function Φ . We consider two fitness functions, for the two above-mentioned testing goals. The former expresses the proportion of covered safety requirements is defined as:

$$\Phi_A(s_k) = \frac{\sum_{j=1}^{|U|} (1 - mM(\mathbf{y}_{k_j}))}{|U|} \quad (1)$$

where $mM(\mathbf{y}_{k_j})$ is the min-Max normalized j -th output variable of the k -th test – lower values are closer to a safety violation – and U is the set of still uncovered safety requirements. This gives priority to tests more likely to affect the output variables related to still-uncovered requirements. Since the set of uncovered requirements changes during testing, this function is called *adaptive*.

The second fitness function maximizes the number of safety violations. This is called *fixed*, and is defined as:

$$\Phi_F(s_k) = \frac{\sum_{j=1}^{|R|} (1 - mM(\mathbf{y}_{k_j}))}{|R|} \quad (2)$$

where R is the set of safety requirements. In this case, the function gives priority to tests more likely to cause a safety violation of any safety requirement, even though some past test already covered it. Whenever possible, we will use ϕ_k to refer to the value of any of the above fitness functions, for test scenario s_k .

At a high level, the CART algorithm acts in two phases.

In the first phase, CART extracts (and iteratively updates) a Structural Causal Model (SCM) from past data, via Causal Structure Discovery (CSD). Such data needs to be in the same format used to define a test case – one entry of the dataset consists of values of the input variables (which correspond to a specific test scenario) and of the obtained output variables in that scenario. In a testing process, this data would naturally come from previous testing sessions; the very first time that CART is executed, the initial set of tests can be obtained by random testing or any other technique. These data could also come from driving data; in such a case, the input and output variables for each scenario need to be extracted from such data (depending on how historical data are gathered and stored) and formatted as test scenarios. The resulting SCM describes the causal relations between the input and output variables of a test scenario (cf. with Definition 2.1).

In the second phase, this model is queried by a Causal Inference (CI) engine, which runs interventional queries such as: *what is the effect on the output variable of interest if we set the input to a given value?* More formally, the CI engine assesses the expected value of the output variable of interest $y_j \in Y$ under a *do*-intervention on the input variable $x_i \in X$: $E[y_j | do(x_i = x)]$, where x is the hypothesized input value (cf. with Section 2.1). For instance, with reference to the example in Figure 1, it means generating queries such as: *what is the effect on “Distance from center of the lane if we set “Vehicle target speed” to 40km/h?* These queries produce a set of “*hypothetical*” tests whose output is an estimate of the expected effect on the output variables of interest under hypothetical inputs. No real test is executed in this phase. Actual real tests to run are then generated by selecting only the best tests from the hypothetical test set. The core idea is to explore the tests search space by causal queries to the model rather than by actually executing the tests (which in contexts like ADS takes several minutes per test).

Algorithm 1 describes the CART steps for tests generation. The algorithm takes, as input, the set of safety requirements R to cover with the associated thresholds E , the database D containing a set of already executed scenarios (e.g., derived from past tests or from driving data), the sample size parameter η_0 used by the Causal Inference (CI) engine as explained below, a flag to select the fitness function, and a parameter ϵ , a probability value for inputs selection policy. It returns the generated test suite S , the updated database D , and the final causal model M , refined over successive

ALGORITHM 1: CART algorithm

Input: D : database of tests; f : boolean to select the adaptive (0) or fixed (1) fitness function; R : safety requirements; Thr : error thresholds; η_0 : CI sample size; ϵ : probability for input selection.
Output: S : Test suite, M : causal model, D : Updated database

```

1:  $S, P \leftarrow \emptyset$ 
2: repeat
3:    $M \leftarrow buildCausalModel(D)$ 
4:    $P \leftarrow updatePopulation(D, T)$ 
5:    $T \leftarrow \emptyset$ 
6:   for  $k = 1$  to  $|P|$  do
7:      $\tilde{H} \leftarrow infer(M, \eta_0, P_k, \epsilon)$             $\triangleright$  Query to the model.  $\tilde{H}$ : set of hypothetical tests
8:      $H \leftarrow estimateFitness(\tilde{H}, f, R, Thr)$ 
9:      $\tilde{t} \leftarrow select(H)$                         $\triangleright t$ : best among  $|H|$  hypothetical tests
10:     $t \leftarrow runOnSimulator(\tilde{t}, R, Thr)$        $\triangleright$  Compute actual fitness
11:     $T \leftarrow T \cup t$ 
12:   end for
13:    $(S, D) \leftarrow (S, D) \cup T$ 
14: until !terminatingCondition
15: return  $S, M, D$ 

```

iterations. The terminating condition could be the exhaustion of the budgeted testing time, or a convergence criterion (e.g., violations no longer found after some iterations). The detailed CART steps follow.

(1) **Causal model construction.**

A structural causal model (SCM) is inferred (line 3) from data contained in D . The SCM is derived by first inferring the graph structure via a Causal Structure Discovery (CSD) algorithm (cf. with Section 2.2). In our implementation, we have tested five CSD algorithms from the Pycausal library based on Tetrad. Then, the stochastic models and functional causal models (FCM) for, respectively, root and non-root nodes are assigned to each variable based on data. Specifically, by using the CI library DoWhy, CART assigns the best-fitting stochastic model among linear, polynomial, and gradient boost, and the best-fitting FCMs among linear and non-linear additive noise models [75]. DoWhy supports a variety of other models, also from the Sklearn and Scipy libraries; we adopted the default configuration.

The so-derived model is refined at every iteration with the updated database D including the scenarios executed in the previous iteration (line 13).

It is worth it to stress that domain knowledge can be leveraged to build or refine the graph – one of the advantages of causal models. This would however require human intervention and domain expertise; for the sake of full automation, we ignore this possibility.

(2) **Causal Inference and test generation** . Lines 6-12 generate tests starting from the population of current tests P and the (updated) causal model M . Tests are derived according to an evolutionary strategy, namely by trying to improve the current population of tests. The population contains at every iteration the top- $|P|$ tests from $D \cup T$, where T is the set of tests selected and executed in the previous iteration (initially empty). The population size is fixed, and we opted for $|P|=|R|$ as in previous studies [5], [30]. For every test s_k in P , denoted as P_k , the causal model M is queried in order to generate multiple hypothetical tests, estimate

their expected fitness (line 7-8), and take the best one (line 9-10). The following steps are carried out:

– *Input variable selection.*

A query is an intervention on an input variable $x_i \in X$ that changes its value to assess the effect on output variables. Therefore, an input variable needs to be selected first. This means, with reference to the simplified example shown in Figure 1, to select one of the two input variables (i.e., “Vehicle target speed” and “Road type”). In CART, we select, with probability ϵ , the input variable in the SCM with the greatest out-degree (counting only edges toward the output variables $y_j \in Y$), with ties broken randomly, since it is the variable expected to impact more safety requirements together (in the example the two input variables have the same out-degree, resulting in a random choice); with probability $1 - \epsilon$, the input variable is selected randomly (all the variables having the same probability) so as to promote diversity.

Thus, a low value of ϵ gives higher diversity, since the input variable on which to intervene would be chosen randomly, scarcely exploiting the knowledge encoded into the model, in favour of exploration. Contrarily, a high value of ϵ leads to choosing, with higher probability, the input variable expected to causally impact on more safety requirements together. In our evaluation, $\epsilon = 0.5$ to balance exploration and exploitation and avoid bias toward one of them.

– *Hypothetical Tests Generation.*

For each P_k , a set of hypothetical test scenarios $\tilde{H} = \{\tilde{h}_1, \tilde{h}_2, \dots, \tilde{h}_q, \dots\}$ is generated by querying the model via interventions on the selected x_i (with q indexing the intervention). The intervention (i.e., setting a value for the selected variable) causes a change in the other variables’ distributions directly or indirectly related to it. CART uses the simulation-based inference (cf. with Section 2) to estimate the expected effect of the intervention on the output variables, using the DoWhy-GCM library [6]. Simulation-based inference draws samples from the post-intervention distributions, namely it uses the SCM after the intervention with the associated distributions, and draws samples from it. Specifically, it *i*) sorts the nodes in topological order, *ii*) samples values from root nodes according to their distribution, and then *iii*) uses the structural equations with randomly sampled noise to compute the values for downstream nodes. The values for those variables not present in the SCM (i.e., with no cause-effect relation with any other variable) are kept with the same value of the P_k test, given as input to the method *infer*. With reference to Figure 1(b), the variable “Vehicle Target Speed” is selected for the intervention; thus the inferential engine fixes the value of this variable, samples from the distribution of “Road type” (since it is a root node), and propagates the sampled values to the only downstream node, which is “Distance from Center of the Lane”, computing new data with the structural equation. The engine draws, for every intervention, samples of size $\eta = \eta_0$ ($\eta_0 = 1,000$ is the default value of DoWhy-GCM), from which we take the expected values of the output variables as post-intervention estimates (denoted as \hat{y}_q for the q -th intervention). It is worth it to stress that these hypothetical tests are queries done to the causal model, and are not scenarios actually executed on the simulator. The output of the query, \hat{y}_q , are estimates of the expected value for each output variable.

As for the value to assign to the intervention variable, several policies are possible (e.g., uniform or non-uniform random sampling, adaptive strategies, learning-based or search-based criteria), also depending on the type of variables, which could be continuous, discrete or mixed. Since, in our case study, we deal only with discrete input variables, we run a query (i.e., do an intervention) for each value of the selected variable. The final number

of hypothetical tests (\tilde{H}) equals the number of interventions. This number can be reduced if needed (for instance, in the presence of continuous variables) by the mentioned policies. In our evaluation, its impact turned out to be negligible compared to tests execution time (see Section 6.5).

– *Fitness estimation and selection.*

The fitness for each hypothetical test $\tilde{h}_q \in \tilde{H}$ is estimated by applying the fitness function (which requires the set of requirements and thresholds, see Equations (1) and (2)) to the generated samples, hence to the output variable estimates, \hat{y}_q , as computed by the CI query. Thus, the estimateFitness() at line 8 uses \hat{y}_q to compute an estimate of the fitness, using one of the two fitness functions (fixed or adaptive, Equations (1) and (2), selected via the boolean f).

This gives the set $H = \{h_1, h_2, \dots, h_q, \dots\}$, with the hypothetical tests \tilde{h}_q along with their fitness estimate $\hat{\phi}_q$ (namely: $h_q = \tilde{h}_q \cup \hat{\phi}_q$), line 8. Only the hypothetical test with the highest $\hat{\phi}_q$ is selected (line 9) and then executed on the simulator to compute its actual fitness ϕ_q (line 10), producing the actual test scenario t . This is added to the set of tests generated in that iteration, T .

The set of so-generated tests are added to both the test suite S and the database D . Note that both S and D are updated in the same way (i.e., by adding T), therefore S can be obtained by simply taking D at the end of the algorithm and removing tests in the initial D given as input. However, we keep both in the pseudo-code for the sake of clarity. The output also includes the causal model M , which is a valuable tool for future tests or even other engineering tasks.

5 EVALUATION

5.1 Compared Techniques

CART is compared to random search and to the following state-of-the-art techniques for many-objective search-based testing: MOSA [61], FITEST [5] and SAMOTA, all used to test ADS in the recent study presenting SAMOTA [30]. Like CART in the adaptive-fitness configuration, these techniques generate a population of solutions aiming to cover a higher proportion of safety requirements. CART, however, addresses the many-objective optimization problem by a classical weighted-sum approach [14, 15], with the normalized objectives having equal weights – cf. with Section 4.2. The combined single objective is used in the test generation process.

MOSA was first used to formulate branch coverage as a many-objective optimization problem. FITEST later extended it, by progressively updating the fitness function marking the already covered requirements as testing progresses, thus reducing the population size and improving efficiency. SAMOTA holds the features of FITEST, but exploits surrogate models to estimate the fitness function without actually executing tests. SAMOTA is the technique closest to CART, as its surrogate models are meant to avoid non-promising simulator runs. We use causal models to this aim in order to support the generation of effective test scenarios.

The mutation and crossover parameters of SAMOTA, FITEST and MOSA are set at the default values used in [30] and in [61] (they all have mutation rate $\frac{1}{psize}$; crossover rate is 0.75 for MOSA and SAMOTA, 0.60 for FITEST). Both SAMOTA configurations are used (SAMOTA-I with the initial database, and SAMOTA-E with the empty database). The implementation of SAMOTA, FITEST and MOSA are taken from the replication package of reference [30]⁷. The random algorithm (called RANDOM hereafter) samples inputs uniformly within their lower-upper bounds. CART has been implemented using `pycausal` for causal structure discovery, and `DoWhy-GCM` for causal inference.

⁷<https://doi.org/10.6084/m9.figshare.16468530>

5.2 Research Questions

- **RQ1** (*Usefulness*). Does causal reasoning support the generation of test scenarios?
 - *RQ1.1*. How does Causal Inference (CI) perform in generating representative test scenarios?
 - *RQ1.2*. How do different Causal Structure Discovery (CSD) algorithms perform?
- **RQ2** (*Coverage of safety requirements*). How do techniques perform in covering safety requirements?
 - *RQ2.1*. How *effective* are the techniques in covering safety requirements?
 - *RQ2.2*. How *efficient* are the techniques in covering safety requirements?
- **RQ3** (*Detection of safety violations*). How do techniques perform in detecting safety violations?
 - *RQ3.1*. How *effective* are the techniques in exposing safety violations?
 - *RQ3.2*. How *efficient* are the techniques in exposing safety violations?
- **RQ4** (*Test suite quality*) What are the fitness and diversity of the generated test suites?

In CART, the CI engine generates what we called “hypothetical” test scenarios, i.e., a combination of test inputs that are not actually run on the simulator to get the output, but are instead used to query the causal model (i.e., to do an *intervention*) and get an estimate of the expected output. Therefore, before assessing CART against the baselines, RQ1 first aims to assess to what extent the expected output of such hypothetical test scenarios are close to the actual test output obtained by running the same test on the simulator. This ability would, by itself, pave the ground to new opportunities in testing, e.g., by exploiting the possibility of running *what happens if* queries to a model (i.e., what is the expected output if we give a certain input) and predict tests outcome without actually running them. Furthermore, as different CSD algorithms can provide different models, hence potentially different results, RQ1 compares five algorithms: PC, FGES, FCI, GFCI, and RFCE.

RQ2 first investigates the proportion of safety requirements violated by at least one test scenario, given a fixed testing budget (RQ2.1); the goal is to have a minimal test suite that covers as many safety requirements as possible. Then, RQ2.2 evaluates the performance over testing time, so as to see which algorithm achieves the goal earlier.

RQ3 evaluates the ability of the compared techniques of generating *critical* (i.e., safety-violating) tests. The fitness function used in RQ2 is what we called the *adaptive* function Φ_A ; this tracks the safety requirements that get covered during testing, so as to orient the test generation toward scenarios more likely to cover the remaining requirements.

Besides requirements coverage, testers might be interested in the number of safety-violating test cases (possibly with more different tests violating a requirement). In fact, having only one example for a violated safety requirement could be not satisfying for engineers; multiple diverse tests violating the same requirements highlight possible different conditions under which the violation occurs, each of which could trigger different faults leading to that violation. Thus, a richer set of safety-violating tests can support the root cause analysis and debugging task. This is pursued by what we called the *fixed* fitness function Φ_F , adopted in RQ3, which pushes toward safety-violating scenarios without looking at the already-covered safety requirements. To this aim, we have implemented Φ_F in CART and have modified SAMOTA, FITEST and MOSA to make them ignore the set of still-uncovered requirements in their fitness function. These modified versions never update the set of covered requirements, always generating tests that try to cover all requirements.

RQ4 investigates the quality of the generated tests, considering their fitness and diversity. As for fitness, we are interested in those test scenarios that, even though not causing a violation of the safety thresholds, push the variables of interest (e.g., *distance from pedestrians*) very close to a violation (i.e., “near-violating” test scenarios). Such tests highlight suspicious behaviours and

potentially dangerous situations. For instance, a scenario in which the car stops very close to a pedestrian might be not desirable even though it is not a collision, and testers may wish to check them. Moreover, since all the compared techniques follow an evolutionary approach (which gradually improves solutions), it is also of interest to check which technique produces higher-fitness tests, as tests with high fitness are more likely to evolve into safety-violating tests if more testing time is available. Finally, multiple similar tests can trigger the same violation or near-violation (especially under the fixed fitness function that can generate more tests for a requirement), while a typical desideratum of test suites is to have diverse tests – hence we also check for diversity of the test scenarios.

5.3 Experiment Design

The case study used for the evaluation is Pylot [26], a top-performing ADS with state-of-the-art techniques based on pre-trained DNNs. It has a natural compatibility with CARLA [17], and it is a top submission for CARLA Autonomous Driving Challenge. Pylot is also chosen because of its high adaptability to customized techniques. A test case is characterized by sixteen input variables, describing the road type, the presence of (possibly two-wheeled) vehicles in front, in adjacent or in opposite lane, weather conditions, presence of pedestrians, of trees, of buildings, speed (full list available in the replication package). Tests are generated with input within the boundaries of the simulator’s domain, thus the scenarios are as required by the simulator. As output, the following six requirements are considered, the same used in Haq et al.’s work [30] implementing the baseline strategies: (1) follow the center of the lane; (2) avoid collision with other vehicles; (3) avoid collision with pedestrians; (4) avoid collision with static objects (e.g., traffic signs); (5) abide by traffic rules (e.g., traffic lights); (6) reach the destination in a given time. The metrics used are, respectively: (1) **Distance from the Center of the Lane (DCL)**; (2) **Distance from other Vehicles (DV)**; (3) **Distance from Pedestrian (DP)**; (4) **Distance from Static obstacles (DS)**; (5) **Traffic Rule violated (yes/no outcome) (TR)**; (6) **Distance Traveled (DT)**. Information about these output metrics is retrieved from the simulator. In particular, the distance from the center of the lane is computed as the distance between the center of the vehicle and waypoints that are typically placed in the center of the lane (when multiple lanes are available, the waypoints follow, in case of lane changes, the trajectory that crosses the lane, preventing a safety violation at each change); collisions are detected both by *collision events* and by collected distances (from vehicles, pedestrians, and static objects).

We run three different experiments to answer RQ1, RQ2, and RQ3. The comparison done in RQ4 uses the test suites produced in RQ2 and RQ3 (with the adaptive and fixed fitness functions, respectively). For RQ1, we execute 1,000 random test scenarios to investigate causal models and causal inference performance. For RQ2, each compared technique is run 20 times to get statistically significant results (at significance value $\alpha = .05$). Every execution has a fixed time budget, which is 2 hours, in line with Haq et al. [30]. Moreover, for the techniques that use a database of initial solutions (i.e., CART and SAMOTA-I), we consider two databases of different size: one with exactly the same 39 test scenarios used in the SAMOTA work [30], for a fair comparison, and one with 100 randomly generated test scenarios (called hereafter CART₁₀₀ and SAMOTA-I₁₀₀). The 39 tests used by SAMOTA are obtained by a 4-way combinatorial coverage testing based on all the attributes used to define the test input space to generate diverse test cases.

We have therefore eight techniques (CART₁₀₀, CART, SAMOTA-I₁₀₀, SAMOTA-I, SAMOTA-E, FITEST, MOSA, RANDOM). For RQ3, we run further 20 repetitions per technique, except RANDOM, with the changed fitness function (the *fixed* one), again with a time budget of 2 hours and with both databases of initial solutions. In fact, the RANDOM technique does not distinguish fixed from adaptive function; for it we used the same tests used in RQ2.

Overall, we have 1,000 random generated tests for RQ1, plus 1,909 tests for RQ2 and 1,548 tests for RQ3 resulting from the 40 (20 + 20) runs for every technique. The total number of tests is 4,457 – all made available in the repository – obtained with 676 computing hours. All experiments were run on a virtual machine built on Google Cloud Compute Engine platform⁸. It was configured with Ubuntu 18.04 running on Intel Haswell CPU (4 cores) with NVIDIA Tesla T4 (16 GB) and 16 GB memory.⁹ Further settings and RQ-dependent details are presented in the following section.

6 RESULTS

6.1 RQ1: Usefulness of Causal Models

RQ1 uses a set of $S = 1,000$ test scenarios generated by the RANDOM technique and executed on the simulator. The procedure is as follows: we first use 5% of these tests to train the causal model (training size $z_{tr} = 50$ randomly-sampled tests); then, from the remaining 950 test scenarios, we randomly sample further $z_{ts} = 50$ tests as test set. We consider 50 tests as a “relatively small” test set (i.e., implicitly assuming that, in a realistic environment, the number of tests easily gets to 50), so as to show that causal models can be useful even with few entries.

Let us denote the training and test sets as TrS and TS , respectively. The aim is to compare, for each of the 50 tests in the testing set, the real observed outputs (specifically, the fitness $\phi(s_k)$ of the test) against the prediction done by the causal inference engine querying the causal model: specifically, for each test $s_k = (\mathbf{x}_k; \mathbf{y}_k) \in TS$, the CI engine queries the causal model to predict what is the output under the input \mathbf{x}_k , thus getting an estimate $\hat{\mathbf{y}}_k$. The fitness is computed on the real observed output, \mathbf{y}_k , and on the estimated one, $\hat{\mathbf{y}}_k$ (getting ϕ_k and $\hat{\phi}_k$, which are then compared to each other). In causal inference terms, this means predicting the effect of applying an intervention. To account for randomness, this process is repeated 20 times, with different training and test sets. The sample size for simulating the intervention is set to $\eta = 1,000$, which is the default value in the used library, DoWhy.

To compare the predicted vs the actual output, we use the following two metrics: the percentage **Root Mean Squared Error (%RMSE)** and the **Rank Biased Overlap (RBO)** for rankings comparison. The former is a well-known metric to compare different models’ prediction, corresponding to the RMSE normalized by the RMS value of the predicted value:

$$\%RMSE = \sqrt{\frac{\frac{1}{n} \sum_{k=1}^n (\phi_k - \hat{\phi}_k)^2}{\sum_{k=1}^n \hat{\phi}_k^2}} \cdot 100 \quad (3)$$

where ϕ_k and $\hat{\phi}_k$ are the actual and predicted fitness value of the k -th test, respectively. This metric measures the accuracy of the fitness value prediction – the lower, the better. A tester may be interested just in the ability of the model to distinguish critical tests, rather than in predicting the exact output value. We therefore report the RBO, a well-known metric to compute rankings similarity that, unlike Spearman’s or Kendall’s correlation, weighs the (dis)agreements on the top positions more than the ones at the bottom. The $t = 50$ tests are ranked by their actual fitness from more to less critical ones (list T), and by their predicted fitness (list P). The extrapolated RBO bounded at t is [94]:

$$RBO(A, P, q, t) = A_t \cdot q^t + \frac{1-q}{q} \sum_{d=1}^t A_d \cdot q^d \quad (4)$$

⁸<https://cloud.google.com/compute>

⁹Google LLC, 2020. G Suite, Available at: <https://gsuite.google.com>

Table 1. RQ1 - CSD Algorithms Configuration

Algorithm	Parameters and values
FGES	scoreId = cg-bic-score, dataType = mixed, numCategoriesToDiscretize = 7, maxDegree = 3, faithfulnessAssumed = True, numberResampling = 5, resamplingEnsemble = 1, addOriginalDataset = True
PC	testId = fisher-z-test, fasRule = 2, depth = 2, conflictRule = 1, concurrentFAS = True, useMaxPOrientationHeuristic = True
GFCI	testId = cg-lr-test, scoreId = cg-bic-score, dataType = mixed, numCategoriesToDiscretize = 7, maxDegree = 3, maxPathLength = -1, completeRuleSetUsed = False, faithfulnessAssumed = True
FCI	testId = fisher-z-test, depth = -1, maxPathLength = -1, completeRuleSetUsed = False
RFCI	testId = cg-lr-test, dataType = mixed, numCategoriesToDiscretize = 7, depth = -1, maxPathLength = -1, discretize = False, completeRuleSetUsed = False, numberResampling = 5, resamplingEnsemble = 1, addOriginalDataset = True

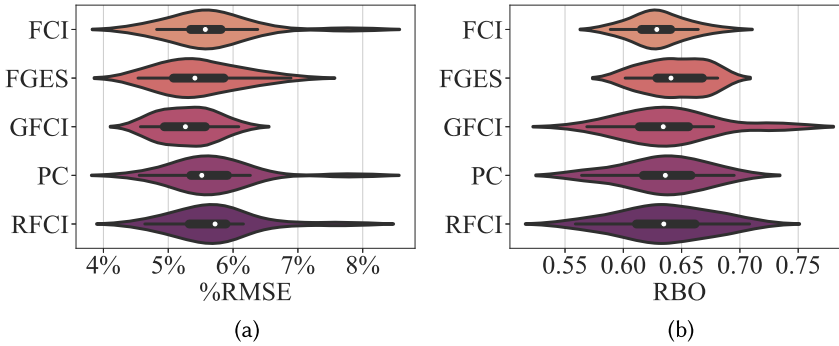


Fig. 2. RQ1 - %RMSE and RBO of the 5 CSD algorithms.

where: $A_t = |T_{1:t} \cap P_{1:t}|/t$ is the proportion of the overlap of the $T_{1:t}$ and $P_{1:t}$ lists (with elements from position 1 to t) of the ranking to be examined; q is a parameter in $(0, 1)$ that determines how steep the decline in weights is, given to the positions in the list – a smaller q gives more weight to the overlaps in top positions. We use $q = 0.98$, giving the $t = 50$ ranks a weight of 86% [94]. In order to build the causal model with the $z_{tr} = 50$ randomly sample tests, we use five different CSD algorithms. The used algorithms are: PC, FGES, FCI, GFCI, RFCI. Their configuration (Table 1) is the default configuration in Tetrad [69] (and pycausal), used in various studies on CSD [96], [101].

The violin plots in Figure 2 show the distribution of %RMSE and RBO. The %RMSE plot indicates that the prediction error done by querying the causal model is between 5% and 6%, regardless the adopted CSD algorithm. The good result is confirmed by the RBO metric, which is approximately between 0.6 and 0.7: this roughly means the two lists of tests (made by ranking the real observed vs predicted fitness) have 60%-70% of their ranks in common [94]. Also in this case, the CSD algorithm seems to not have an impact. To confirm this statistically, we run the Friedman test [23], a non-parametric hypothesis test for Analysis of Variance, which assesses if there is at least one technique that significantly differs from others. For both %RMSE and RBO the null hypothesis of no significant difference between CSD algorithms cannot be rejected (p -value equals 0.9655 for

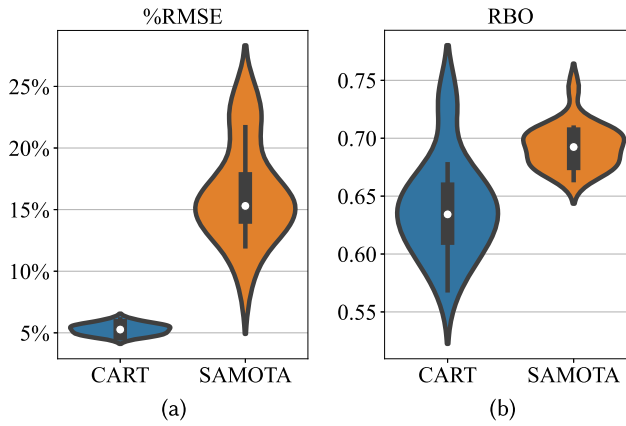


Fig. 3. RQ1.1 - %RMSE and RBO: CART causal models and SAMOTA ML-based models.

%RMSE and 0.2177 for RBO). Thus, the CSD algorithms turned out to be statistically equivalent, any of them can be adopted in CART. In the next RQs, we configure CART with GFCI, since it has slightly smaller %RMSE and similar RBO to the others.

RQ1 answer

Causal inference is a useful tool for engineers to predict the effect of hypothetical inputs on target outputs; the observed error in the case study ranges from 5% to 6%.

The model predictions are also useful to rank the highest-fitness test first (e.g., for prioritizing tests), as the ranking similarity is between 0.6 and 0.7. This can enable the design of new testing techniques based on causal inference.

The CSD algorithm had no significant impact on the performance of the prediction, as both the prediction error and ranking on the give models built by the algorithms are statistically equivalent.

6.1.1 Comparing CART causal models against SAMOTA ML-based models. In the following, we compare the prediction made by causal models vs the prediction made by ML-based surrogate models used by SAMOTA. For this comparison, we consider the same configuration used for the CSD algorithms, training SAMOTA from scratch. The CSD algorithm used is GFCI, the one we used in CART. Figure 3 reports the violin plots. The prediction made by using the causal model via interventions has considerably better (i.e., smaller) %RMSE (5% vs 15%), and with much smaller variance ($1.8 \text{ E-}05$ vs $1.5 \text{ E-}03$), meaning that it yields estimates of the expected test output (hence of the fitness) much closer to the real values and more stable. As a consequence, SAMOTA is expected to generate worse tests (in terms of fitness). The next RQs will investigate this hypothesis. Interestingly, we note that SAMOTA has a slightly better (i.e., bigger) RBO (about 0.68 vs 0.64). This means that SAMOTA surrogates could rank a set of tests slightly better, even though their predicted fitness is far from the real one; this can still be useful for prioritizing existing tests.

6.2 RQ2: Coverage of safety requirements

For RQ2, we run all the testing techniques 20 times for 2 hours. For SAMOTA and CART, which use an initial database D , we consider $|D| = 39$ and $|D| = 100$ test scenarios. CART uses GFCI for causal structure discovery: in the inference step, we keep the sample size for simulating the intervention as $\eta = 1,000$. The comparison is in terms of *coverage*, namely the proportion of safety

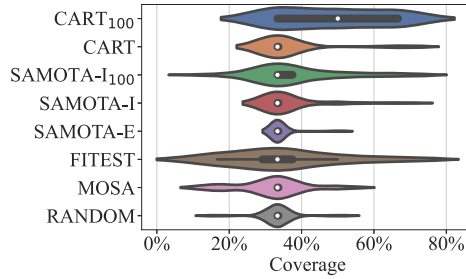


Fig. 4. RQ2.1 - Coverage effectiveness.

Table 2. RQ2.1 - Pairwise Comparison

vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E	FITEST	MOSA	RANDOM
CART ₁₀₀	7.40E-03	5.51E-02	1.65E-02	1.40E-03	1.00E-03	<1.00E-04	5.00E-04
CART	-	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00
SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00	1.00E+00	1.00E+00	1.00E+00
SAMOTA-I	-	-	-	1.00E+00	1.00E+00	1.00E+00	1.00E+00
SAMOTA-E	-	-	-	-	1.00E+00	1.00E+00	1.00E+00
FITEST	-	-	-	-	-	1.00E+00	1.00E+00
MOSA	-	-	-	-	-	-	1.00E+00

requirements violated, computed as:

$$coverage(S) = \frac{|C(S)|}{|R|} \times 100 \quad (5)$$

where $S = \{s_1, s_2, \dots, s_k\}$ is the testing session executing k tests scenarios, $C(S)$ denotes the set of requirements violated by at least one test case in S , and R is the set of safety requirements under consideration (6 in our case).

To answer RQ2.1 we evaluate the coverage after the 2-hours testing session. Figure 4 shows the distribution of the coverage values over 20 repetitions. Except for CART₁₀₀, all the techniques have a comparable median (white dots) around 33%, namely, all are able to cover 2 out of 6 requirements. None of them covers all requirements. CART and SAMOTA-I cover 4 requirements in 2 and 1 cases respectively, achieving a coverage of 36.67%. With a larger database, CART violates consistently more than 2 requirements: CART₁₀₀ has a median coverage of 50% (33.3% for SAMOTA-I₁₀₀) and violates 4 requirements (i.e., 66.7%) in 6 repetitions (SAMOTA₁₀₀ violates 4 requirements in 2 repetitions).

The Friedman test detects a significant difference for at least one pair (p -value = 2.66E-04). We run the Dunn test [18] for *post hoc* analysis (that protects against the multiple comparison problem) to detect which pair of techniques differ significantly. It confirms that CART₁₀₀ is significantly better than all the others (p -values are: 0.0165, 0.0074, 0.0014, 0.0010, 0.0005, and <0.0001 for CART₁₀₀ against, respectively: SAMOTA-I, CART, SAMOTA-E, FITEST, RANDOM, and MOSA) except against SAMOTA-I₁₀₀ but with a p -value slightly above 0.05 (0.0551). In all the other pairs, differences are not significant – the p -values for all the pairs are in the Table 2.

Table 3 reports, for each technique, the number of repetitions (out of 20) that covered the safety requirement. The two requirements *distance from other vehicles* (DV) and *distance traveled* (DT) are clearly the easiest to cover as all the techniques are able to consistently find the respective violations with at least one scenario per repetition, and, in some cases, even in every repetition (CART₁₀₀, CART₃₉, SAMOTA-I₃₉, SAMOTA-E). On the other hand, *distance from the center of the*

Table 3. RQ2.1 - Number of Repetitions (out of 20) that Violate Requirements

Technique	DCL	DV	DP	DS	DT	TR
CART ₁₀₀	7	20	0	10	20	1
CART ₃₉	2	20	0	2	20	0
SAMOTA-I ₁₀₀	5	19	0	2	20	0
SAMOTA-I ₃₉	3	20	0	1	20	0
SAMOTA-E	0	20	0	0	20	1
FITEST	5	13	0	2	18	0
MOSA	2	15	0	0	19	1
RANDOM	2	18	0	0	20	0

Legend: DCL: distance from the center of the lane, DV: distance from other vehicles, DP: distance from pedestrians, DS: distance from static obstacles, DT: distance traveled, TR: traffic rules.

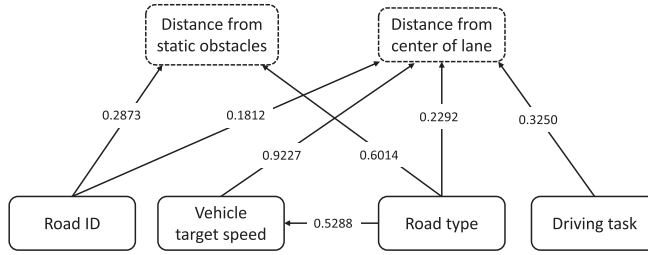


Fig. 5. Excerpt of causal model.

lane (DCL) and distance from static obstacles (DS) are more difficult to cover, with techniques covering them in less (or equal for CART₁₀₀ on the latter requirement) than half of the repetitions; *abide by traffic rules* (TR) is covered only once by CART₁₀₀, SAMOTA-E, and MOSA. *Distance from pedestrian* (DP) turned out to be the hardest to cover in our setting; no technique succeeds in covering it.

The superiority of CART₁₀₀ shows that a bigger dataset has led to higher-quality models and consequently to better test suites. Although a small improvement is noticed also in SAMOTA-I₁₀₀ compared to the other SAMOTA versions, the impact of a bigger archive size is not so pronounced as in CART. This suggests that causal models can benefit more from greater archive size. In Section 6.5 we further investigate this hypothesis.

An important advantage of causal models is their interpretability. They are human-readable and allow a quick identification of inputs causally related to outputs, and consequently to safety violations. In particular, inspecting the results and looking at the model, we can find some input-output relations. For instance, we notice that collisions with other vehicles were mainly caused by the target speed of the ego vehicle, the weather conditions, and the variables specifying the presence of other vehicles (mainly the presence of a vehicle ahead of the ego vehicle in the same lane). When the speed is too high in rainy conditions, the ego vehicle hardly manages to avoid a collision with a vehicle on the same lane that slows down. Similarly, the collisions with pedestrians/static objects were heavily impacted by the roads the ego vehicle was spawned to (e.g., junction, straight road, highway).

More in general, engineers can get different types of insights from a causal model in a relatively simple way compared to ML models. Figure 5 reports an excerpt of causal graph we obtained, along with the weights from the structural equation coefficients representing how much, on average, the effect is expected to change for a change in the cause.

It is possible to get some insights from the model:

- (i) from the graph, we see that Road ID (i.e., the part of the CARLA's map where the scenario takes place), Vehicle target speed, Road type (e.g., "cross road", "left/right turn", "straight") and Driving task ("follow road", "take 1st/2nd/3rd exit") are all causally related to DCL. It is possible to note that there are important differences between the mentioned causes for the DCL effect. In fact, Vehicle target speed is also impacted by Road type. This means that an even strong correlation between Vehicle target speed and DCL can be well due to Road type that can cause both Vehicle target speed and DCL, namely Road type is a confounder. Any prediction based on that correlation without accounting (i.e., controlling for) Road type would fail. With causal inference, one can predict the causal effect of Vehicle target speed, net of confounders and correctly attribute the cause for an observed effect.
- (ii) The graph highlights that the violations of DS and DCL can be caused by multiple variables together, and the underlying equations express the strength of the causal effect of the causes. Also, common causes are also of interest; for instance Road type is a common cause of both DCL and DS.
- (iii) Several other patterns can be inspected. For instance, we can also investigate the chains of causality connecting multiple inputs, assess the direct and indirect effect (i.e., through other input variables) of an input variable on the output. Other patterns have been identified [65], such as the front-door criterion, highlighting situations in which it is not immediate to derive a causal effect of an input of interest to the output.

The second sub-question of RQ2 (RQ2.2) was about the efficiency in covering safety requirements. To answer RQ2.2, we measure the coverage over testing time every 20 minutes. Figure 6 shows that CART-100 achieves by far the highest coverage, with a rapid increase in the first 20 minutes. Also CART is better than the baselines in the first 20 minutes. After 40 minutes, CART and SAMOTA-I₁₀₀ are equivalent and slightly better than the other baselines, and keep this superiority along the whole test duration.

RQ2 answer

CART covers a proportion of safety requirements comparable to the other techniques, but more efficiently, as it covers a higher proportion than SAMOTA-I, SAMOTA-E, MOSA and FITEST in the first 40 minutes. When fed with a larger knowledge (CART₁₀₀, using 100 tests to build the model), it markedly outperforms the other techniques, in terms of both proportion of violated requirements and of efficiency.

6.3 RQ3: Detection of Safety Violations

With RQ3, we aim at evaluating techniques in their ability to expose multiple violations of (possibly the same) safety requirements. Indeed, while RQ2 already shows that CART covers more safety requirements than competitors and earlier, having only one example for a violated safety requirement could be not satisfying for a tester. Multiple diverse violating tests highlight possible different conditions under which a violation occurs, each of which could trigger different faults leading to that violation. This supports the root cause analysis and debugging task.

For instance, let us assume to have a same safety violation of the Distance from the Vehicle (DV) safety requirement exposed by multiple failing tests, as shown in Table 4. The Table shows that the DV violation is present in quite different scenarios (with different values of the "Road ID", i.e., the part of the CARLA's map where the scenario takes place, and "Weather" conditions),

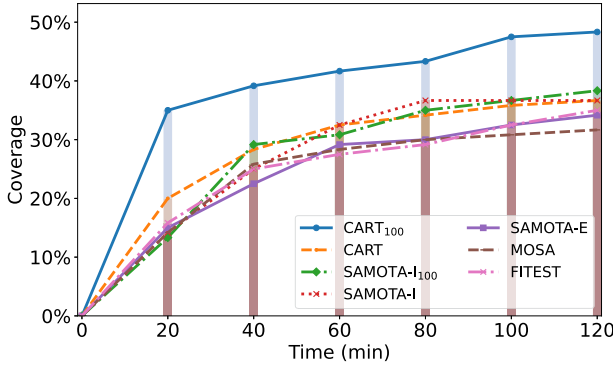


Fig. 6. RQ2.2 - Coverage efficiency.

Table 4. Example of Multiple Safety-Violating Tests for DV Requirement

Road Type	Road ID	Vehicle in Adjacent Lane	Two-wheeled Vehicle in Front	Two-wheeled Vehicle in Adjacent Lane	Two-wheeled Vehicle in Opposite Lane	Weather	Vehicle Target Speed	DV
Cross Road	0	0	1	1	1	Cloudy	40	Violated
Cross Road	1	0	1	0	0	Cloudy	40	Violated
Straight	3	0	1	0	0	Wet/cloudy	30	Violated
Right Turn	2	0	1	1	1	Cloudy	40	Violated
Cross Road	2	0	1	0	1	Cloudy	40	Violated
Cross Road	2	0	1	0	0	Cloudy	40	Violated
Cross Road	2	0	1	0	1	Cloudy	40	Violated
Straight	3	0	1	1	0	Clear	40	Violated
Right Turn	2	0	0	0	0	Wet	40	Violated
Cross Road	1	1	0	1	1	Wet/cloudy	40	Violated
Cross Road	2	0	0	1	0	Wet	40	Not violated
Cross Road	2	0	0	0	0	Wet	40	Not violated

which can be due to different faults, activated by different combinations of (possibly a subset) of the variables. Except for one test (red cells), the failed tests have at least one input among “Two-wheeled Vehicle in Front”, “Two-wheeled Vehicle in Adjacent Lane” and “Two-wheeled Vehicle in Opposite Lane” equal to 1 (yellow cells). This suggests that the presence of a two-wheeled vehicle is a possible fault trigger, hence providing insight into the root cause of the violation - e.g., the autopilot’s DNN for obstacle detection could be inaccurate in detecting two-wheeled vehicles. Having multiple violations (i.e., multiple manifestations of the fault) helps to spot the problem. In addition, the Table shows that the violation is present also when no two-wheeled vehicle is present (the red row). The red-cells test has two input values, “Road Type” and “Road ID” that are the same as another failing test (orange cells). Therefore, this can suggest an additional fault, or at least a more complex activation pattern not depending solely on the “two-wheeled” input variables.

If engineers had just one of the above tests, it would be much harder to derive some conclusion about possible causes of the safety violation. The implications are therefore for the root cause analysis and debugging phase.

To assess the ability of the techniques of finding multiple violations, we run again all the techniques 20 times for 2 hours, but with a *fixed* fitness function. The comparison is in terms of number of violations of safety requirements exposed by a testing session S , denoted as $v(S)$. With $v_i(S)$ we denote the number of violations of the i -th safety requirement.

Results for RQ3.1 (how many violations are found) are in Figure 7. The Friedman test detects a significant difference for at least one pair (p -value = $2.2E-16$). Table 5 reports the p -values for pairwise comparisons, again with the Dunn test. CART₁₀₀ is confirmed to significantly outperform

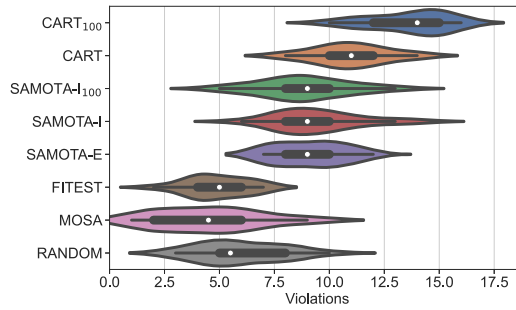


Fig. 7. RQ3.1 - Number of violations.

Table 5. RQ3.1 - Number of Violations, Pairwise Comparison

vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E	FITEST	MOSA	RANDOM
CART ₁₀₀	5.25E-01	4.00E-04	4.90E-03	1.50E-03	<1.00E-04	<1.00E-04	<1.00E-04
CART	–	2.02E-01	8.18E-01	4.56E-01	<1.00E-04	<1.00E-04	<1.00E-04
SAMOTA-I ₁₀₀	–	–	1.00E+00	1.00E+00	5.70E-03	5.40E-03	1.12E-01
SAMOTA-I	–	–	–	1.00E+00	4.00E-04	4.00E-04	1.33E-02
SAMOTA-E	–	–	–	–	1.50E-03	1.40E-03	3.62E-02
FITEST	–	–	–	–	–	1.00E+00	1.00E+00
MOSA	–	–	–	–	–	–	1.00E+00

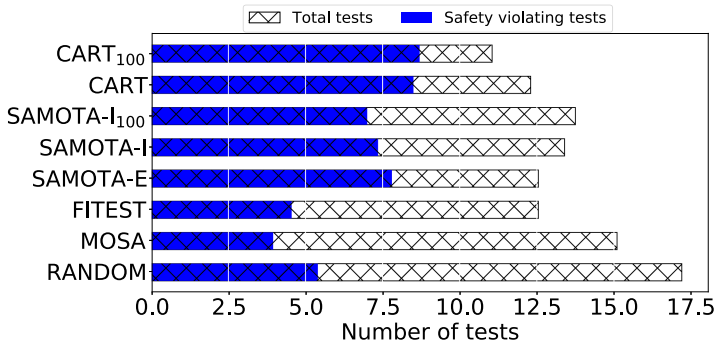


Fig. 8. RQ3.1 - Number of tests and safety-violating tests.

all the algorithms. CART significantly outperforms FITEST, MOSA and RANDOM, while, despite the greater average value, it cannot be claimed to be different from all SAMOTA variants.

Figure 8 shows the average number of tests violating at least one safety requirement, and the average number of executed tests. Since a test can violate more than one requirement, this is different from the number of violations. Besides exposing more violations, CART₁₀₀ generates more safety-violating tests (as well as a higher ratio of safety-violating tests over the executed ones).

With Figure 9, we now analyze the ability of tests to violate more than one requirement, representing very critical scenarios. The Figure shows the number of tests violating 1, 2, and 3 safety requirements together (no test violates more than 3 requirements). CART₁₀₀ with the fixed fitness

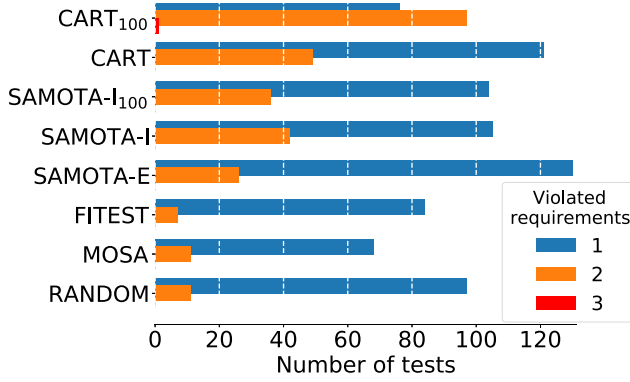


Fig. 9. RQ3.1 - Number of tests with one or more violations.

Table 6. RQ3.1 - Average Number of Violations

Technique	\bar{q}_i						\bar{q}
	DCL	DV	DP	DS	DT	TR	
CART ₁₀₀	0.00	6.10	0.00	0.05	7.50	0.00	13.65
CART	0.05	5.25	0.00	0.00	5.65	0.00	10.95
SAMOTA-I ₁₀₀	0.15	3.20	0.00	0.00	5.45	0.00	8.80
SAMOTA-I	0.05	3.50	0.00	0.00	5.90	0.00	9.45
SAMOTA-E	0.10	2.45	0.00	0.00	6.55	0.00	9.10
FITEST	0.15	1.70	0.00	0.00	3.05	0.00	4.90
MOSA	0.25	2.10	0.00	0.00	2.15	0.00	4.50
RANDOM	0.10	2.65	0.00	0.00	3.20	0.00	5.25

Legend: DCL: distance from the center of the lane, DV: distance from other vehicles, DP: distance from pedestrians, DS: distance from static obstacles, DT: distance traveled, TR: traffic rules.

function is the only technique generating more tests that violate two requirements than those violating a single requirement. It also finds a test violating 3 requirements together (DV, DS, TR). CART is the second technique generating tests violating two requirements, followed by SAMOTA-I and SAMOTA-I₁₀₀.

Table 6 reports the average violations of each requirement. The *distance from pedestrians* (DP) and *traffic rules* (TR) requirements are violated by no technique in this fixed-fitness configuration. *Distance traveled* (DT) and *distance from other vehicles* (DV) are the most frequently violated, followed by *distance from the center of lane* (DCL), which is however never violated by CART₁₀₀. Except CART₁₀₀, no other technique violates *distance from static objects* (DS). It is worth noting that, while the adaptive function (used in RQ2) tends, when a requirement is already covered, to target other requirements, the fixed function tends to violate more often some specific, more failure-prone, requirements (e.g., many violations to DV and DT are generated). For instance, in the adaptive case, CART₁₀₀ was able to cover 5 different requirements in 6 out of 20 repetitions, which never happened in this configuration, but of course detecting a lower average number of total violations (9.4 vs 13.65).

Finally, Figure 10 plots the efficiency comparison (RQ3.2). It clearly shows that both CART₁₀₀ and CART detect more safety violations since the beginning, thus supporting the early identification of critical scenarios.

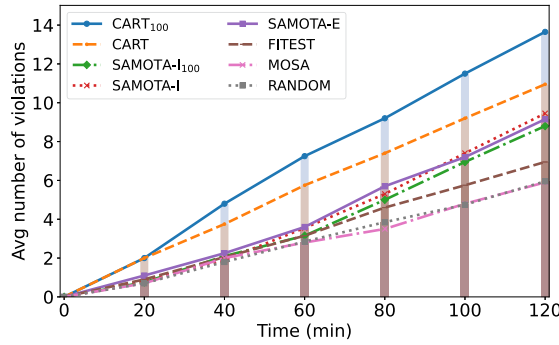


Fig. 10. RQ3.2 - Efficiency.

RQ3 answer

CART₁₀₀ and CART detect the greatest number of violations after the entire testing session, expose more violations earlier, generate more safety-violating tests and with a better ratio of safety-violating tests over total tests.

6.4 RQ4: Fitness and Diversity

RQ4 investigates the *fitness* values and *diversity* of the generated tests. This comparison is done on the test suites produced in RQ2 and RQ3 (with the adaptive and fixed fitness functions, respectively) by the best-performing techniques, which are: CART₁₀₀, CART, SAMOTA-I₁₀₀, SAMOTA-I and SAMOTA-E.

6.4.1 Fitness Evaluation. To evaluate the fitness of tests, we consider the values of the output variables of each test (namely: DT, DP, DV, DS, DCL). We exclude the TR output, since it is not a continuous but a binary variable (all other 5 are distances).

Figures 11 and 12 report the (min-max) normalized values for each output variable, averaged over all the test scenarios generated in the 20 repetitions, split in two groups: failing (i.e., safety-violating) and non-failing tests. For all of them but DCL, the smaller the better; for DCL, we report (1-DCL). The plots show that CART and CART₁₀₀ achieve better values for almost all the 10 output variables (5 for the *adaptive* fitness function, RQ2, and 5 for the *fixed* one, RQ3), except for DT, in both the datasets. The results are confirmed for both groups, with, expectedly, a more pronounced advantage of CART in failing tests.

Considering the whole test set, the Friedman hypothesis test rejects the null hypothesis of no difference between the techniques (p -value < 0.05) in all the cases but two: DT and DV in the *adaptive* case (p -values: 0.949, 0.9578, respectively). Table 7 reports the p -values for the pairwise comparison with the Dunn test for the *adaptive* case. The comparison highlights that CART and CART₁₀₀ are significantly better (p -value < 0.05) than all SAMOTA variants for three outputs (DP, DCL, DS). While the difference is confirmed to be not significant for DT and DV (we do not report p -values for DV as they were all $1.00E + 00$).

In the *fixed* case, the Friedman test rejects the null hypothesis of no difference between in all the cases. Table 8 reports the p -values for the pairwise comparison. In this case, CART₁₀₀ and CART are significantly better than SAMOTA in the DP, DV and DS output (with the exception of SAMOTA-I vs CART₁₀₀ for DS, p -value = 0.1435). For DCL, the significant differences are CART vs

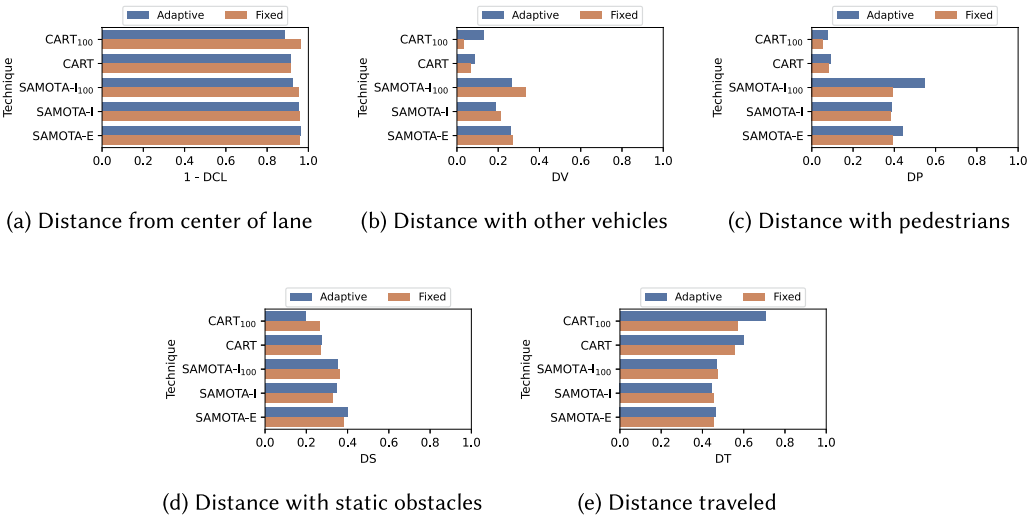


Fig. 11. RQ4 - test suite fitness per requirement (failing tests).

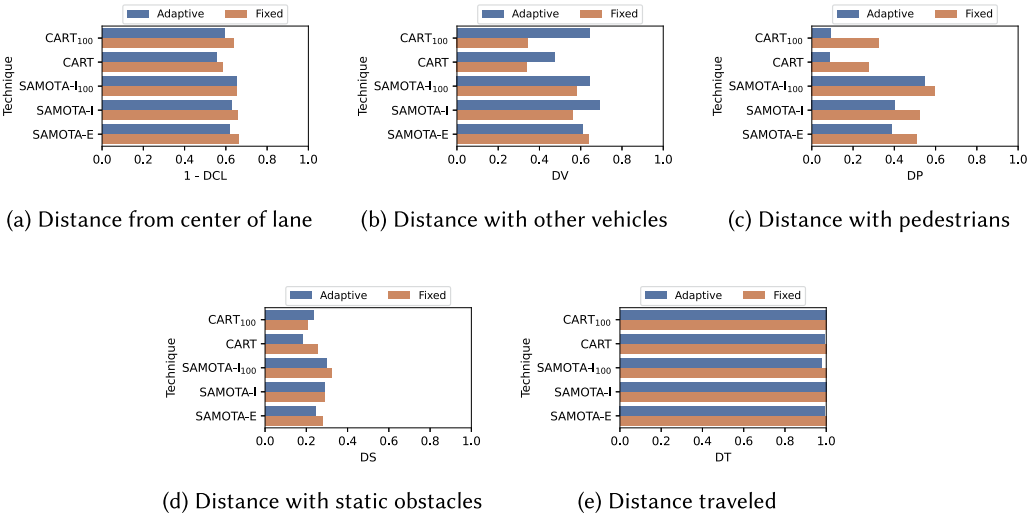


Fig. 12. RQ4 - test suite fitness per requirement (not failing tests).

SAMOTA-I and CART vs SAMOTA-E (the others, although in favour of CART, are not significant). For DT, the only significant difference is SAMOTA-I₁₀₀ vs CART₁₀₀, in favour of the former.

Overall, the results indicate that CART₁₀₀ and CART generate tests that push almost all the output variables closer to the thresholds, generating near-violating scenarios, especially with the *fixed* fitness. These could better highlight critical behaviours (e.g., the car stops very close to a pedestrian, which might be not desirable even though it is not a collision), and can make it easier to manually derive violating scenarios by tuning the obtained near-critical ones. Moreover, since CART follows an evolutionary approach, these high-fitness tests would more likely evolve into violating tests with more testing time available for the algorithm.

Table 7. RQ4 - P-Values for Pairwise Comparison

(a) Distance from the Center of the Lane (DCL)					(b) Distance from Pedestrians (DP)				
vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E	vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E
CART ₁₀₀	1.00E+00	1.14E-02	< 1.00E-04	1.00E-04	CART ₁₀₀	4.00E-01	< 1.00E-04	1.68E-02	3.20E-03
CART	-	2.50E-03	< 1.00E-04	< 1.00E-04	CART	-	< 1.00E-04	< 1.00E-04	< 1.00E-04
SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00	SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00
SAMOTA-I	-	-	-	1.00E+00	SAMOTA-I	-	-	-	1.00E+00

(c) Distance from Static obstacles (DS)					(d) Distance Traveled (DT)				
vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E	vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E
CART ₁₀₀	1.00E+00	1.03E-02	5.00E-04	5.00E-04	CART ₁₀₀	1.00E+00	1.00E+00	6.20E-02	2.85E-01
CART	-	1.00E-04	< 1.00E-04	< 1.00E-04	CART	-	1.00E+00	6.20E-02	2.85E-01
SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00	SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00
SAMOTA-I	-	-	-	1.00E+00	SAMOTA-I	-	-	-	1.00E+00

Adaptive fitness.

Table 8. RQ4 - P-Values for Pairwise Comparison

(a) Distance from the Center of the Lane (DCL)					(b) Distance from other Vehicles (DV)				
vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E	vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E
CART ₁₀₀	9.20E-02	1.00E+00	1.00E+00	6.23E-01	CART ₁₀₀	1.00E+00	< 1.00E-04	< 1.00E-04	< 1.00E-04
CART	-	7.40E-02	2.00E-03	< 1.00E-04	CART	-	< 1.00E-04	1.00E-03	1.00E-04
SAMOTA-I ₁₀₀	-	-	1.00E+00	7.34E-01	SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00
SAMOTA-I	-	-	-	1.00E+00	SAMOTA-I	-	-	-	1.00E+00

(c) Distance from Pedestrians (DP)					(d) Distance from Static obstacles (DS)				
vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E	vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E
CART ₁₀₀	1.00E+00	< 1.00E-04	< 1.00E-04	< 1.00E-04	CART ₁₀₀	3.77E-01	4.50E-02	1.43E-01	5.50E-03
CART	-	< 1.00E-04	< 1.00E-04	< 1.00E-04	CART	-	< 1.00E-04	< 1.00E-04	< 1.00E-04
SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00	SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00
SAMOTA-I	-	-	-	1.00E+00	SAMOTA-I	-	-	-	1.00E+00

(e) Distance Traveled (DT)				
vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E
CART ₁₀₀	5.85E-02	3.17E-02	1.00E+00	1.00E+00
CART	-	1.00E+00	3.54E-01	1.41E-01
SAMOTA-I ₁₀₀	-	-	2.16E-01	8.08E-02
SAMOTA-I	-	-	-	1.00E+00

Fixed fitness.

6.4.2 *Diversity evaluation.* As for diversity of the test suites, we used the **Test Set Diameter (TSD)**, a well-known metric for black-box test suite diversity used in test prioritization [21, 33, 54], computed on the 20 repetitions for both test suites (generated in RQ2 and RQ3). The TSD is computed via the **Normalized Compression Distance (NCD₁)**:

$$NCD_1(X) = \frac{C(X) - \min_{x \in X} \{C(x)\}}{\max_{x \in X} \{C(X \setminus \{x\})\}} \quad (6)$$

$$TSD(X) = \max\{NCD_1(X), \max_{Y \subset X} \{TSD(Y)\}\} \quad (7)$$

where $x \in X$ denotes a test, X the whole test suite, and $C(X)$ is the length of compressing X with the *bzip2* compression program [21].

Figure 13 shows that CART with the adaptive fitness function produces test suites with more diverse inputs and outputs. For the inputs (Figure 13(a)), the difference is however significant only for CART vs SAMOTA-I₁₀₀ (p -value: 0.0157), while for the outputs (Figure 13(b)) the p -values are: 0.0013, <0.0001, and 0.0006 vs, respectively, SAMOTA-I, SAMOTA-I₁₀₀, SAMOTA-E). The second

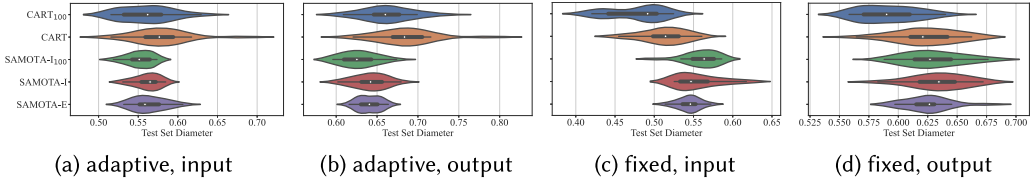


Fig. 13. RQ4 - test suite diversity.

Table 9. RQ4 - P-Values for Pairwise Comparison

(a) TSD on input					(b) TSD on output				
vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E	vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E
CART ₁₀₀	5.54E-01	1.00E+00	1.00E+00	1.00E+00	CART ₁₀₀	9.01E-01	1.10E-03	3.38E-01	2.01E-01
CART	-	1.57E-02	1.00E+00	1.00E+00	CART	-	<1.00E-04	1.30E-03	6.00E-04
SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00	SAMOTA-I ₁₀₀	-	-	8.11E-01	1.00E+00
SAMOTA-I	-	-	-	1.00E+00	SAMOTA-I	-	-	-	1.00E+00

Adaptive fitness.

Table 10. RQ4 - P-Values for Pairwise Comparison

(a) TSD on input					(b) TSD on output				
vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E	vs	CART	SAMOTA-I ₁₀₀	SAMOTA-I	SAMOTA-E
CART ₁₀₀	4.34E-01	<1.00E-04	<1.00E-04	<1.00E-04	CART ₁₀₀	9.70E-03	5.00E-04	<1.00E-04	4.00E-03
CART	-	<1.00E-04	2.60E-03	2.62E-02	CART	-	1.00E+00	1.00E+00	1.00E+00
SAMOTA-I ₁₀₀	-	-	1.00E+00	9.21E-01	SAMOTA-I ₁₀₀	-	-	1.00E+00	1.00E+00
SAMOTA-I	-	-	-	1.00E+00	SAMOTA-I	-	-	-	1.00E+00

Fixed fitness.

one is CART₁₀₀. With the *fixed* fitness, the test suites, especially those by CART₁₀₀, have (statistically) worse diversity than SAMOTA in both inputs and outputs (Figure 13(c), 13(d)), while CART is statistically equivalent to SAMOTA for outputs and worse for inputs. Tables 9 and 10 report the p-values for all the pairwise comparisons.

The observed smaller diversity in the fixed fitness function is a side effect: the focus of the fixed fitness function on exposing critical tests is exploited by the CART's causal models more than the SAMOTA's surrogate models to learn input-output relations, giving higher fitness but also more similar tests. Improving diversity with the fixed function too is a future step we are interested in investigating.

RQ4 answer

With the adaptive fitness function, CART and CART₁₀₀ produce test suites that have higher fitness values for 4 out of 5 requirements and have a better diversity than the compared algorithms. With the fixed fitness function, they produce test suites with (even) higher fitness values for 4 out of 5 requirements, but with worse diversity.

6.5 Additional Remarks

Like SAMOTA-I, CART requires an initial database of executed scenarios to build the model, which can be taken from historical driving data or from past tests data. When unavailable, the database generation entails a start-up cost. This is, however, paid off not only by a more effective and efficient testing, but also by yielding a causal model characterising the system under test – a useful

Table 11. Coverage of Safety Requirements of CART with Different Initial Database Size

Technique	Mean	Std.Dev
CART ₁₅₀	0.48	0.06
CART ₁₀₀	0.48	0.14
CART	0.37	0.10

asset to support other quality-related activities besides testing. Also, with a causal model, it is relatively easy to embed domain knowledge by engineers, which would further boost performance. Specifically, adding knowledge requires specifying known (or forbidden) cause-effect edges in the graph. In the evaluation, we have left this option out for the sake of full automation, but also to avoid human intervention that could have biased the result (in favour of CART). Indeed, in a real setting known cause-effect relations would be added to the model so as to fine-tune CART for a certain context.

Related to this, the results suggest that having a larger database could be beneficial. To assess this hypothesis, we executed an additional experiment with CART, setting an initial dataset of 150 random tests, 20 repetitions. Results are reported in Table 11. They show that both CART₁₀₀ and CART₁₅₀ improve over CART with 39 initial tests. On the other hand, we notice no significant difference between CART₁₅₀ and CART₁₀₀, except a better standard deviation for the former – hence more stable results. We have inspected the causal graph generated by the causal structure discovery algorithms in the two cases, and they turned out to be the same. We conclude that 100 test cases were enough in this scenario to learn a close approximation of the “true” causal model, and further tests add little.

Finally, the CI queries to derive a test impact the test generation time. The impact is however negligible, as the average generation time for a test case over all the CART (and CART₁₀₀) executions turned out to be 7.25 seconds per test, which is 1.3% of the average test execution time (that is ≈ 9 minutes). As shown by the efficiency plots (Figure 6 and 10), this has not undermined the gain of our solution. If needed, this time can be reduced by implementing policies for sampling values for the intervention (i.e., for the CI queries). The literature proposes some strategies in this regard. For instance, an option is to select the intervention maximizing the information gained (i.e., minimize the uncertainty) about the true graph – hence intervene to better learn the “true” graph [82]. A different path can be to set an intervention trying to maximize/minimize the desired test objective(s) (e.g., increase coverage or fitness), or maximize diversity. These criteria can be implemented in several ways, e.g., via probabilistic sampling (e.g., (non-)uniform random sampling), search- or learning-based techniques, or adaptive strategies (using previous selections to drive the next ones).

7 THREATS TO VALIDITY

The time budget for each technique is set at 120 minutes (according to [30]). We decided this both to limit the experimentation (we spent almost 700 computing hours) and to fairly compare with previous experiments. Giving more or less time to each technique could lead to different results. Even if we consider many input variables (16 input variables) that constitute a large input space, it can be worth considering more variables in a real-world system. This could affect the performance of causal discovery, for which scalability is an open challenge. However, CSD algorithms have shown tractable running time on datasets with up to 500 variables [67], which is indeed a remarkable upper bound for setting up effective testing sessions.

The use of a simulator can be a further threat to validity. For instance, it may rarely generate scenarios physically impossible. However, our manual inspection of the executed scenarios did not reveal any such case. Also, the use of simulators is widespread in ADS, as testing in the real world is costly, and this has pushed toward the development of high-fidelity simulators like CARLA. Indeed, challenges are still open about simulators' representativeness [83], but they are increasingly trustable and are often an obliged path [31], [32].

The initial database of tests provided to CART and SAMOTA-I can affect performance; we used two databases, one provided by SAMOTA's authors and one of the randomly generated tests, to mitigate the impact. Replicating with different databases would reinforce the results. Also, the results are obtained under default setting of Tetrad and DoWhy; a fine-tuning of CSD algorithms and CI estimation methods is left to future work.

Despite our efforts to ensure defect-free code – the CART prototype and the infrastructure code interfacing to CARLA, partially borrowed from [30] – their presence cannot be excluded. Last but not least, experiments are on one specific ADS and simulator. Although Pylot and CARLA are representative widely-used choices [19, 56], replicating the experiments with other subjects is needed to improve external validity. Also, CART is evaluated on the same six safety requirements used by SAMOTA; however, CART is not tied to these specific requirements, it can be used with any other safety requirement of interest, provided that their violation can be checked (i.e., there is an oracle).

8 CONCLUSION

We presented CART, a new ADS testing strategy based on causal reasoning for intelligent tests space exploration and efficient tests generation. Results show that capturing and then exploiting the causal relations between test input and output can allow spotting more failure-exposing tests. Several alternatives can be explored to improve the obtained results. For instance, one option is to act on the input variable selection process to decide the intervention on the causal model. Currently, we select the input variable with the greatest out-degree toward the output variables; we plan to experimentally compare other criteria, such as an adaptive selection (e.g., counting the edges toward the “uncovered” output variables only), a selection accounting for the strength of the relation (measured as the increment of variance on the effect variable Y obtained by removing the arrow $X \rightarrow Y$ or as effect variation given the cause variation), or accounting for the confidence or any uncertainty reduction criteria such as entropy.

More broadly, the results underscore the effectiveness of causal reasoning as an alternative for ADS software testing, paving the ground for a broader exploration and exploitation of its capacity to emulate human reasoning in test engineering tasks. Therefore, in addition to consolidating CART for the autonomous driving domain, we plan to extend its application to other areas of autonomous systems, such as unmanned aerial systems or autonomous underwater vehicles. Also, we plan to explore causal reasoning for other quality-related tasks. In fact, the causal model given as output is a valuable asset that can support the inspection of test results and the consequent debugging/root cause analysis of observed failures. For instance, the use of counterfactual inference has been proven to be useful for root cause analysis in Microservice architectures [97]. Moreover, the interventions on the model can be useful in the design phase, e.g., by *what happens if* queries aimed to evaluate design alternatives, to fine-tune parameters, or for capacity planning.

DATA AVAILABILITY

For the sake of Open Science, we provide all artefacts (code of CART, the baselines, the experimental code for replication, results, and the set of 4,457 test scenarios). Available at: <https://doi.org/10.6084/m9.figshare.21937121>

REFERENCES

- [1] 2022. Self-Driving Cars Market Size, Share, Growth, Report 2022-2030. <https://www.precedenceresearch.com/self-driving-cars-market>. n. 1779, Precedence Research, Accessed: 2022-08-31.
- [2] Z. Bai, G. Shu, and A. Podgurski. 2015. NUMFL: Localizing faults in numerical software using a value-based causal model. In *IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 1–10. <https://doi.org/10.1109/ICST.2015.7102597>
- [3] R. Ben Abdesslem, S. Nejati, L. C. Briand, and T. Stifter. 2016. Testing advanced driver assistance systems using multi-objective search and neural networks. In *31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 63–74.
- [4] R. Ben Abdesslem, S. Nejati, L. C. Briand, and T. Stifter. 2018. Testing vision-based control systems using learnable evolutionary algorithms. In *40th International Conference on Software Engineering (ICSE)*. ACM, 1016–1026.
- [5] R. Ben Abdesslem, A. Panichella, S. Nejati, L. C. Briand, and T. Stifter. 2018. Testing autonomous cars for feature interaction failures using many-objective search. In *33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*. ACM, 143–154.
- [6] P. Blöbaum, P. Götz, K. Budhathoki, A. A. Mastakouri, and D. Janzing. 2022. DoWhy-GCM: An extension of DoWhy for causal inference in graphical causal models. <https://doi.org/10.48550/ARXIV.2206.06821>
- [7] M. Caliendo and S. Kopeinig. 2008. SOME practical guidance for the implementation of propensity score matching. *Journal of Economic Surveys* 22, 1 (2008), 31–72. <https://doi.org/10.1111/j.1467-6419.2007.00527.x> arXiv:<https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-6419.2007.00527.x>
- [8] A. Calò, P. Arcaini, S. Ali, F. Hauer, and F. Ishikawa. 2020. Generating avoidable collision scenarios for testing autonomous driving systems. In *IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 375–386.
- [9] H. Chen, T. Harinen, J. Lee, M. Yung, and Z. Zhao. 2020. CausalML: Python package for causal machine learning. *CoRR* abs/2002.11631 (2020). arXiv:2002.11631
- [10] J. Chen, Z. Wu, Z. Wang, H. You, L. Zhang, and M. Yan. 2020. Practical accuracy estimation for efficient deep neural network testing. *ACM Trans. Softw. Eng. Methodol.* 29, 4, Article 30 (Oct. 2020), 35 pages.<https://doi.org/10.1145/3394112>
- [11] D. M. Chickering. 2002. Learning equivalence classes of Bayesian-network structures. *J. Mach. Learn. Res.* 2 (Mar. 2002), 445–498.<https://doi.org/10.1162/153244302760200696>
- [12] A. G. Clark, M. Foster, N. Walkinshaw, and R. M. Hierons. 2022. Metamorphic Testing with Causal Graphs. © 2023 IEEE.
- [13] D. Colombo, M. H. Maathuis, M. Kalisch, and T. S. Richardson. 2012. Learning high-dimensional directed acyclic graphs with latent and selection variables. *The Annals of Statistics* 40, 1 (2012), 294–321. <https://doi.org/10.1214/11-AOS940>
- [14] K. Deb. 2001. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley & Sons, Inc., USA.
- [15] K. Deb. 2014. *Multi-Objective Optimization*. Springer US, Boston, MA, 403–449.https://doi.org/10.1007/978-1-4614-6940-7_15
- [16] A. Dieng, Y. Liu, S. Roy, C. Rudin, and A. Volfvsky. 2019. Interpretable almost-exact matching for causal inference. In *Proceedings of the Twenty-Second International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research, Vol. 89)*, K. Chaudhuri and M. Sugiyama (Eds.). PMLR, 2445–2453.
- [17] A. Dosovitskiy, G. Ros, F. Codevilla, A. Lopez, and V. Koltun. 2017. CARLA: An open urban driving simulator. In *1st Annual Conference on Robot Learning*, Vol. 78. Proceedings of Machine Learning Research (PMLR), 1–16.
- [18] O. J. Dunn. 1964. Multiple comparisons using rank sums. *Technometrics* 6, 3 (1964), 241–252. <https://doi.org/10.1080/00401706.1964.10490181>
- [19] D. Dworak, F. Ciepela, J. Derbisz, I. Izzat, M. Komorkiewicz, and M. Wójcik. 2019. Performance of LiDAR object detection deep learning architectures based on artificially generated point cloud data from CARLA simulator. In *24th International Conference on Methods and Models in Automation and Robotics (MMAR)*. IEEE, 600–605. <https://doi.org/10.1109/MMAR.2019.8864642>
- [20] F. Eberhardt. 2017. Introduction to the foundations of causal discovery. *International Journal of Data Science and Analytics* 3, 2 (2017), 81–91.<https://doi.org/10.1007/s41060-016-0038-6>
- [21] R. Feldt, S. Poulding, D. Clark, and S. Yoo. 2016. Test set diameter: Quantifying the diversity of sets of test cases. In *2016 IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 223–233. <https://doi.org/10.1109/ICST.2016.33>
- [22] C. Fong, M. Ratkovic, K. Imai, C. Hazlett, X. Yang, S. Peng, and I. Lee. 2022. CBPS: Covariate balancing propensity score. *R Package Version 0.23*. *R Foundation for Statistical Computing*. (2022).
- [23] M. Friedman. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *J. Amer. Statist. Assoc.* 32, 200 (1937), 675–701. <https://doi.org/10.1080/01621459.1937.10503522>

- [24] A. Gambi, M. Mueller, and G. Fraser. 2019. Automatically testing self-driving cars with search-based procedural content generation. In *28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA)*. ACM, 318–328. <https://doi.org/10.1145/3293882.3330566>
- [25] C. Glymour, K. Zhang, and P. Spirtes. 2019. Review of causal discovery methods based on graphical models. *Frontiers in Genetics* 10 (06 2019). <https://doi.org/10.3389/fgene.2019.00524>
- [26] I. Gog, S. Kalra, P. Schafhalter, M. A. Wright, J. E. Gonzalez, and I. Stoica. 2021. Pylot: A modular platform for exploring latency-accuracy tradeoffs in autonomous vehicles. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 8806–8813.
- [27] R. Gore and P. F. Reynolds. 2012. Reducing confounding bias in predicate-level statistical debugging metrics. In *34th International Conference on Software Engineering (ICSE)*. IEEE, 463–473. <https://doi.org/10.1109/ICSE.2012.6227169>
- [28] A. Guerriero, R. Pietrantuono, and S. Russo. 2021. Operation is the hardest teacher: Estimating DNN accuracy looking for mispredictions. In *43rd International Conference on Software Engineering (ICSE)*. IEEE, 348–358.
- [29] R. Guo, L. Cheng, J. Li, P. R. Hahn, and H. Liu. 2020. A survey of learning causality with data: Problems and methods. *ACM Comput. Surv.* 53, 4, Article 75 (2020), 37 pages. <https://doi.org/10.1145/3397269>
- [30] F. U. Haq, D. Shin, and L. Briand. 2022. Efficient online testing for DNN-enabled systems using surrogate-assisted and many-objective optimization. In *44th International Conference on Software Engineering (ICSE)*. ACM, 811–822.
- [31] F. U. Haq, D. Shin, S. Nejati, and L. Briand. 2021. Can offline testing of deep neural networks replace their online testing? A case study of automated driving systems. *Empirical Softw. Eng.* 26, 5 (Sep. 2021), 30 pages. <https://doi.org/10.1007/s10664-021-09982-4>
- [32] F. U. Haq, D. Shin, S. Nejati, and L. C. Briand. 2020. Comparing offline and online testing of deep neural networks: An autonomous car case study. In *IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*. IEEE, 85–95. <https://doi.org/10.1109/ICST46399.2020.00019>
- [33] C. Henard, M. Papadakis, M. Harman, Y. Jia, and Y. Le Traon. 2016. Comparing white-box and black-box test prioritization. In *38th International Conference on Software Engineering (ICSE)* (Austin, Texas). ACM, 523–534. <https://doi.org/10.1145/2884781.2884791>
- [34] D. Ho, K. Imai, G. King, and E. A. Stuart. 2011. MatchIt: Nonparametric preprocessing for parametric causal inference. *Journal of Statistical Software* 42, 8 (2011), 1–28. <https://doi.org/10.18637/jss.v042.i08>
- [35] Yimin Huang and Marco Valtorta. 2012. Pearl’s Calculus of Intervention is Complete. <https://doi.org/10.48550/ARXIV.1206.6831>
- [36] G. W. Imbens. 2020. Potential outcome and directed acyclic graph approaches to causality: Relevance for empirical practice in economics. *Journal of Economic Literature* 58, 4 (2020), 1129–79. <https://doi.org/10.1257/jel.20191597>
- [37] M. S. Iqbal, R. Krishna, M. A. Javidian, B. Ray, and P. Jamshidi. 2022. Unicorn: Reasoning about configurable system performance through the lens of causality. In *17th European Conference on Computer Systems (Rennes, France) (EuroSys ’22)*. ACM, New York, NY, USA, 199–217. <https://doi.org/10.1145/3492321.3519575>
- [38] Y. Jin. 2011. Surrogate-assisted evolutionary computation: Recent advances and future challenges. *Swarm and Evolutionary Computation* 1, 2 (2011), 61–70. <https://doi.org/10.1016/j.swevo.2011.05.001>
- [39] M. Kalisch, M. Mächler, D. Colombo, M. H. Maathuis, and P. Bühlmann. 2012. Causal inference using graphical models with the R package pcalg. *Journal of Statistical Software* 47, 11 (2012), 1–26. <https://doi.org/10.18637/jss.v047.i11>
- [40] J. Kim, R. Feldt, and S. Yoo. 2019. Guiding deep learning system testing using surprise adequacy. In *41st International Conference on Software Engineering (Montreal, Quebec, Canada) (ICSE)*. IEEE, 1039–1049. <https://doi.org/10.1109/ICSE.2019.00108>
- [41] M. Klischat and M. Althoff. 2019. Generating critical test scenarios for automated vehicles with evolutionary algorithms. In *IEEE Intelligent Vehicles Symposium (IV)*. IEEE, 2352–2358.
- [42] Z. Kong and C. Liu. 2019. Generating adversarial fragments with adversarial networks for physical-world implementation. *CoRR* abs/1907.04449 (2019). arXiv:1907.04449 <http://arxiv.org/abs/1907.04449>
- [43] Y. Küçük, T. A. D. Henderson, and A. Podgurski. 2021. Improving fault localization by integrating value and predicate based causal inference techniques. In *43rd International Conference on Software Engineering (ICSE) (Madrid, Spain) (ICSE ’21)*. IEEE, 649–660. <https://doi.org/10.1109/ICSE43902.2021.00066>
- [44] G. Li, Y. Li, S. Jha, T. Tsai, M. Sullivan, S. K. S. Hari, Z. Kalbarczyk, and R. Iyer. 2020. AV-FUZZER: Finding safety violations in autonomous driving systems. In *IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 25–36.
- [45] Z. Li, X. Ma, C. Xu, C. Cao, J. Xu, and J. Lü. 2019. Boosting operational DNN testing efficiency through conditioning. In *Proc. 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. ACM, 499–509.
- [46] Q. Liu, X. Wu, Q. Lin, J. Ji, and K. Wong. 2021. A novel surrogate-assisted evolutionary algorithm with an uncertainty grouping based infill criterion. *Swarm and Evolutionary Computation* 60 (2021), 100787. <https://doi.org/10.1016/j.swevo.2020.100787>

- [47] Y. Luo, X. Zhang, P. Arcaini, Z. Jin, H. Zhao, F. Ishikawa, R. Wu, and T. Xie. 2021. Targeting requirements violations of autonomous driving systems by dynamic evolutionary search. In *36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE/ACM, 279–291. <https://doi.org/10.1109/ASE51524.2021.9678883>
- [48] L. Ma, F. Juefei-Xu, F. Zhang, J. Sun, M. Xue, B. Li, C. Chen, T. Su, L. Li, Y. Liu, J. Zhao, and Y. Wang. 2018. DeepGauge: Multi-granularity testing criteria for deep learning systems. In *33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)* (Montpellier, France). ACM, 120–131. <https://doi.org/10.1145/3238147.3238202>
- [49] L. Ma, F. Zhang, J. Sun, M. Xue, B. Li, F. Juefei-Xu, C. Xie, L. Li, Y. Liu, J. Zhao, and Y. Wang. 2018. DeepMutation: Mutation testing of deep learning systems. In *29th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 100–111.
- [50] L. Ma, F. Zhang, M. Xue, B. Li, Y. Liu, J. Zhao, and Y. Wang. 2018. Combinatorial testing for deep learning systems. *CoRR* abs/1806.07723 (2018). arXiv:1806.07723 <http://arxiv.org/abs/1806.07723>
- [51] R. Majumdar, A. S. Mathur, M. Pirron, L. Stegner, and D. Zufferey. 2019. Paracosm: A language and tool for testing autonomous driving systems. *CoRR* abs/1902.01084 (2019). arXiv:1902.01084 <http://arxiv.org/abs/1902.01084>
- [52] D. Malinsky and D. Danks. 2018. Causal discovery algorithms: A practical guide. *Philosophy Compass* 13, 1 (2018), e12470. [10.1111/phc3.12470](https://doi.org/10.1111/phc3.12470)
- [53] H. Mao and L. Li. 2022. PSW: Propensity score weighting methods for dichotomous treatments. *R package version 1.1-3*. *R Foundation for Statistical Computing* (2022).
- [54] B. Miranda, R. Verdecchia, E. Cruciani, and A. Bertolino. 2018. FAST approaches to scalable similarity-based test case prioritization. In *40th International Conference on Software Engineering (ICSE)*. ACM, 222–232. <https://doi.org/10.1145/3180155.3180210>
- [55] H. Niederreiter. 1992. *Random Number Generation and Quasi-Monte Carlo Methods*. Society for Industrial and Applied Mathematics. <https://doi.org/10.1137/1.9781611970081> arXiv:<https://pubs.siam.org/doi/pdf/10.1137/1.9781611970081>
- [56] D. R. Niranjani, B. C. VinayKarthik, and Mohana. 2021. Deep learning based object detection model for autonomous driving research using CARLA simulator. In *2nd International Conference on Smart Electronics and Communication (ICOSEC)*. IEEE, 1251–1258. <https://doi.org/10.1109/ICOSEC51865.2021.9591747>
- [57] A. R. Nogueira, A. Pugnana, S. Ruggieri, D. Pedreschi, and J. Gama. 2022. Methods and tools for causal discovery and causal inference. *WIREs Data Mining and Knowledge Discovery* 12, 2 (2022), e1449. <https://doi.org/10.1002/widm.1449> arXiv:<https://wires.onlinelibrary.wiley.com/doi/pdf/10.1002/widm.1449>
- [58] J. M. Ogarrio, P. Spirtes, and J. Ramsey. 2016. A hybrid causal search algorithm for latent variable models. In *Proceedings of the Eighth International Conference on Probabilistic Graphical Models (Proceedings of Machine Learning Research, Vol. 52)*, Alessandro Antonucci, Giorgio Corani, and Cassio Polpo Campos (Eds.). PMLR, Lugano, Switzerland, 368–379.
- [59] S. Oh, S. Lee, and S. Yoo. 2021. Effectively sampling higher order mutants using causal effect. In *IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 19–24. <https://doi.org/10.1109/ICSTW52544.2021.00017>
- [60] V. Orlandi, S. Roy, C. Rudin, and A. Volfovsky. 2022. FLAME: Interpretable matching for causal inference. *R package version 2.1.1*. *R Foundation for Statistical Computing*. (2022).
- [61] A. Panichella, F. M. Kifetew, and P. Tonella. 2015. Reformulating branch coverage as a many-objective optimization problem. In *IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 1–10.
- [62] D. Parthasarathy and A. Johansson. 2021. SilGAN: Generating driving maneuvers for scenario-based software-in-the-loop testing. In *IEEE International Conference on Artificial Intelligence Testing (AITest)*. IEEE, 65–72. <https://doi.org/10.1109/AITEST52744.2021.00022>
- [63] K. Patel and R. M. Hierons. 2018. A mapping study on testing non-testable systems. *Software Quality Journal* 26, 4 (Dec. 2018), 1373–1413. <https://doi.org/10.1007/s11219-017-9392-4>
- [64] J. Pearl. 2009. *Causality: Models, Reasoning and Inference* (2nd ed.). Cambridge University Press, USA.
- [65] J. Pearl and D. Mackenzie. 2018. *The Book of Why: The New Science of Cause and Effect* (1st ed.). Basic Books, Inc., USA.
- [66] K. Pei, Y. Cao, J. Yang, and S. Jana. 2019. DeepXplore: Automated whitebox testing of deep learning systems. *Commun. ACM* 62, 11 (2019), 137–145. <https://doi.org/10.1145/3361566>
- [67] V. K. Raghunathan, J. D. Ramsey, A. Morris, D. V. Manatakis, P. Sprites, P. K. Chrysanthis, C. Glymour, and P. V. Benos. 2018. Comparison of strategies for scalable causal discovery of latent variable models from mixed data. *International Journal of Data Science and Analytics* 6, 1 (01 Aug. 2018), 33–45. <https://doi.org/10.1007/s41060-018-0104-3>
- [68] J. Ramsey, M. Glymour, R. Sanchez-Romero, and C. Glymour. 2017. A million variables and more: The Fast Greedy Equivalence Search algorithm for learning high-dimensional graphical causal models, with an application to functional magnetic resonance images. *International Journal of Data Science and Analytics* 3 (03 2017). <https://doi.org/10.1007/s41060-016-0032-z>

- [69] J. Ramsey, K. Zhan, M. Glymour, R. Sanchez Romero, B. Huang, I. Ebert-Uphoff, S. M. Samarasinghe, E. A. Barnes, and C. Glymour. 2018. TETRAD - A toolbox for causal discovery. In *8th International Workshop on Climate Informatics*.
- [70] V. Riccio, G. Jahangirova, A. Stocco, N. Humbatova, M. Weiss, and P. Tonella. 2020. Testing machine learning based systems: A systematic mapping. *Empirical Software Engineering* 25, 6 (2020), 5193–5254.
- [71] V. Riccio and P. Tonella. 2020. Model-based exploration of the frontier of behaviours for deep learning system testing. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (Virtual Event, USA) (ESEC/FSE 2020)*. ACM, 876–888. <https://doi.org/10.1145/3368089.3409730>
- [72] P. R. Rosenbaum and D. B. Rubin. 1984. Reducing bias in observational studies using subclassification on the propensity score. *J. Amer. Statist. Assoc.* 79, 387 (1984), 516–524.
- [73] M. Scutari. 2010. Learning Bayesian networks with the bnlearn R package. *Journal of Statistical Software* 35, 3 (2010), 1–22. <https://doi.org/10.18637/jss.v035.i03>
- [74] J. S. Sekhon. 2011. Multivariate and propensity score matching software with automated balance optimization: The matching package for R. *Journal of Statistical Software* 42, 7 (2011), 1–52. <https://doi.org/10.18637/jss.v042.i07>
- [75] A. Sharma and E. Kiciman. 2019. DoWhy: A Python package for causal inference. <https://github.com/microsoft/dowhy>.
- [76] N. Sharma, V. Jain, and A. Mishra. 2018. An analysis of convolutional neural networks for image classification. *Procedia Computer Science* 132 (2018), 377–384. <https://doi.org/10.1016/j.procs.2018.05.198>
- [77] S. Shimizu, P. O. Hoyer, A. Hyvärinen, and A. Kerminen. 2006. A linear non-Gaussian acyclic model for causal discovery. *Journal of Machine Learning Research* 7, 72 (2006), 2003–2030.
- [78] J. Siebert. 2023. Applications of statistical causal inference in software engineering. *Information and Software Technology* 159 (2023), 107198. <https://doi.org/10.1016/j.infsof.2023.107198>
- [79] P. Spirtes, C. Glymour, and R. Scheines. 2001. *Causation, Prediction, and Search* (2nd ed.). MIT Press, Cambridge, MA, USA.
- [80] P. Spirtes and K. Zhang. 2016. Causal discovery and inference: Concepts and recent methodological advances. *Applied Informatics* 3, 1 (18 Feb. 2016), 3. <https://doi.org/10.1186/s40535-016-0018-x>
- [81] P. C. Sruthi, S. Rao, and B. Ribeiro. 2020. Pitfalls of data-driven networking: A case study of latent causal confounders in video streaming. In *Workshop on Network Meets AI & ML (Virtual Event, USA) (NetAI '20)*. ACM, New York, NY, USA, 42–47. <https://doi.org/10.1145/3405671.3405815>
- [82] M. Steyvers, J. B. Tenenbaum, E. Wagenmakers, and B. Blum. 2003. Inferring causal networks from observations and interventions. *Cognitive Science* 27, 3 (2003), 453–489. https://doi.org/10.1207/s15516709cog2703_6
- [83] A. Stocco, B. Pulfer, and P. Tonella. 2022. Mind the Gap! A study on the transferability of virtual vs physical-world testing of autonomous driving systems. *IEEE Transactions on Software Engineering (Early Access)* (2022), 1–13. <https://doi.org/10.1109/TSE.2022.3202311>
- [84] A. Stocco and P. Tonella. 2022. Confidence-driven weighted retraining for predicting safety-critical failures in autonomous driving systems. *Journal of Software: Evolution and Process* 34, 10 (2022), e2386. DOI: <https://doi.org/https://doi.org/10.1002/smr.2386>
- [85] A. Stocco, M. Weiss, M. Calzana, and P. Tonella. 2020. Misbehaviour prediction for autonomous driving systems. In *42nd International Conference on Software Engineering (ICSE)* (Seoul, South Korea). ACM, 359–371. <https://doi.org/10.1145/3377811.3380353>
- [86] S. Sun and A. Podgurski. 2016. Properties of effective metrics for coverage-based statistical fault localization. In *IEEE International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 124–134. <https://doi.org/10.1109/ICST.2016.31>
- [87] H. Theil. 1961. *Economic Forecasts and Policy*. North-Holland Publishing Company, Amsterdam.
- [88] D. L. Thistlethwaite and D. T. Campbell. 1960. Regression-discontinuity analysis: An alternative to the ex post facto experiment. *Journal of Educational Psychology* 51 (1960), 309–317.
- [89] Y. Tian, K. Pei, S. Jana, and B. Ray. 2018. DeepTest: Automated testing of deep-neural-network-driven autonomous cars. In *40th International Conference on Software Engineering (ICSE)* (Gothenburg, Sweden). ACM, 303–314. <https://doi.org/10.1145/3180155.3180220>
- [90] J. Togelius, A. J. Champandard, P. L. Lanzi, M. Mateas, A. Paiva, M. Preuss, and K. O. Stanley. 2013. Procedural content generation: Goals, challenges and actionable steps. In *Artificial and Computational Intelligence in Games. Dagstuhl Follow-Ups, Vol. 6. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany*, 61–75. <https://doi.org/10.4230/DFU.Vol6.12191.61>

- [91] C. E. Tuncali, G. Fainekos, H. Ito, and J. Kapinski. 2018. Simulation-based adversarial test generation for autonomous vehicles with machine learning components. *CoRR* abs/1804.06760 (2018). arXiv:1804.06760<http://arxiv.org/abs/1804.06760>
- [92] W. M. van der Wal and R. B. Geskus. 2011. ipw: An R package for inverse probability weighting. *Journal of Statistical Software* 43, 13 (2011), 1–23. <https://doi.org/10.18637/jss.v043.i13>
- [93] H. Wang, Y. Jin, and J. Doherty. 2017. Committee-based active learning for surrogate-assisted particle swarm optimization of expensive problems. *IEEE Transactions on Cybernetics* 47, 9 (2017), 2664–2677. <https://doi.org/10.1109/TCYB.2017.2710978>
- [94] W. Webber, A. Moffat, and J. Zobel. 2010. A similarity measure for indefinite rankings. *ACM Trans. Inf. Syst.* 28, 4, Article 20 (Nov. 2010), 38 pages. <https://doi.org/10.1145/1852102.1852106>
- [95] M. Wicker, X. Huang, and M. Kwiatkowska. 2018. Feature-guided black-box safety testing of deep neural networks. In *Tools and Algorithms for the Construction and Analysis of Systems*, Dirk Beyer and Marieke Huisman (Eds.). Springer International Publishing, Cham, 408–426.
- [96] C. K. Wongchokprasitti, H. Hochheiser, J. Espino, E. Maguire, B. Andrews, M. Davis, and C. Inskip. 2019. bd2kccd/py-causal v1.2.1. <https://doi.org/10.5281/zenodo.3592985>
- [97] L. Wu, J. Tordsson, E. Elmroth, and O. Kao. 2021. Causal inference techniques for microservice performance diagnosis: Evaluation and guiding recommendations. In *IEEE International Conference on Automatic Computing and Self-Organizing Systems (ACSOS)*. IEEE, 21–30. <https://doi.org/10.1109/ACSOS52086.2021.00029>
- [98] W. Wu, H. Xu, S. Zhong, M. R. Lyu, and I. King. 2019. Deep validation: Toward detecting real-world corner cases for deep neural networks. In *49th Annual IEEE/IFIP Int. Conference on Dependable Systems and Networks (DSN)*. IEEE, 125–137.
- [99] Z. Yang, Y. Chai, D. Anguelov, Y. Zhou, P. Sun, D. Erhan, S. Rafferty, and H. Kretschmar. 2020. SurfelGAN: Synthesizing realistic sensor data for autonomous driving. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 11115–11124. <https://doi.org/10.1109/CVPR42600.2020.01113>
- [100] M. Zhang, Y. Zhang, L. Zhang, C. Liu, and S. Khurshid. 2018. DeepRoad: GAN-based metamorphic testing and input validation framework for autonomous driving systems. In *33rd ACM/IEEE International Conference on Automated Software Engineering (ASE)*. ACM, 132–142. <https://doi.org/10.1145/3238147.3238187>
- [101] X. Zheng, B. Aragam, P. K. Ravikumar, and E. P. Xing. 2018. DAGs with no tears: Continuous optimization for structure learning. In *Advances in Neural Information Processing Systems*, Vol. 31. Curran Associates, Inc.
- [102] H. Zhou, W. Li, Z. Kong, J. Guo, Y. Zhang, B. Yu, L. Zhang, and C. Liu. 2020. DeepBillboard: Systematic physical-world testing of autonomous driving systems. In *42nd International Conference on Software Engineering (ICSE)*. ACM, 347–358.
- [103] Z. Zhou, Y. Soon Ong, P. B. Nair, A. J. Keane, and K. Y. Lum. 2007. Combining global and local surrogate models to accelerate evolutionary optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 37, 1 (2007), 66–76. <https://doi.org/10.1109/TSMCC.2005.855506>

Received 25 January 2023; revised 28 May 2023; accepted 1 November 2023