RESEARCH-ARTICLE

Using Causal Inference to Test Systems with Hidden and Interacting Variables: An Evaluative Case Study

MICHAEL FOSTER, The University of Sheffield, Sheffield, South Yorkshire, U.K.

R. HIERONS, The University of Sheffield, Sheffield, South Yorkshire, U.K.

DONGHWAN SHIN, The University of Sheffield, Sheffield, South Yorkshire, U.K.

NEIL WALKINSHAW, The University of Sheffield, Sheffield, South Yorkshire, U.K.

CHRISTOPHER WILD, The University of Sheffield, Sheffield, South Yorkshire, U.K.

Published: 17 June 2025

EASE '25: Evaluation and Assessment in Software Engineering
June 17 - 20, 2025
Istanbul, Turkiye

EASE '25: Proceedings of the 29th International Conference on Evaluation and Assessment in Software Engineering (June 2025)
https://doi.org/10.1145/3756681.3756967
ISBN: 9798400713859

Using Causal Inference to Test Systems with Hidden and Interacting Variables: An Evaluative Case Study

Michael Foster
The University of Sheffield
Sheffield, United Kingdom
m.foster@sheffield.ac.uk

Robert Hierons
The University of Sheffield
Sheffield, United Kingdom
r.hierons@sheffield.ac.uk

Donghwan Shin
The University of Sheffield
Sheffield, United Kingdom
d.shin@sheffield.ac.uk

Neil Walkinshaw
The University of Sheffield
Sheffield, United Kingdom
n.walkinshaw@sheffield.ac.uk

Christopher Wild
The University of Sheffield
Sheffield, United Kingdom
c.wild@sheffield.ac.uk

Abstract

Software systems with large parameter spaces, nondeterminism and high computational cost are challenging to test. Recently, software testing techniques based on causal inference have been successfully applied to systems that exhibit such characteristics, including scientific models and autonomous driving systems. One significant limitation is that these are restricted to test properties where all of the variables involved can be observed and where there are no interactions between variables. In practice, this is rarely guaranteed; the logging infrastructure may not be available to record all of the necessary runtime variable values, and it can often be the case that an output of the system can be affected by complex interactions between variables. To address this, we leverage two additional concepts from causal inference, namely effect modification and instrumental variable methods. We build these concepts into an existing causal testing tool and conduct an evaluative case study which uses the concepts to test three system-level requirements of CARLA, a high-fidelity driving simulator widely used in autonomous vehicle development and testing. The results show that we can obtain reliable test outcomes without requiring large amounts of highly controlled test data or instrumentation of the code, even when variables interact with each other and are not recorded in the test data.

CCS Concepts

• **Software and its engineering** → **Software testing and debugging**; • **Computing methodologies** → **Simulation evaluation**; **Modeling methodologies**; • **Computer systems organization** → **Embedded and cyber-physical systems**.

Keywords

Causal Testing, Causal Inference, Software Testing

Foster, Walkinshaw, Hierons, and Wild were supported by EPSRC CITCoM grant [EP/T030526/1]. Donghwan Shin was supported by EPSRC SimpliFaiS grant [EP/Y014219/1].

EASE '25, Istanbul, Turkiye

ACM ISBN 979-8-4007-1385-9/25/06
https://doi.org/10.1145/3756681.3756967

ACM Reference Format:
Michael Foster, Robert Hierons, Donghwan Shin, Neil Walkinshaw, and Christopher Wild. 2025. Using Causal Inference to Test Systems with Hidden and Interacting Variables: An Evaluative Case Study. In *Evaluation and Assessment in Software Engineering (EASE '25), June 17–20, 2025, Istanbul, Turkiye.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3756681.3756967

1 Introduction

Complex software systems appear in a broad range of applications such as computational models, autonomous driving systems, and cyber-physical systems. These have several fundamental characteristics that make them difficult to test, notably large input spaces, nondeterminism, and uncontrollable behaviour. Furthermore, long runtimes and high computational cost often limit the number of tests that can be feasibly executed.

Causal reasoning is increasingly being applied to address these testing challenges [15, 21, 43]. Well-established in fields such as epidemiology and sociology, the idea is to specify a model of the expected causal relationships between variables and use this to identify and remove bias when applying statistical estimation techniques. This enables the expected causal effects to be validated using pre-existing uncontrolled data rather than requiring a specially curated dataset, without risking test outcomes being made unreliable by a biased data generation process.

However, there are two essential challenges that have not been considered by previous work on causality-based testing: (1) Non-observability: Key variables that are required to evaluate correctness properties may not be observable. When testing relationships between software inputs and outputs, we need to account for other inputs and internal variables to isolate the causal effect of interest. If any of those variables are missing from the test data (e.g. if they are not logged during execution), this can lead to biased and unreliable test outcomes. (2) Interacting variables: Software outputs may depend on *combinations* of interacting variable values. So far, causal relationships between variables have only been considered in isolation, neglecting faults involving interactions.

We perform an evaluative case study [44] to investigate how two additional concepts from causal inference — effect modification and instrumental variable methods [29] — can address these challenges. Effect modification allows us to reason about the causal effects of interacting variables. Instrumental variable methods allow us to adjust for variables that are missing from the test data. While

these are both well-established causal inference techniques, this is the first work to explicitly apply them to a software engineering context. Our main contributions are as follows:

- We apply effect modification from causal inference to reason about interactions between variables when testing software.
- We apply instrumental variable methods from causal inference to reason about unobservable variables when testing software.
- We perform an evaluative case study considering three testing requirements in the context of the CARLA high-fidelity driving simulator [17]. The results show that the above techniques can yield reliable test outcomes for software with interacting and unobservable variables.

The remainder of this paper is structured as follows. Section 2 introduces the CARLA simulator and the testing challenges we consider in this work. Section 3 gives background on causal software testing and the essential elements of causal inference that we use in this work. For a more comprehensive introduction, we refer the reader to [29, 41]. Section 4 lays out the design of our evaluative case study. Section 5 shows how we used causal inference to test our three requirements. Section 6 provides answers to our research questions. Section 7 discusses potential threats to validity and our chosen mitigation strategies. Section 8 highlights key related works. Finally, Section 9 concludes the paper.

2 Testing Challenges

In this section, we outline the main testing challenges considered in this paper in the context of a motivating example concerning automated driving system (ADS) testing. While the testing challenges we consider in this work are not unique to the field of ADS testing, they are particularly pronounced here, which makes it an ideal context within which to explore our research questions.

2.1 Motivating Example: CARLA Driving Simulator

CARLA [17] is a popular open-source high-fidelity driving simulator developed to support the development, training, and validation of ADSs. The CARLA GitHub repository [4] has over 10,000 stars and over 3,000 forks at the time of writing. CARLA provides a wide range of configurable driving scenario entities, such as weather conditions, traffic lights, non-playing character (NPC) vehicles (i.e. traffic), and pedestrians. This makes CARLA the state-of-the-art "system" [33] for simulation-based ADS testing.

The CARLA leaderboard [3] is a benchmark for evaluating ADSs. Its V1.0 SENSOR track, which restricts ADSs to sensor inputs (e.g. cameras), has 36 entrants at the time of writing. ADSs are scored on their ability to drive predefined driving scenarios from start to end in a given time. Penalties are applied for infractions, such as collisions or running red lights. To enable a fair comparison, these penalties must be implemented correctly. If CARLA were a conventional software system, this would be trivial to test: we would simply commit each infraction and check that the correct penalty was applied. Unfortunately, CARLA exhibits four characteristics that make this impractical – nondeterminism, limited observability, interactions between parameters, and long runtimes.

Nondeterminism CARLA can produce different behaviours for different runs of the same input configuration. For example, pedestrian movement is completely random, even for the same seed [1]. Since CARLA cannot spawn an agent if their spawn point is occupied, we may end up with fewer pedestrians than specified if they move into each other's spawn points. This means that the impact of particular configurations can only be studied statistically using multiple runs. Nondeterminism also raises issues of controllability [19]: we cannot reliably elicit particular behaviours. This makes it hard to isolate the effect of any particular input.

Observability CARLA has a large number of configuration parameters and internal variables, many of which are not logged by default. This means that, not only are we unable to *control* certain aspects of the simulation directly, but we cannot even *observe* them. For example, CARLA does not record how many pedestrians and NPC vehicles were successfully spawned into the simulation. Again, this makes it hard to isolate the effects of particular inputs.

Interaction Much of CARLA's behaviour depends on complex interactions between parameters. For example, the numbers of pedestrians and NPC vehicles both affect how long it takes to drive a particular scenario, with the delay caused by increasing the number of pedestrians being compounded by busy roads, since more vehicles will have to stop to allow pedestrians to cross, causing longer traffic jams and greater disruption. This interaction further adds to the difficulty in isolating the effect of any individual input.

Execution Time and Computational Cost CARLA requires high-end hardware [4], and is time-consuming and computationally demanding to execute. Furthermore, CARLA tends to run slower than real time: it takes longer than 1 second of real-world time to run 1 second of the simulation. For example, the driving scenarios we collected as part of this study took an average of around 13 minutes to execute. The longest took over two hours, even though it was still only a few minutes of simulation time. Thus, a tester can only consider a small fraction of potential test executions, especially if configurations need to be run repeatedly to mitigate nondeterminism. Since the events that are of interest from a testing perspective (e.g., collisions) tend to occur relatively rarely, a premium is placed on the ability to extrapolate as much useful information as possible from the few test executions that can be collected.

2.2 Limitations of Existing Techniques

Within the research literature on ADS testing, the notion of faulty behaviour is often restricted to faults that are easily detectable, such as obvious driving violations (e.g. collisions or running a red light) [49, 55, 56]. The authors are not aware of any testing approaches that can test behaviour against more nuanced requirements (e.g. "The model of ego-vehicle should not affect how often it crashes."). We suspect that this is at least partially because of the practical challenges that this would entail (as mentioned above).

In principle, statistical metamorphic testing (SMT) [25] provides a framework within which to test such properties. The SMT approach involves repeatedly running the software under two (or more) configurations and performing statistical tests on the resulting output data. For example, we would run the ADS several times with two different ego-vehicles, and perform a hypothesis test to investigate whether either ego-vehicle had significantly more crashes.

This is similar to A/B testing [48], where different groups of users are assigned different versions of software to see which performs better. The main limitation of these approaches is that all variables must be carefully controlled in the manner of a laboratory experiment. This may not always be possible, especially when testing relationships between different software outputs. Furthermore, test data must be collected separately for each property being tested, meaning that large amounts of test data are often required.

3 Causal Testing

Several recent techniques apply the model-based statistical framework of causal inference (CI) [41] to test software. This *Causal Testing* excels for testing properties of nondeterministic systems where it is difficult to obtain large numbers of carefully controlled executions, such as computational models [15] and ADSs [21].

As with SMT, multiple runs of the software are used to draw statistical conclusions about the *relationships* between program inputs, outputs, and internal variables. However, CI explicitly separates the *collection* and *analysis* of test data by employing domain knowledge supplied by the tester in the form of a causal model that specifies the expected causal relationships between program variables. This means that the test cases can be evaluated using pre-existing runtime data rather than specially curated test data.

Causal Testing applies to properties framed as the effect of a treatment on an outcome, and has four main steps: (1) Specify the Causal Model, (2) Collect Test Data, (3) Define Causal Test Cases, and (4) Evaluate the Causal Test Cases. These are elaborated in the following sub-sections. In principle, everything except the initial formation of the causal model can be (semi-)automated.

3.1 Specify the Causal Model

The first step is to specify the expected causal relationships between variables in the system using a directed acyclic graph (DAG), exemplified in Figure 1. Nodes represent variables, and an edge $X \rightarrow Y$ represents the domain knowledge that X may have a direct causal effect on Y. The absence of such an edge means that X *definitely does not* have a direct causal effect on Y. A causal DAG should include all inputs, outputs, and internal variables that are relevant to the properties being tested, even if they cannot be controlled or observed. By analysing paths in a DAG [41], it is possible to identify which variables must be adjusted (controlled for) to isolate the causal effect of X on Y. We provide an example in Section 5.

Causal DAGs form an intuitive model of the system under test and are widely used in fields such as epidemiology and sociology [29], where they are often hand-drawn by domain experts. As with any model-based testing technique, drawing a DAG requires domain knowledge since it forms part of the test oracle [10]. However, causal DAGs are much lighter weight than traditional models, such as finite state machines [14]. They do not specify the precise form of the relationships between variables, merely their existence.

3.2 Collect Test Data

The second step is to collect test data. A major benefit of Causal Testing is that this data can be "observational", i.e., collected without needing to tightly control the inputs. The advantage of this from a software engineering standpoint is that the same test data can be reused to test multiple properties [15], without requiring carefully controlled test data generation. For example, it would be valid to use pre-existing log data recorded during normal use. The important limitation is that the test data must satisfy the *positivity* assumption, which is fundamental to CI [29]. Formally, this means that the probability of each treatment (typically an input configuration) of interest must be non-zero. Intuitively, this means that test outcomes are more accurate and reliable if the test data achieves a good coverage of the input space.

3.3 Define Causal Test Cases

The next step is to encode the properties to be tested as causal test cases. These are intuitively similar to metamorphic relations [12] in that we observe the effect of *changing* a particular variable. Definition 1 formalizes this, and is slightly adapted from [15].

Definition 1. Given a causal DAG G representing the expected causal relationships between variables of the system under test, a *causal test case* is a triple (X, Y, E), where X and Y are nodes in G, respectively referred to as the *treatment* and *outcome*. E is the expected causal effect of X on Y, serving as the test oracle [10].

For example, to test that the model of the ego-vehicle does not affect the number of infractions it commits, we would define our treatment variable X to be the model of the car, the outcome Y to be the number of infractions, and the expected causal effect E of X on Y to be zero (indicating no effect).

3.4 Evaluate the Causal Test Cases

The final step is to use CI to evaluate each test case. There are three sub-steps to this.

Identification: First, the DAG is used to identify which variables need to be adjusted to remove bias. This is done by automatically searching for "backdoor paths" in the DAG and variables which can be controlled to close them [41]. A common source of bias is *confounding*, where a third variable Z has causal paths to both the treatment X and the outcome Y, which can introduce a spurious correlation between X and Y, even if there is no direct causal link between them. To adjust for this, Z (and other sources of bias) are controlled for by including them as features in the estimation step (below) so that their values are properly taken into account.

Estimation: Next, we use the test data to estimate the causal effect, with 95% confidence intervals [40]. This involves using a statistical estimator, such as regression. In this work, we estimate unit Average Treatment Effect (ATE), which represents the change in the outcome Y we would expect to see if we increased the treatment X by 1. In a linear setting with $Y = aX + bZ + c$ (where a, b, and c are constant coefficients), this is given by a. To test non-linear relationships, one can add extra terms (e.g. powers, reciprocals, interaction terms) or use a machine learning model [36] if the equational form is not known. The advantage of CI is that the identification step automatically identifies the relevant features from the DAG instead of factoring in all of the (potentially irrelevant) features in the data.

Comparison: Finally, the causal effect estimate is checked against the expected causal effect E to determine the test outcome. At the coarsest level, we can simply check for the presence or absence of a causal effect. For unit ATE (defined above), there is deemed to be a causal effect if the confidence intervals do not contain zero. A

causal DAG can act as a test oracle in itself, and can be automatically transformed into a suite of causal tests that validate the specified causal effects and independence relations [16]. It is also possible to make the test oracle more precise by, for example, checking for a positive or negative causal effect, or even a specific value.

3.5 Handling Interaction and Unobservable Variables

Previous Causal Testing research has been limited to the analysis of causal effects between pairs of variables in systems where all relevant variable values are recorded in the test data. However, two additional challenges still present a barrier to its broader application: interaction and unobservable variables.

3.5.1 Interaction and Effect Modification. Many software faults manifest themselves as interactions between multiple variables [37], where several variables may need to take particular values. When this is the case, the causal effect of one variable on the outcome is modified by another variable. In CI, this is known as *effect modification* [29]: the actual *relationship* $X \rightarrow Y$ is changed, depending on the value of a third variable Z. The CI solution to this is to include an *interaction term* as an additional feature when estimating causal effects. For example, we may use the regression equation $Y = aX + bZ + cXZ + d$, where XZ is the interaction term. In Section 5, we investigate whether this allows us to obtain reliable test outcomes when variables interact.

3.5.2 Unobservable Variables. Software logs may be incomplete [11] and may not record every variable during execution. This can lead to biased, unreliable test outcomes as we may not be able to adjust for confounding variables by controlling their values. While we can sometimes instrument programs to provide extra logging, this may not always be possible.

Instrumental variable (IV) methods [41, 53] from CI provide an elegant solution to this problem under certain circumstances. As an example, consider the causal DAG in Figure 1. This shows the causal relationships between four variables: X, Y, Z, and U (which is unobserved), along with path coefficients [53] (a, b, c, d) that represent the unit ATE of each causal relationship. To estimate the direct effect of X on Y (b in Figure 1) in the presence of confounder U, we would typically need data for U to adjust its biasing effect.

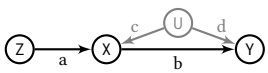

Figure 1: General setup for IVs. The unobserved variable U is highlighted in gray.

Instead, we can use Z as an *instrument* to calculate b without needing data for U. To do this, we divide the total effect of Z on Y (ab) by the direct effect of Z on X (a). That is, $ab/a = b$. This is possible because Figure 1 satisfies the following three conditions and operates in a linear setting: (1) there is no arrow between U and Z, (2) there is an arrow between Z and X, and (3) there is no direct arrow from Z to Y. Although these assumptions can be restrictive, and IV methods tend to give less precise estimates [29], they nevertheless enable us to draw causal conclusions about relationships between variables, which would otherwise be impossible. In Section 5.5, we investigate whether IV methods allow us to obtain reliable test outcomes for systems with unobservable variables but where the DAG conforms to the above constraints.

4 Research Design

We perform an evaluative case study [44] to gain an in-depth understanding of *how* Causal Testing applies to software systems like CARLA, where interacting and unobservable variables prevent us from obtaining reliable test outcomes with current techniques. The case study methodology is well suited to this because our object of study is a contemporary phenomenon (Causal Testing) that must be studied in its context (by testing a system) and not in isolation [45]. Our study design follows the guidelines of Runeson et al. [45].

4.1 Objectives and Research Questions

The goal of this study, stated using the Goal-Question-Metric (GQM) approach [51], is to "analyse Causal Testing *for the purpose of* evaluation and characterisation of its testing ability *with respect to* software systems with nondeterminism and limited controllability and observability *from the point of view of* software testers *in the context of* ADS testing". Critically, we are investigating the *process* of Causal Testing rather than trying to find faults in individual systems. Furthermore, the application of Causal Testing to CARLA is not about generating test scenarios, like many existing ADS testing studies [49, 55, 56]. Causal Testing is not intended to replace the existing ADS testing techniques but should instead be considered complementary. We achieve our goal by answering the following research questions.

RQ1 *Can Causal Testing deliver reliable test outcomes for software with interacting parameters?* To answer RQ1 we examine whether the use of interaction terms produces more reliable test outcomes.
RQ2 *Can Causal Testing deliver reliable test outcomes when using uncontrolled data?* To answer RQ2, we compare causal effect estimates calculated using SMT-style data with those calculated using a smaller amount of less controlled data.
RQ3 *Can Causal Testing deliver reliable test outcomes for software with unobservable parameters?* To answer RQ3, we compare causal effect estimates calculated using IV methods, traditional adjustment (which requires the values to be observed), and no adjustment to investigate how this affects the *accuracy* of our estimates and the *reliability* of test outcomes.
RQ4 *Can Causal Testing discover faults under the above circumstances?* To answer RQ4, we consider the unexpected behaviour we encountered when testing our requirements.

4.2 Case Selection and Units of Analysis

Our case study is charactesized as *single-case* and *embedded* [45]. Our case is the CARLA platform, and we have multiple units of analysis embedded within this. Our units of analysis are the following three requirements, which we selected to enable us to investigate key aspects of Causal Testing and answer our RQs, while still being relevant to ADS testing.

RE1 (Infraction penalties). The CARLA leaderboard evaluates an ADS's ability to drive a set of predefined scenarios from start to end

in a given time. The `DrivingScore` is then calculated as the proportion of the route that the ADS managed to complete within its lane, with penalties being applied for any infractions committed. This is shown in Equation (1), which is given on the CARLA leaderboard website [3]. For the leaderboard to be a fair platform, it is crucial that Equation (1) is implemented correctly. This is hard to test using traditional techniques as we cannot reliably force particular infractions without incurring the considerable overhead of building a custom ego-vehicle and specially controlled driving environment.

$$InfractionPenalty \times CompletionScore \times (1 - OutsideLane) \quad (1)$$

RE2 (Ego-vehicle model). Human drivers are expected to adapt well to new models of vehicles, for example, when they buy a new car. It would be beneficial if ADSs could also achieve this, as it would mean they would not need to be retrained every time a new car was released. Our objective is to test whether the model of the ego-vehicle has a causal effect on the number of infractions that occur. This cannot be tested using traditional techniques as it is a statistical property over multiple runs. While SMT could test this property, it requires a large amount of highly controlled test data.
RE3 (CARLA version). This case explores a regression testing scenario between different CARLA versions. Our subject ADSs are designed to run on CARLA v0.9.10.1, but several versions of CARLA have subsequently been released. Since the changelog [2] does not suggest that any changes or additional features should significantly affect performance, the ADSs should run equally well (if not better) on newer CARLA versions. Here, we test that newer versions of CARLA do not adversely impact the real-world time taken to simulate in-simulation time. As with RE2, this is a statistical property over multiple runs. Previous approaches to Causal Testing [15] cannot test this, as there are unobservable variables at play. That is, the numbers of pedestrians and NPC vehicles are not recorded in the CARLA logs by default. They are unobserved.

We test our three requirements (the units of analysis) on four driving agents (which form sub-units of analysis) and analyse the resulting evidence separately. We used the top two ADSs on the CARLA leaderboard with available and reusable code: TCP [54] and CARLA Garage [30]. Each has two kinds of driving agents (privileged and trained), which behave very differently. *Privileged agents* have access to "privileged" information such as the road layout and the locations of the other agents. This makes them excellent drivers who commit very few infractions. *Trained agents* are machine learning models trained on data collected by a privileged agent. They drive using only non-privileged data sources such as image data and LiDaR. Thus, they typically commit more infractions than the privileged agents. To make our study as diverse as possible, we consider one of each kind of agent for each ADS.

5 Data Collection and Analysis

This section presents how we obtained and analysed the evidence that we will use to answer our RQs in Section 6. Our evidence is obtained by applying the *four steps of causal testing* outlined in Section 3. The first two steps (constructing the DAG and collecting test data) are shared between the three requirements. The last two steps (defining and evaluating causal test cases) are unique to each requirement. In particular, we consider how different estimation techniques, including interaction and IV methods, lead to different causal effect estimates and test outcomes. Our replication package[1] includes the data and code used to answer the RQs in this paper, as well as the artefacts (causal DAGs, test code and ADS setup) required to reproduce the results from fresh executions of CARLA.

5.1 Step 1: Specify the Causal Model

The first step of Causal Testing is to construct a causal DAG to represent the system. This is shown in Figure 2, and is shared between our three requirements. We used the CARLA documentation [3, 4] and domain knowledge to determine which variables were relevant to our three requirements and how they related to each other. The root nodes (`Weather`, `EgoVehicle`, `NPCvehicles`, `Pedestrians`, and `RouteLength`) represent CARLA configuration inputs. The other nodes represent outputs, and will be discussed when we test the relevant requirements. We used the method described in [15] to determine the connections between the nodes. Specifically, we assume that inputs are independent of each other (since they are chosen by the tester), and prune connections between the remaining nodes based on our knowledge of the system.

5.2 Step 2: Collect Test Data

The second step of Causal Testing is to collect test data. To evaluate our requirements, we need each of our four driving agents to drive multiple models of ego-vehicle (RE2) using multiple versions of CARLA (RE3). We need to record their infractions (RE1 and RE2) and the simulation runtimes (RE3). While we can control the version of CARLA and the model of ego-vehicle, we cannot force infractions to occur, nor can we control the runtime of the simulation. We must allow these to happen "naturally", as they normally would. The advantage of Causal Testing is that the DAG in Figure 2 allows us to identify and adjust for any bias this introduces [15].

To generate our test data, we followed the data collection instructions on the README page of each ADS, using the provided driving scenarios for the Town 01 map, which is the simplest of 12 road layouts supported by CARLA. These scenarios are primarily intended for training and evaluating the respective ADSs. TCP is distributed with 300 scenarios for Town 01. CARLA Garage has just 132 scenarios, which are distinct from those of TCP. We ran the scenarios for two versions of CARLA (v0.9.10.1 and v0.9.11) and two models of ego-vehicle (the default Lincoln MKZ2017 and the BMW Isetta), terminating execution at the first infraction so that a maximum of one infraction per scenario is considered.

Crucially, we did not generate the driving scenarios ourselves; we chose not to apply state-of-the-art ADS test scenario generation techniques here to maintain focus on the Causal Testing methodology as a whole. This means our test data may not achieve the best coverage or produce the best test outcomes, but it makes it well suited to evaluate RQ2 as it represents the kind of pre-existing runtime data with which Causal Testing is intended to be used [15].

To judge the accuracy and reliability of the IV methods we use to test RE3, we modified the code of TCP to enable the numbers of pedestrians and NPC vehicles (i.e., traffic participants) to be customised and recorded. We then randomly spawned between 80 and 200 of each. This was a non-trivial process that required several

[1] https://github.com/CITCOM-project/carla-case-study

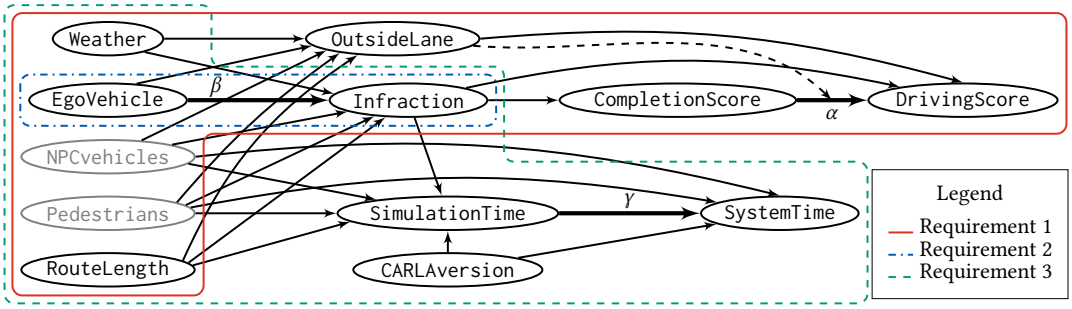

Figure 2: The causal DAG for all three requirements, with the variables relevant to each requirement highlighted. The specific causal edges of interest are emboldened for clarity. Unobservable variables are drawn in grey. The dashed edge represents effect modification. We use the notation proposed in [52] of drawing (dashed) edges from nodes to other edges.

code files to be modified, but it allowed us to perform traditional adjustment, which we use as a "gold standard" to answer RQ3. It also enabled us to more effectively investigate RE1 as it yielded test data with more runs containing an infraction than the default value of 120. We did not modify CARLA Garage except to change the model of the ego-vehicle, which is hardcoded.

5.3 RE1: Infraction Penalties

Having drawn the DAG (step 1) and collected test data (step 2), we now define (step 3) and evaluate (step 4) causal test cases for RE1 to test that the correct penalty is applied for each infraction.

5.3.1 Step 3: Define Causal Test Cases. Definition 1 states that a causal test case has three components: a treatment, an outcome, and an expected causal effect. We want to test that the *infraction penalty* is correct. This corresponds to α in Figure 2, which is the effect of `CompletionScore` on `DrivingScore`. Thus, these are our treatment and outcome, respectively.

There are four possible infractions in the Town 01 map: collisions with pedestrians, vehicles, and objects, and running red lights. We define one causal test for each, and one for no infraction. These all have `CompletionScore` as the treatment and `DrivingScore` as the outcome. The expected causal effect is the corresponding infraction penalty, taken from the CARLA leaderboard [3] (see Table 1).

5.3.2 Step 4: Evaluate the Causal Test Cases. We can now evaluate our five causal test cases. As discussed in Section 3, this process has three substeps: identification, estimation, and comparison to the expected causal effect.

Identification. The first step is to use Figure 2 to identify sources of bias that must be adjusted to obtain an unbiased estimate. Figure 2 shows that `Infraction` is a common cause of `CompletionScore` and `DrivingScore`. To adjust for this, the data are grouped into *strata* by `Infraction`, and each test case is evaluated using the corresponding stratum. Because some infractions happened more than others (see Table 1), the strata are not all the same size, which will affect the resulting estimate. We discuss this further in Section 6.

The dashed edge in Figure 2 shows that `OutsideLane` is an *effect modifier* of `CompletionScore` on `DrivingScore`. This stems from Equation (1), which contains the term `ComplectionScore` × `OutsideLane`, indicating that the two variables interact. If we do not adjust for this, our estimates of the infraction penalties will be biased, potentially leading to unreliable test results.

Estimation. Having identified the sources of bias, we now estimate the causal effect α. We use the regression model in Equation (2) for this, where c is a constant. This has the same form as Equation (1) from the CARLA leaderboard website [3], except that `Infraction penalty` is replaced with α_1, and a constant term has been added for completeness[2]. If the `DrivingScore` is being calculated according to Equation (1), the estimated coefficient α_1 (i.e. the unit ATE) should equal the penalty for each infraction, as discussed in Section 3.

$$\alpha_1 \times \texttt{CompletionScore} \times (1 - \texttt{OutsideLane}) + c \tag{2}$$

Note that expanding out the bracket in Equation (1) gives the *interaction term* `CompletionScore` × `OutsideLane`. This is how we adjust for the effect modification bias. To investigate the importance of effect modification, we also consider Equation (3), which ignores the effect modification of `OutsideLane`. This represents a naive estimation that only adjusts for confounding (by stratifying the data). Figure 2 indicates that this should produce a biased estimate.

$$\alpha_2 \times \texttt{CompletionScore} + c \tag{3}$$

Expected Effect. We determine the test outcomes by comparing our estimates for α_1 and α_2 to the expected causal effects, which are the infraction penalties from the CARLA leader board [3]. Table 1 shows these values, and is divided into two sections. The first section shows the estimates for α_1 in Equation (2). The second section shows the estimates for α_2 in Equation (3). Missing entries correspond to infractions that were never committed. Infractions that only occurred once did not give sufficient data to estimate confidence intervals.

Test Outcomes. The top five rows of Table 1 show that the estimates of α_1 in Equation (2) are as expected. The identical confidence intervals come from the infraction penalty being deterministic, so there is no variation in the dataset. Thus, the regression model perfectly fits the data. Every test case that we could evaluate passed for all four ADSs.

[2]If we were not adjusting for `Infraction` by stratification, it would also need a term in the equation.

Table 1: Test outcomes with estimated α_1 as per Equation (2), and α_2 as per Equation (3) to 3 decimal places. Failing test cases are highlighted with an (*) symbol. Missing values are shown with a (-) symbol.

	Infraction	α_{Expected}	TCP Privileged	TCP Trained	CARLA Garage Privileged	CARLA Garage Trained
Equation (2) α_1	No infraction	1.00	1.000[1.000, 1.000]	1.000[1.000, 1.000]	1.000[1.000, 1.000]	1.000[1.000, 1.000]
	Red light	0.70	0.700[0.700, 0.700]	0.700[0.700, 0.700]	0.700[-, -]	0.700[0.700, 0.700]
	Collisions layout	0.65	0.650[0.650, 0.650]	0.650[0.650, 0.650]	-	0.650[0.650, 0.650]
	Collisions vehicle	0.60	0.600[0.600, 0.600]	0.600[0.600, 0.600]	0.600[0.600, 0.600]	0.600[0.600, 0.600]
	Collisions pedestrian	0.50	-	0.500[0.500, 0.500]	-	0.500[-, -]
Equation (3) α_2	No infraction	1.00	1.000[1.000, 1.000]	1.024[1.005, 1.043]	1.000[1.000, 1.000]	1.046[0.964, 1.128]
	Red light	0.70	0.700[0.700, 0.700]	0.698[0.696, 0.700]	0.700[-, -]	0.700[0.700, 0.700]
	Collisions layout	0.65	0.650[0.650, 0.650]	0.650[0.620, 0.680]	-	* 0.538[0.475, 0.601]
	Collisions vehicle	0.60	0.600[0.600, 0.600]	0.604[0.594, 0.614]	0.600[0.600, 0.600]	* 0.482[0.372, 0.591]
	Collisions pedestrian	0.50	-	0.500[0.500, 0.500]	-	0.500[-, -]

The bottom five rows of Table 1 show that ignoring the effect modification from `OutsideLane` can lead to unreliable test outcomes. For the privileged drivers, the results are unaffected because they never went `OutsideLane`, thereby nullifying the bias. However, the estimates for the trained drivers are less precise than for α_1, and two test cases fail because the effect estimates are not close enough (in this case, within 5%) to the expected value. This is because Equation (3) does not include an interaction term, meaning α_2 aggregates the infraction penalty and the proportion of the route spent `OutsideLane`.

5.4 RE2: Ego-Vehicle Model

Let us now define and evaluate causal test cases for RE2 from Section 4, which tests that the model of ego-vehicle does not have a significant effect on the infractions committed.

5.4.1 Step 3: Define Causal Test Cases. As for RE1, we first define the treatment, outcome, and expected causal effects. We want to test that the model of `EgoVehicle` does not impact the `Infractions` we observe. Hence, the `EgoVehicle` is the treatment, and `Infraction` is the outcome. Our expected causal effect (β in Figure 2) is zero, since a good ADS should intuitively perform equally well on any vehicle, just as we would expect from a human driver.

5.4.2 Step 4: Evaluate the Causal Test Cases. Having defined our causal test case, we now carry out identification, estimation, and comparison to the expected causal effect.

Identification. Figure 2 shows that there is no bias that needs adjusting for here. `EgoVehicle` is an input to the software, so there are no common causes or effect modifiers.

Estimation. Since there are no sources of bias to adjust for, and we do not have a predefined equation to relate `EgoVehicle` and `Infraction`, we simply fit a model of the form $\texttt{Infraction} = \beta \times \texttt{EgoVehicle} + c$, where β is the causal effect. We here identify each `Infraction` by its numeric penalty rather than its name, as we did in Section 5.3. Since our test data includes every driving scenario run with both models of the ego-vehicle, Causal Testing using the full dataset effectively becomes SMT. To help answer RQ2, which concerns uncontrolled data, we additionally estimate β, using the first half of the driving scenarios for the Lincoln, and the second half for the BMW, for each version of CARLA. When the data is partitioned in this way, no route is driven by both ego-vehicles, so SMT is not directly applicable. However, we expect Causal Testing to produce similar estimates that lead to the same test outcomes.

Table 2: Estimated effect on the infraction penalty of changing the model of ego-vehicle from the Lincoln MKZ2017 to the BMW Isetta. Failing test cases are highlighted with (*).

ADS	β estimate	β estimate 1/2
TCP trained	*-0.111[-0.131, -0.092]	*-0.115[-0.143, -0.087]
TCP privileged	* 0.0132[0.003,0.023]	* 0.018[0.005, 0.031]
CARLA G. trained	*-0.112[-0.135, -0.089]	* -0.102[-0.133, -0.07]
CARLA G. privileged	0.006[-0.003,0.015]	0.003[-0.01, 0.016]

Expected Effect. To determine the test outcomes, we compare our estimates of β to the expected causal effect (i.e., zero). As mentioned in Section 3.4, the absence of a causal effect is indicated by the confidence intervals for the estimate containing zero. Table 2 shows our estimates for each of the four drivers. The second column shows our estimates calculated using the full dataset, where each agent drove both vehicles for all scenarios. The last column shows our estimates calculated using the dataset, where each agent drove each vehicle for half of the scenarios. As expected, the two columns show very similar causal effects for each driver.

Test Outcomes. Table 2 shows that the test case for the CARLA Garage privileged driver passes. The confidence intervals contain zero, indicating no significant causal effect. The other three drivers fail. Both trained drivers have a negative effect of around -0.1 with confidence intervals that do not contain zero. This indicates that the BMW leads to worse driving than the Lincoln. This is not surprising as the drivers were only trained in the Lincoln, but the result is still cause for concern and is discussed further in Section 6.

More surprisingly, the TCP privileged driver seems to drive *better* in the BMW than in the Lincoln. While the effect size is very small, the confidence intervals do not contain zero, so the test case fails. This is surprising, and we will discuss the underlying causes and implications of this in Section 6.

5.5 RE3: CARLA Version

We now define and evaluate the causal test case for RE3. This considers a regression testing scenario to validate that updating CARLA from v0.9.10.1 to v0.9.11 does not adversely affect the performance.

5.5.1 Define Causal Test Cases. Our expected causal effect is "not positive", as we do not anticipate a slower simulation, but we do not mind if it speeds it up or stays the same.

Defining the treatment and outcome is a little more complex than the first two requirements. Simply testing the causal effect of the `CARLAversion` on the `SystemTime` does not incorporate the actual performance of the simulation, i.e. how much real-world time it takes to simulate each second of in-simulation time. This is characterised by the direct causal effect of `SimulationTime` on `SystemTime`. We need to test that this causal effect stays the same *between* the versions of CARLA. Thus, `SimulationTime` is our treatment and `SystemTime` is our outcome.

5.5.2 Evaluate the Causal Test Cases. Having defined our causal test case, we now carry out identification, estimation, and comparison to the expected causal effect.

Identification. Figure 2 shows that there are three sources of bias here: the `CARLAversion`, and the numbers of `NPCvehicles` and `Pedestrians`, as these are all common causes of our treatment and outcome. The intuition for this is that heavy traffic may lead to routes taking more simulation time to complete and more real-world time per time step, as there are more agents to update.

Unlike RE2, SMT is not applicable here since, as mentioned in Sections 2 and 5.2, we cannot precisely control the values of `NPCvehicles` and `Pedestrians`. Indeed, we are not even able to *observe* these values by default, as they are not logged by TCP or CARLA Garage. This means that we cannot collect the controlled data necessary to perform SMT.

Estimation. Having established our sources of bias, we can now estimate the causal effects. Since we are comparing causal effects between versions of CARLA, we adjust for the bias from `CARLAversion` by considering the two versions separately. The adjustment for `NPCvehicles` and `Pedestrians` is more nuanced, and we will consider and compare three different estimation methods here in order to obtain sufficient evidence to answer RQ3.

Firstly, because the values of `NPCvehicles` and `Pedestrians` are not logged by default, we will use IV methods (see Section 3.5.2) to estimate our causal effect γ without needing this data. Using `RouteLength` as the *instrument*, we can estimate γ by dividing its total causal effect on `SystemTime` (Equation (4a)) by its direct effect on `SimulationTime` (Equation (4b)).

$$\texttt{SystemTime} = \gamma_{sys} \times \texttt{RouteLength} \tag{4a}$$

$$\texttt{SimulationTime} = \gamma_{sim} \times \texttt{RouteLength} \tag{4b}$$

$$\gamma = \gamma_{sys}/\gamma_{sim} \tag{4c}$$

We use `RouteLength` as the instrument because it matches the causal structure in Figure 1. That is, there is no edge between `Pedestrians` or `NPCvehicles` and `RouteLength`, and there is a path `RouteLength` → `SimulationTime` → `SystemTime`. While we cannot know for sure that the relationships are linear, the intuition is that the longer a route is, the more simulation time it should take, since the ego-vehicle has to travel further, so more wall-clock `SystemTime` is required to run the simulation.

Secondly, since we have no way of knowing the true causal effect, we created an artificial "gold standard" (as discussed in Section 5.2) by modifying TCP to record the numbers of pedestrians and NPC vehicles to enable traditional adjustment [41] using Equation (5)

Table 3: Estimated direct causal effect of simulation time on system time for the different versions of CARLA. This represents how much real-world time it takes to simulate one second of the simulation.

	Driver	CARLA v0.9.10.1	CARLA v0.9.11
IV methods (Equation 4)	TCP trained	4.470[4.401, 4.569]	6.829[6.766, 6.886]
	TCP privileged	3.886[3.837, 3.938]	6.306[6.253, 6.364]
	Garage trained	6.412[6.199, 6.595]	8.522[8.312, 8.767]
	Garage Privileged	7.751[7.389, 8.100]	8.383[8.114, 8.617]
Gold standard (Equation (5))	TCP trained	4.523[4.437, 4.609]	6.677[6.605, 6.749]
	TCP privileged	3.838[3.759, 3.918]	6.180[6.101, 6.259]
	Garage trained	6.779[6.696, 6.861]	9.009[8.875, 9.143]
	Garage Privileged	7.162[6.883, 7.441]	7.814[7.398, 8.231]
No adjustment (Equation (6))	TCP trained	4.522[4.437, 4.607]	6.682[6.611, 6.753]
	TCP privileged	3.832[3.755, 3.909]	6.182[6.104, 6.260]
	Garage trained	6.779[6.696, 6.861]	9.009[8.875, 9.143]
	Garage Privileged	7.162[6.883, 7.441]	7.814[7.398, 8.231]

to compute estimates that are as accurate as possible. We gained the same data for CARLA Garage by manual code inspection, revealing that it always spawns 120 NPC vehicles and either zero or one pedestrian, depending on the driving scenario. In both cases, the information was time-consuming to obtain, and may not be obtainable at all in the general case.

$$\begin{aligned}\texttt{SystemTime} = \gamma \times \texttt{SimulationTime} + \\ c_1 \times \texttt{Pedestrians} + c_2 \times \texttt{NPCvehicles} + c_3\end{aligned} \tag{5}$$

Finally, we consider Equation (6), which ignores the confounding effect of `Pedestrians` and `NPCvehicles`. Figure 2 indicates that this should give a biased estimate. This may be more *accurate* (i.e. closer to traditional adjustment) than IV methods, if the bias is sufficiently weak. However, this cannot be determined in advance.

$$\texttt{SystemTime} = \gamma \times \texttt{SimulationTime} + c \tag{6}$$

Expected Effect. We now determine the test outcomes by comparing estimates of γ between CARLA versions. Table 3 shows the estimated direct causal effect of `SimulationTime` on `SystemTime`, i.e. how long it takes to simulate one second of time in simulation (γ in Figure 2). We here expect the causal effect to be zero.

Test Outcomes. In Table 3, the first four rows show the estimates for γ calculated using IV methods. This shows that CARLA 0.9.11 is slower for all four drivers. The confidence intervals for the corresponding estimates between different versions of CARLA do not overlap, so the test cases all fail.

The second four rows show that classical adjustment gives similar estimates, although the confidence intervals between CARLA versions for the CARLA Garage privileged driver overlap very slightly, leading to a passing test result. For the other drivers, there is no overlap and test cases still fail.

The final four rows show that the biased estimates calculated without adjustment are very close to those calculated with classical adjustment and produce the same test outcomes. We will discuss this further in Section 6.3

6 Analysis and Answers to Research Questions

This section answers our RQs using the evidence from Section 5.

6.1 RQ1 Can Causal Testing deliver reliable test outcomes for software with interacting parameters?

In RE1, the infraction penalty is the direct causal effect of the `CompletionScore` on the `DrivingScore`. The proportion of the route spent `OutsideLane` interacts with the `CompletionScore`, introducing a source of bias. We used an *interaction term* in the estimator to adjust for this, allowing us to validate the penalties for each infraction. When we estimated the causal effects without adjusting for this bias, two test cases for the CARLA Garage trained driver failed (even though they should have passed) because the causal effect estimates were not within 5% of the expected effect.

> *Interaction between parameters can cause tests to fail when there is no fault. Causal Testing enables us to isolate direct causal effects, even when variables interact.*

6.2 RQ2 Can Causal Testing deliver reliable test outcomes when using uncontrolled data?

In RE1, we stratified our test data by `Infraction` to adjust for its bias. To achieve a similar result using SMT, we would need to control which infraction the ego-vehicle committed each test run. This is impossible here due to the non-controllability of CARLA and the ADSs under test, meaning that some infractions did not occur often enough in the test data for us to calculate confidence intervals. This highlights an inherent property of using uncontrolled data: we can only test aspects of behaviour that are covered by the data.

In RE2, we discovered an unexpected direct causal link between the model of ego-vehicle and the infraction penalty for three of the four driving agents we tested (i.e. the choice of ego-vehicle model has a causal effect on the infractions committed). Here, causal testing essentially reduces to SMT because there is no confounding between the treatment and outcome. However, without the DAG in Figure 2, we would have no way to confirm this, so would not have been able to draw a causal conclusion. We also investigated the ability of Causal Testing to obtain reliable test outcomes from uncontrolled data by evaluating the same test cases using just half of our test data, where no route was driven by both models of the ego-vehicle. Where SMT is not directly applicable to this data, the causal tests all led to the same test outcomes as the full dataset.

We used the same test data for all three of our requirements. We also used the same test scripts to validate four separate driving agents. While it was computationally expensive to collect test data from each driver, the effort required to test all four systems was no more than testing just one.

> *Observing sufficiently many inputs is critical to estimate causal effects. However Causal Testing allows us to use less (and less controlled) test data than would be needed for SMT. It also enables us to reuse the same test data to test multiple requirements.*

6.3 RQ3 Can Causal Testing support the testing of software with unobservable variables?

In RE3, we investigated the simulation performance of two versions of CARLA. As with RE1, SMT cannot be applied here as the causal effect of `SimulationTime` on `SystemTime` is confounded by the numbers of `Pedestrians` and `NPCvehicles`, which are not recorded in the logs by default. Since we have no way of knowing the true causal effect, we recorded their values to enable us to use traditional adjustment as an artificial "gold standard" to compare estimates calculated using IV methods and without any form of adjustment. IV methods gave a median error of 0.31, and no adjustment gave 0.001. Since our estimates represent how many real-world seconds it takes to simulate one second of simulation time, these errors are barely perceptible.

While the estimates produced without adjustment are closer to the gold standard, the IV estimates still produce reliable test outcomes. In the general case, where we could not obtain the values of confounding variables, we would have no way of knowing which estimate is more accurate. However, IV methods produce causally valid and sufficiently accurate estimates as they adjust for the bias without needing to know the values of the confounders.

> *Causal Testing enables unbiased causal effect estimates to be calculated when we cannot observe certain variables, as long as certain assumptions are satisfied.*

6.4 RQ4 Can Causal Testing reveal faults in software with interacting and unobservable parameters?

While testing RE2, we discovered two inconsistencies. Firstly, the trained agents performed worse when driving ego-vehicles that they were not trained on. This is concerning, as it suggests that expensive training data needs to be collected for every new vehicle.

Secondly, we discovered that the TCP privileged agent performs slightly better when driving the BMW rather than the Lincoln, which is unexpected as the privileged agent is not a trained model, so it should drive all vehicles equally well. Inspecting the driving scenarios revealed that the ego-vehicle is sometimes spawned already committing an infraction (e.g. just in front of a red light). The BMW experiences this less because it is smaller than the Lincoln. While this behaviour is unexpected, we do not call it a "bug", as neither CARLA nor TCP is doing anything wrong. It is just that some of TCP's driving scenarios represent unrealistic behaviour.

In RE3, we discovered a significant decrease in performance between CARLA v0.9.10.1 and CARLA v0.9.11. Although the cause of this is beyond the scope of this paper, it suggests either a regression or an omission from the changelog [2]. Without Causal Testing, we would not be able to obtain reliable test outcomes for this requirement because we cannot control (or even observe, by default) the numbers of pedestrians and NPC vehicles within the simulation. We would, therefore, only be able to conclude that the CARLA version was *associated* with a change in runtime.

> *Causal Testing can discover faults in software with interacting and unobservable parameters.*

7 Threats to Validity

External validity: We carried out an evaluative case study by testing three requirements surrounding ADS testing. We chose this setting because it addresses the challenges posed in previous work on Causal Testing [15, 21] (see Section 2). As discussed in Section 3, Causal Testing can, in principle, test any behaviour framed as the effect of treatment on an outcome. While underlying CI has been shown to be generally applicable [9], including for investigating complex non-linear relationships between variables, a broader study is required to establish the circumstances under which Causal Testing is applicable in the general case.

Internal validity: We selected our three requirements for their relevance to our research questions and drew the associated DAG ourselves. This leads to the risk that the success of the approach has been biased: that our chosen requirements favour the technique. This is an intrinsic risk to any case study and is integral to our future work. However, it is worth noting that we did not control the test data; the driving scenarios formed part of the training data for TCP and Carla Garage. The fact that the same routes could be used to address all three test objectives is a testament to a core attribute of CI (and Causal Testing) – the fact that the approach used to analyse a test set is independent of the data.

8 Related Work

ADS Testing. There is a wealth of literature on ADS testing [49, 50, 55, 56]. A key research topic in this area is the generation of driving scenarios that lead to misbehaviour [27, 57], with several recent techniques [21, 31] employing causal reasoning. Our RQ2 showed Causal Testing is complementary to such approaches, as the generated scenario data can be used to explain *why* particular scenarios failed, and reused to test additional causal relationships. Along these lines, Han and Zhou [26] use metamorphic tests to answer questions like "Would the ego-vehicle still have crashed into an object if it had been further away?". While such questions are clearly causal, they are answered via the controlled collection of new data, where our approach can use existing data.

Machine Learning-Inferred Models of Tested Behaviour. In this work, we used causality-informed linear regression models to estimate causal effects. This relates to a significant body of work on machine learning approaches for inferring models from test executions. Such approaches often use off-the-shelf algorithms, such as linear regression [5], support vector regression [13], and ensemble models [27]. Machine learning approaches have also been applied to ADSs to estimate the probability of safety violations [38, 39]. A key limitation of these approaches is that the challenges outlined in Section 2 — namely, nondeterminism, observability, interaction, and long execution times — typically prevent us from collecting a sufficiently large and diverse set of executions to characterise the underlying behaviour. Norden et al. [39] tackle this by limiting execution time, but this is not always feasible.

Causality in Software Engineering. Causal reasoning is increasingly being applied in a range of software engineering contexts [22, 47]. The technique of Causal Testing was originally published in Clark et al. [15], with subsequent papers proposing techniques to automatically generate metamorphic test cases from causal DAGs [16] and measure test adequacy [18].

Causal reasoning is also popular in the field of fault localisation. For example, Johnson et al. [32] explain the root cause of faulty software behaviour by mutating existing tests to form a suite of minimally different tests that are not fault-causing. The test suites are then compared to understand *why* a fault occurred. Several techniques also employ CI, using the program dependence graph as a DAG [6–8, 24, 42, 46].

The great advantage of Causal Inference is the fact that it can be applied to observational data, without the need for a controlled experiment. In this context, it has also been shown to be a valuable tool for empirical software engineering. Recent work by Furia et al. [20] has shown how it can be more precise at analysing programmer performance than purely predictive techniques.

Automatic Generation of DAGs. While manual creation of DAGs is widely accepted in fields such as epidemiology, causal discovery [35] aims to automatically learn causal structures from data by exploiting asymmetries that separate association from causation [23]. The ADS scenario generation techniques mentioned above [21, 31] all employ causal discovery rather than relying on the user to supply the DAG. However, a fundamental weakness of this from a testing point of view is that inferred DAGs represent the *actual system* rather than its *intended behaviour*, so it will reflect any bugs in the implementation. Causal DAGs have also been generated via static analysis of source code [34, 42].

9 Conclusion

Testing nondeterministic software with uncontrollable and unobservable variables, such as ADSs, can be challenging due to the difficulty in obtaining test data. In this paper, we investigated how two ideas from CI — effect modification and instrumental variables — can be used to tackle these problems.

We performed an evaluative case study by testing three requirements of the CARLA driving simulator and two associated ADSs. Our results indicate that the above techniques can facilitate the testing of properties for which we could not otherwise obtain reliable outcomes using uncontrolled observational data. Interaction terms in statistical estimators allow us to isolate direct causal effects in the presence of effect modification. IV methods enable us to adjust for bias from variables that do not appear in the test data, although the accuracy will vary from system to system. Furthermore, the main benefit of Causal Testing identified in [15], namely that we can obtain useful test results using observational data not collected expressly for testing, still applies in this new context. A more extensive study to investigate the limitations and generalisability of the approach is desirable future work.

As identified in [15, 16], the main barrier to Causal Testing is the domain knowledge necessary to draw the causal DAG. A promising direction for future research is the creation of (semi-)automated tools to assist developers with this process. Another line of research would be to investigate the applicability of IV methods for testing concurrent systems, where logging can hide faults [28].

References

[1] Accessed 2023-05-05. Non determinism of Walker AI controllers. https://github.com/carla-simulator/carla/issues/3493
[2] Accessed 2024-02-27. *CARLA changelog.* https://github.com/carla-simulator/carla/blob/master/CHANGELOG.md

[3] Accessed 2024-19-03. CARLA Autonomous Driving Leaderboard. https://leaderboard.carla.org

[4] Accessed 2024-19-03. CARLA simulator. https://github.com/carla-simulator/carla

[5] Aitor Arrieta, Jon Ayerdi, Miren Illarramendi, Aitor Agirre, Goiuria Sagardui, and Maite Arratibel. 2021. Using machine learning to build test oracles: an industrial case study on elevators dispatching algorithms. In *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*. IEEE, 30–39.

[6] George K. Baah, Andy Podgurski, and Mary Jean Harrold. 2011. Mitigating the Confounding Effects of Program Dependences for Effective Fault Localization. In *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering* (Szeged, Hungary) *(ESEC/FSE '11)*. Association for Computing Machinery, New York, NY, USA, 146–156. https://doi.org/10.1145/2025113.2025136

[7] George K. Baah, Andy Podgurski, and Mary Jean Harrold. 2010. Causal Inference for Statistical Fault Localization. In *Proceedings of the 19th International Symposium on Software Testing and Analysis* (Trento, Italy) *(ISSTA '10)*. Association for Computing Machinery, New York, NY, USA, 73–84. https://doi.org/10.1145/1831708.1831717

[8] Zhuofu Bai, Gang Shu, and Andy Podgurski. 2015. NUMFL: Localizing Faults in Numerical Software Using a Value-Based Causal Model. In *2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 1–10. https://doi.org/10.1109/ICST.2015.7102597

[9] Elias Bareinboim and Judea Pearl. 2016. Causal inference and the data-fusion problem. *Proceedings of the National Academy of Sciences of the United States of America* 113, 27 (2016), 7345–7352.

[10] Earl T. Barr, Mark Harman, Phil McMinn, Muzammil Shahbaz, and Shin Yoo. 2015. The Oracle Problem in Software Testing: A Survey. *IEEE Transactions on Software Engineering* 41, 5 (2015), 507–525. https://doi.org/10.1109/TSE.2014.2372785

[11] Boyuan Chen and Zhen Ming (Jack) Jiang. 2021. A Survey of Software Log Instrumentation. *Comput. Surveys* 54, 4 (may 2021), 1–34. https://doi.org/10.1145/3448976

[12] Tsong Y. Chen, Shing C. Cheung, and Shiu Ming Yiu. 1998. *Metamorphic testing: A new approach for generating next test cases*. Technical Report HKUST-CS98-01. The Hong Kong University of Science and Technology.

[13] Yuqi Chen, Christopher M. Poskitt, Jun Sun, Sridhar Adepu, and Fan Zhang. 2020. Learning-guided network fuzzing for testing cyber-physical system defences. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering (ASE '19)*. IEEE Press, 962–973. https://doi.org/10.1109/ASE.2019.00093

[14] Kwang Ting Cheng and A. S. Krishnakumar. 1993. Automatic functional test generation using the extended finite state machine model. In *Proceedings of the 30th International Design Automation Conference* (Dallas, Texas, USA) *(DAC '93)*. Association for Computing Machinery, New York, NY, USA, 86–91. https://doi.org/10.1145/157485.164585

[15] Andrew G. Clark, Michael Foster, Benedikt Prifling, Neil Walkinshaw, Robert M. Hierons, Volker Schmidt, and Robert D. Turner. 2023. Testing Causality in Scientific Modelling Software. *ACM Trans. Softw. Eng. Methodol.* 33, 1, Article 10 (nov 2023), 42 pages. https://doi.org/10.1145/3607184

[16] Andrew G. Clark, Michael Foster, Neil Walkinshaw, and Robert M. Hierons. 2023. Metamorphic Testing with Causal Graphs. In *2023 IEEE Conference on Software Testing, Verification and Validation (ICST)*. 153–164. https://doi.org/10.1109/ICST57152.2023.00023

[17] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning (Proceedings of Machine Learning Research, Vol. 78)*. PMLR, 1–16.

[18] Michael Foster, Christopher Wild, Robert M. Hierons, and Neil Walkinshaw. 2024. Causal Test Adequacy. In *2024 IEEE Conference on Software Testing, Verification and Validation (ICST)*. IEEE, 161–172. https://doi.org/10.1109/icst60714.2024.00023

[19] R.S. Freedman. 1991. Testability of software components. *IEEE Transactions on Software Engineering* 17, 6 (1991), 553–564. https://doi.org/10.1109/32.87281

[20] Carlo A Furia, Richard Torkar, and Robert Feldt. 2023. Towards causal analysis of empirical software engineering data: The impact of programming languages on coding competitions. *ACM Transactions on Software Engineering and Methodology* 33, 1 (2023), 1–35.

[21] 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 (2024), 35 pages. https://doi.org/10.1145/3635709

[22] Luca Giamattei, Antonio Guerriero, Roberto Pietrantuono, and Stefano Russo. 2025. Causal reasoning in Software Quality Assurance: A systematic review. *Information and Software Technology* 178 (Feb. 2025), 107599. https://doi.org/10.1016/j.infsof.2024.107599

[23] Clark Glymour, Kun Zhang, and Peter Spirtes. 2019. Review of causal discovery methods based on graphical models. *Frontiers in genetics* 10 (2019), 524.

[24] Ross Gore and Paul F. Reynolds. 2012. Reducing confounding bias in predicate-level statistical debugging metrics. In *2012 34th International Conference on Software Engineering (ICSE)*. IEEE, 463–473. https://doi.org/10.1109/ICSE.2012.6227169

[25] Ralph Guderlei and Johannes Mayer. 2007. Statistical Metamorphic Testing Testing Programs with Random Output by Means of Statistical Hypothesis Tests and Metamorphic Testing. In *Seventh International Conference on Quality Software (QSIC 2007)*. 404–409. https://doi.org/10.1109/QSIC.2007.4385527

[26] Jia Cheng Han and Zhi Quan Zhou. 2020. Metamorphic Fuzz Testing of Autonomous Vehicles. In *Proceedings of the IEEE/ACM 42nd International Conference on Software Engineering Workshops* (Seoul, Republic of Korea) *(ICSEW'20)*. Association for Computing Machinery, New York, NY, USA, 380–385. https://doi.org/10.1145/3387940.3392252

[27] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Efficient Online Testing for DNN-Enabled Systems Using Surrogate-Assisted and Many-Objective Optimization. In *Proceedings of the 44th International Conference on Software Engineering (ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 811–822. https://doi.org/10.1145/3510003.3510188

[28] D.P. Helmbold and C.E. McDowell. 1996. A Taxonomy of Race Conditions. *J. Parallel and Distrib. Comput.* 33, 2 (1996), 159–164. https://doi.org/10.1006/jpdc.1996.0034

[29] Miguel A Hernán and James M Robins. 2020. *Causal Inference: What if*. Chapman & Hall/CRC, Boca Raton.

[30] Bernhard Jaeger, Kashyap Chitta, and Andreas Geiger. 2023. Hidden Biases of End-to-End Driving Models. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE. https://doi.org/10.1109/iccv51070.2023.00757

[31] Zhengmin Jiang, Jia Liu, Peng Sun, Ming Sang, Huiyun Li, and Yi Pan. 2024. Generation of Risky Scenarios for Testing Automated Driving Visual Perception Based on Causal Analysis. *IEEE Transactions on Intelligent Transportation Systems* 25, 11 (2024), 15991–16004. https://doi.org/10.1109/TITS.2024.3421343

[32] Brittany Johnson, Yuriy Brun, and Alexandra Meliou. 2020. Causal testing: understanding defects' root causes. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 87–99.

[33] Prabhjot Kaur, Samira Taghavi, Zhaofeng Tian, and Weisong Shi. 2021. A Survey on Simulators for Testing Self-Driving Cars. In *2021 Fourth International Conference on Connected and Autonomous Driving (MetroCAD)*. 62–70. https://doi.org/10.1109/MetroCAD51599.2021.00018

[34] Seongmin Lee, Dave Binkley, Robert Feldt, Nicolas Gold, and Shin Yoo. 2021. Causal program dependence analysis. *arXiv preprint arXiv:2104.09107* (2021).

[35] Daniel Malinsky and David Danks. 2018. Causal discovery algorithms: A practical guide. *Philosophy Compass* 13, 1 (2018), e12470.

[36] K. John McConnell and Stephan Lindner. 2019. Estimating treatment effects with machine learning. *Health Services Research* 54, 6 (oct 2019), 1273–1282. https://doi.org/10.1111/1475-6773.13212

[37] Changhai Nie and Hareton Leung. 2011. A survey of combinatorial testing. *ACM Comput. Surv.* 43, 2, Article 11 (feb 2011), 29 pages. https://doi.org/10.1145/1883612.1883618

[38] P. Nitsche, R.H. Welsh, A. Genser, and P.D. Thomas. 2018. A novel, modular validation framework for collision avoidance of automated vehicles at road junctions. In *2018 21st International Conference on Intelligent Transportation Systems (ITSC)*. 90–97. https://doi.org/10.1109/ITSC.2018.8569631

[39] Justin Norden, Matthew O'Kelly, and Aman Sinha. 2019. Efficient Black-box Assessment of Autonomous Vehicle Safety. https://doi.org/10.48550/ARXIV.1912.03618

[40] Sheila F O'Brien and Qi Long Yi. 2016. How do I interpret a confidence interval? *Transfusion* 56, 7 (2016), 1680–1683.

[41] Judea Pearl. 2009. *Causality*. Cambridge university press, Cambridge.

[42] Andy Podgurski and Yiğit Küçük. 2020. CounterFault: Value-Based Fault Localization by Modeling and Predicting Counterfactual Outcomes. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 382–393.

[43] Christopher M. Poskitt, Yuqi Chen, Jun Sun, and Yu Jiang. 2023. Finding Causally Different Tests for an Industrial Control System. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. 2578–2590. https://doi.org/10.1109/ICSE48619.2023.00215

[44] Paul Ralph et al. 2021. Empirical Standards for Software Engineering Research. arXiv:2010.03525 [cs.SE]

[45] Per Runeson, Martin Host, Austen Rainer, and Bjorn Regnell. 2012. *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons.

[46] Gang Shu, Boya Sun, Andy Podgurski, and Feng Cao. 2013. Mfl: Method-level fault localization with causal inference. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation*. IEEE, IEEE, 124–133.

[47] Julien Siebert. 2023. Applications of statistical causal inference in software engineering. *Information and Software Technology* 159 (jul 2023), 107198. https://doi.org/10.1016/j.infsof.2023.107198

[48] Dan Siroker and Pete Koomen. 2015. *A/B testing: The most powerful way to turn clicks into customers*. John Wiley & Sons.

[49] Jian Sun, He Zhang, Huajun Zhou, Rongjie Yu, and Ye Tian. 2022. Scenario-Based Test Automation for Highly Automated Vehicles: A Review and Paving the Way for Systematic Safety Assurance. *IEEE Transactions on Intelligent Transportation*

Systems 23, 9 (2022), 14088–14103. https://doi.org/10.1109/TITS.2021.3136353

[50] Shuncheng Tang, Zhenya Zhang, Yi Zhang, Jixiang Zhou, Yan Guo, Shuang Liu, Shengjian Guo, Yan-Fu Li, Lei Ma, Yinxing Xue, and Yang Liu. 2023. A Survey on Automated Driving System Testing: Landscapes and Trends. *ACM Trans. Softw. Eng. Methodol.* 32, 5, Article 124 (jul 2023), 62 pages. https://doi.org/10.1145/3579642

[51] C. Caldiera V. Basili and D. H. Rombach. 1994. *Goal question metric paradigm.* Vol. 2. Wiley. 528–532 pages.

[52] Clarice R. Weinberg. 2007. Can DAGs Clarify Effect Modification? *Epidemiology* 18, 5 (sep 2007), 569–572. https://doi.org/10.1097/ede.0b013e318126c11d

[53] S Wright. 1920. The Relative Importance of Heredity and Environment in Determining the Piebald Pattern of Guinea-Pigs. *Proc Natl Acad Sci U S A* 6, 6 (jun 1920), 320–332.

[54] Penghao Wu, Xiaosong Jia, Li Chen, Junchi Yan, Hongyang Li, and Yu Qiao. 2022. Trajectory-guided Control Prediction for End-to-end Autonomous Driving: A Simple yet Strong Baseline. arXiv:2206.08129 [cs.CV]

[55] Xinhai Zhang, Jianbo Tao, Kaige Tan, Martin Törngren, José Manuel Gaspar Sánchez, Muhammad Rusyadi Ramli, Xin Tao, Magnus Gyllenhammar, Franz Wotawa, Naveen Mohan, Mihai Nica, and Hermann Felbinger. 2023. Finding Critical Scenarios for Automated Driving Systems: A Systematic Mapping Study. *IEEE Transactions on Software Engineering* 49, 3 (2023), 991–1026. https://doi.org/10.1109/TSE.2022.3170122

[56] Ziyuan Zhong, Yun Tang, Yuan Zhou, Vania de Oliveira Neves, Yang Liu, and Baishakhi Ray. 2021. A Survey on Scenario-Based Testing for Automated Driving Systems in High-Fidelity Simulation. https://doi.org/10.48550/ARXIV.2112.00964

[57] Tahereh Zohdinasab, Vincenzo Riccio, Alessio Gambi, and Paolo Tonella. 2023. Efficient and Effective Feature Space Exploration for Testing Deep Learning Systems. *ACM Trans. Softw. Eng. Methodol.* 32, 2, Article 49 (mar 2023), 38 pages. https://doi.org/10.1145/3544792