
Proceedings Paper:
Devlin, L. and Foster, M. orcid.org/0000-0001-8233-9873 (2025) Evolving estimation models for causal testing. In: FSE Companion '25: Proceedings of the 33rd ACM International Conference on the Foundations of Software Engineering. 33rd ACM International Conference on the Foundations of Software Engineering (FSE 2025), 23-27 Jun 2025, Trondheim, Norway. Association for Computing Machinery (ACM) , pp. 1394-1401. ISBN 9798400712760

https://doi.org/10.1145/3696630.3731613

Evolving Estimation Models for Causal Testing

Luca Devlin
ldevlin1@sheffield.ac.uk
University of Sheffield
Sheffield, UK

Michael Foster
m.foster@sheffield.ac.uk
University of Sheffield
Sheffield, UK

Abstract

Causal reasoning is a promising and increasingly popular approach for testing complex software systems that cannot be tested using conventional approaches. This involves using a causal model and previous execution data to estimate and validate causal relationships between variables. To produce accurate causal estimates and reliable test outcomes, current approaches rely on users to specify the equational relationships between variables or sacrifice explainability for automation by using black-box estimation. In this paper, we present a hybrid between genetic programming and linear regression to automatically infer human-readable non-linear equations that can be used to evaluate causal test cases. Our results show that our technique tends to produce more accurate causal estimates and more reliable test outcomes than either technique used in isolation.

CCS Concepts

• **Software and its engineering → Software verification and validation**.

Keywords

Causal Testing, Causal Estimation, Genetic Programming, Linear Regression

ACM Reference Format:
Luca Devlin and Michael Foster. 2025. Evolving Estimation Models for Causal Testing. In *33rd ACM International Conference on the Foundations of Software Engineering (FSE Companion '25), June 23–28, 2025, Trondheim, Norway*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3696630.3731613

1 Introduction

Complex software systems are increasingly being applied in critical applications such as computational modelling, autonomous driving, and cyber-physical systems. Such systems tend to exhibit several fundamental characteristics that make them difficult to test using traditional techniques, notably uncontrollable behaviour, nondeterminism, and extremely complex relationships between inputs and outputs. This makes it difficult to predict the output of systems prior to running them, and to judge whether the observed behaviour is actually correct. Furthermore, these systems tend to have long runtimes and high computational cost, which limits the number of tests that can be feasibly executed.

Causal reasoning is increasingly being applied to address these testing challenges [8]. The idea is to specify a model of the expected causal relationships between variables and use statistical estimation techniques to validate these relationships. Critically, the model is used to automatically identify which variables need to be *adjusted for* in order to remove bias when estimating causal effects. This enables the expected causal effects to be validated using pre-existing runtime data rather than requiring a specially curated dataset, without risking the causal effect estimates and subsequent test outcomes being biased by the data generation process.

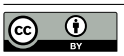
However, we may still get inaccurate causal effect estimates if an unsuitable estimation technique is used. For example, the Causal Testing Framework [5] provides an automated suite of causal testing tools and supports a range of estimation techniques. The default technique is standard linear regression, which implicitly assumes that the variables under investigation are related linearly and do not interact with each other. If this is not the case, the causal effect estimates and resulting test outcomes will not be reliable: tests may fail without a fault, or worse, may leave faults undetected. While this can be mitigated by using more complex regression equations, the tester must specify these equations themselves, which requires detailed and extensive knowledge of the system under test. Given the complex and often exploratory nature of the systems to which this kind of testing is intended to be applied [5], testers may not have sufficient knowledge to do this.

While black-box estimation techniques can automatically learn complex relationships between variables, their results are not explainable, and they tend to require large amounts of training data. In this paper, we present a hybrid approach between genetic programming and linear regression to automatically discover non-linear relationships and interaction between variables, without sacrificing explainability. Our main contributions are as follows:

- A technique based on genetic programming and linear regression for discovering the relationships between variables in a dataset.
- An extension to the Causal Testing Framework [5] which implements this technique.
- An experimental evaluation of our technique consisting of 300 randomly generated expressions, and 2 causal test cases for real computational models taken from [5].

The remainder of this paper is structured as follows. Section 2 introduces a motivating example and necessary background for our work. Section 3 presents our approach. Section 4 discusses our empirical evaluation. Section 5 summarises related work. Finally, Section 6 concludes the paper and gives ideas for future research.

Foster was supported by EPSRC CITCoM grant [EP/T030526/1].



FSE Companion '25, Trondheim, Norway

ACM ISBN 979-8-4007-1276-0/2025/06
https://doi.org/10.1145/3696630.3731613

2 Background

This section discusses the necessary background for this work. We first present a motivating example, and then use this to introduce the techniques of causal testing and genetic programming.

2.1 Motivating Example

Covasim [13] is an agent-based epidemiological model that was used to inform COVID-19 policy decisions in several countries. However, Covasim has several characteristics that make it particularly challenging to test using conventional techniques. Firstly, there are 64 unique input parameters, many of which are complex objects or statistical distributions with their own parameters. Secondly, due to its exploratory nature, Covasim suffers from the oracle problem [3]; the expected behaviour of many modelling scenarios is unknown. To compound this issue, Covasim is also nondeterministic: its outputs differ between multiple runs with the same inputs. This makes it hard to associate particular outcomes with individual input parameters, meaning that properties must be evaluated statistically over several runs.

A popular solution to these problems, when testing models like Covasim, is to test that *changing* an input in a particular way leads to a corresponding *change* in the output [12]. For example, in Covasim, we might test that increasing the infectiousness (β) of the disease leads to an increase in the overall number of infections, assuming all other parameters are kept the same. Statistical Metamorphic Testing [9] offers a framework for this using statistical tests on repeated runs of multiple configurations. Unfortunately, this tends to require many repeats of each of the configurations of interest in order to give reliable results, which is simply not feasible for Covasim due to its long runtimes and high computational cost.

2.2 Causal Testing

Causal Testing [5] is a software testing methodology based on Causal Inference [16]. Here, a causal model and statistical estimator are used to validate causal properties — like the effect of β on infections — using existing runtime data rather than the bespoke, highly controlled test data required for Statistical Metamorphic Testing. The Causal Testing Framework (CTF) [5] provides a suite of automated tools for Causal Testing. The Causal Testing workflow requires three main inputs from the tester: a causal model, causal test cases, and test data with which to evaluate the test cases.

The causal model takes the form of a directed acyclic graph (DAG) where nodes represent variables and edges $X \rightarrow Y$ represent X having a direct causal affect on Y. An example for Covasim is shown in Figure 1. As with any model-based testing technique, causal DAGs require domain knowledge to draw, but are much lighter weight than traditional models like finite state machines.

Causal test cases describe the expected causal effect of a *treatment variable* on an *outcome variable*. This can be either qualitative (e.g. increasing β should increase the number of infections) or quantitative (e.g. increasing β from 0.016 to 0.032 should increase the number of infections by 1000) depending on the tester's knowledge of the system under test. Where very little is known about the expected nature of the specified causal relationships, a causal DAG can be automatically converted to a set of causal tests to validate the presence and absence of the specified causal relationships [6].

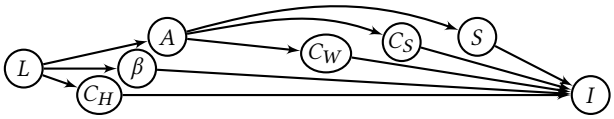


Figure 1: Causal DAG for Covasim testing the causal effect of β on infections I in location L with average population age A, susceptibility S, and household, school, and workplace contacts C_H, C_S, C_W respectively. Reproduced from [5, Figure 7].

The CTF then uses Causal Inference to automatically evaluate the causal test cases using the provided DAG and runtime data, which is simply a table with rows representing software runs and columns representing the variables in the DAG. This has two steps: *identification* and *estimation*. First, the causal DAG is used to *identify* which variables need to be adjusted for to estimate the causal effect without bias. For example, in Figure 1, the effect of β on I is confounded by the location L, which determines β and several other factors which all have causal paths to I. This must be adjusted for by controlling for either L or all of these other factors, otherwise what looks like a direct causal effect of β on I could simply be a spurious correlation through L.

To estimate the causal effect, a statistical model is fitted to the data. Critically, this estimation model is only fitted to features identified from the DAG in the previous step and not every feature in the data, as traditional Machine Learning approaches tend to do. If the estimated causal effect matches the expected causal effect, then the test case passes, otherwise it fails.

When evaluating causal tests, the CTF performs standard linear regression of the outcome on the identified features, unless the tester specifies a more complex estimator. However, this will not give accurate causal effect estimates or reliable test outcomes if the variables are not linearly related, or if they interact with each other. For complex models like Covasim, that are often employed in an exploratory manner, testers may not precisely know these relationships in advance [12], so are unlikely to be able to provide the complex functions necessary to obtain reliable causal estimates. Black-box Machine Learning estimation could be used instead, but this lacks explainability, and may require much more data than is typically available. In Section 3, we present an approach to automatically infer complex regression equations from the data that testers can then validate and refine.

2.3 Genetic Programming

The technique we present in Section 3 is based on Genetic Programming (GP) [14]. In tree-based GP, functions are represented as syntax trees where branch nodes represent operators, and leaf nodes represent variables and constants. GP uses a genetic algorithm to evolve a population of candidate functions over a series of generations in a manner inspired by natural evolution.

The basic $\mu + \lambda$ loop is shown in Algorithm 1. We begin with a population of μ random individuals. Each generation, these are recombined and mutated to form λ new individuals, with only the μ fittest individuals continuing on to the next generation. This means that the population size remains constant between iterations, and the average fitness of individuals in the population improves as GP

progresses. Here, an expression's *fitness* is an aggregation of the distances between the predicted outputs and the observed outputs for each input in the data.

Algorithm 1 Basic $\mu + \lambda$ genetic programming algorithm.

1: Generate an initial population of μ random expressions
2: Evaluate each expression according to the fitness function
3: gen = 0
4: **while** gen < maxGens ∧ optimal individual not found **do**
5: gen++
6: Create λ new individuals by recombining and mutating expressions from the population
7: Evaluate each expression according to the fitness function
8: Select the fittest μ expressions for the next generation

One problem with GP is that it can struggle to discover the best constants and coefficients. This is because the problem is continuous rather than combinatorial in nature. Our approach in Section 3 uses a hybrid between linear regression and GP to tackle this.

3 Combining GP and Linear Regression

This section presents our technique to automatically learn equational relationships between variables from data. We combine GP to learn the "shape" of the equation and linear regression to learn the constants. While this concept in itself is not new [11], it has not yet been applied in a CI context to estimate causal effects. To this end, we still only consider the variables from the causal identification step rather than every variable in the dataset so that our equations produce an unbiased estimate. Furthermore, our technique is able to work with arbitrarily complex operators and expressions, where [11] only considers polynomials.

Our approach follows the same basic GP structure outlined in Algorithm 1. The only difference is that we introduce a *repair operator*, which is run on the initial population and each new individual before it enters the population. Algorithm 2 outlines our repair operator. The candidate function is first split into its top level linear components (line 2). For example, the function $y = \log(x_1) + \sin(x_1 + x_2) + x_2^3$ has three components: $\log(x_1)$, $\sin(x_1 + x_2)$, and x_2^3, plus an implicit constant term. The second step (line 3) uses linear regression to infer the best coefficients for each component. Finally, the coefficients and components are combined to form the repaired equation (line 4). For example, the repaired equation above may be $y = 5.3\log(x_1) + 1.2\sin(x_1 + x_2) + 2.7x_2^3 + 8$. An investigation into a recursive application of this to handle deeper non-linear components is left for future work.

Algorithm 2 Linear regression repair operator.

1: **function** REPAIROPERATOR(equation, trainingData)
2: components ← LINEARCOMPONENTS(equation)
3: coefficients ← LEARNCOEFFICIENTS(components, data)
4: **return** $\sum_i(\text{coefficients}_i \times \text{components}_i)$

We implemented our technique using the Python DEAP framework [7]. Our implementation is available as part of the CTF in version 8.0.0 and subsequent releases. By default, our implementation uses a population size $\mu = 20$ and generates $\lambda = 10$ new individuals per generation. Each new individual is generated by recombining two parents, selected via binary tournament selection, which has been shown to perform well on a number of problems [15]. Each new individual is then mutated using one of the three mutation operators implemented DEAP, chosen uniformly at random. These are described as follows:

- **mutUniform** – replaces a randomly selected subtree with a newly generated random subtree.
- **mutNodeReplacement** – replaces a randomly chosen operator in the individual with a new operator of the same arity (e.g. replacing + with −).
- **mutShrink** – replaces a randomly selected subtree with one of its child nodes, effectively reducing the tree size.

Unlike the approach from [11], which only considers polynomial functions, our approach supports arbitrarily complex operations and functional forms, and is fully customisable in this respect. By default, we consider logs, reciprocals, and interaction terms (i.e. terms of the form x_1x_2) when evolving functions. Users can also specify the maximum polynomial order to be considered, and supply a list of additional operators, which may include any python function (including user-defined functions). We leave an investigation into the accuracy and scalability of our tool with respect to the operators used for future work. However, where users do not know the relevant operators, they can use polynomial approximation [17] by iteratively increasing the degree until a suitably accurate fit to the training data is obtained.

4 Evaluation

This section describes our experimental evaluation comparing the performance of our technique to GP and linear regression in isolation. The aim is to infer equations that lead to accurate causal effect estimates and reliable test outcomes. Our research questions are as follows:

RQ1 (Accuracy) What factors affect the ability of our approach to accurately model causal relationships?
RQ2 (Test Outcomes) Do the functions fitted by our technique lead to reliable causal test outcomes?
RQ3 (Practicality) Is our technique practical to apply in a realistic setting?

4.1 Experimental Setup

Our evaluation consists of two studies. The first addresses RQ1 by investigating how the number of variables, the amount of data, and the amount of noise within the data affect the accuracy of our approach, using randomly generated synthetic datasets. Our second study addresses RQ2 by using our technique to perform causal testing on two real computational models. We answer RQ3 as a meta-analysis of both studies. All our code and experimental data are available in our replication package[1].

[1]Replication package available at https://github.com/Luca0414/SURE-project

In both studies, we compare our technique to two baseline approaches: standard linear regression and GP without our regression-based repair operator. We chose these baselines since standard linear regression is the current default behaviour of the CTF, and our approach extends GP. We do not consider state-of-the-art black-box Machine Learning techniques here as they do not produce human-readable equations, thereby removing the possibility of manual inspection and validation by the user.

We ran both our approach and baseline GP for 100 generations, and used the default configuration values of $\mu = 20$ and $\lambda = 10$. Of course, there may be configurations which lead to better outcomes. We discuss this threat to validity further in Section 4.3. Additionally, we seeded the baseline standard linear regression equation into the initial population of both techniques, meaning that their final result is guaranteed to fit the data at least as well.

4.1.1 Study 1: Exploring accuracy for random expressions. We began by generating 30 random expressions containing each number of variables between 1 and 10, to give 300 expressions in total. Our expressions took the form $\sum_{i=1}^{N} f_i(x_i) + c$ where c is a constant, and $f_i(x_i)$ is an expression that references x_i and may reference other variables. Each $f_i(x_i)$ is built by randomly combining variables with the operators +, ×, sin, cos, tan, log, reciprocal, square, cube, and square root[2], and gave our approach and baseline GP access to all of these operators. An investigation into how our technique behaves when it is *not* given access to all of the operations used by the system under test is desirable future work.

For each expression, we generated a dataset of 1000 executions with inputs sampled independently from $\mathbf{U}(0, 100)$. We then generated two noisy datasets by adding a noise term sampled from $\mathbf{U}(-y\epsilon, y\epsilon)$ to each output y in the data for $\epsilon = 0.1$ and $\epsilon = 0.25$. The original dataset represents $\epsilon = 0$. We ran our approach with each dataset and a random sample of 10, 50, 100, and 500 data points. To test the prediction of *counterfactual outcomes* — outcomes that correspond to inputs not in the original dataset — we generated an additional 100 such inputs from $\mathbf{U}(0, 100)$.

We measure accuracy in terms of the normalised root mean square error (NRMSE) between the reference outputs Y from the observed dataset and the values X predicted by the fitted equation. NRMSE is defined as $\text{RMSE}/\max(Y) - \min(Y)$, where RMSE is defined as $\sqrt{\sum_{i=1}^{n} (x_i - y_i)^2 / n}$, where n is the number of data points in X and Y. A lower NRMSE indicates a lower error and better performance. Typically, the NRMSE gives a value between 0 and 1, however, if the values X predicted by the fitted equation fall well outside of the range of Y, the value can be arbitrarily high.

4.1.2 Study 2: Causal testing of computational models. To investigate how our approach fits into the broader workflow of Causal Testing described in Section 2, we applied our technique to evaluate two causal properties published by Clark et al. [5], which form part of the tutorial for the CTF along with the data necessary to evaluate them. Our first causal property revisits the motivating example from Section 2 and studies the effect of β on the total number of infections in Covasim. We define one causal test case for each of the 156 countries supported by Covasim, and use the gold standard established by Clark et al. [5] as our expected causal effect. Using the hand-crafted regression equation shown in Equation (1) (with two internal splines on β), Clark et al. [5] could estimate the change in infections to within a 5.5% margin of error for the vast majority of countries.

$$\begin{aligned} I \sim{}& \log S + \log(S)^2 + \log C_W + \log {C_W}^2 + \\ & \log C_S + \log {C_S}^2 + \log C_H + \log {C_H}^2 + \\ & \beta \log C_W + \beta \log C_S + \beta \log C_H + \beta \log S + \beta + c \end{aligned} \tag{1}$$

Our second property concerns a Poisson line tessellation model, which generates lines positioned and oriented at random within a sampling window of a given width W and height H to form a tessellation of polygons. The specified intensity I of the process controls the number of lines that are drawn. For a given sampling window, doubling the intensity should increase the number of polygons per unit area by a factor of 4. Using the hand-crafted regression equation shown in Equation (2), Clark et al. [5] tested this property with $W = H$ for integers 1 to 10 and intensities 1 to 2, 2 to 4, 4 to 8, and 8 to 16, giving a total of 40 test cases. They found that the increase was actually slightly less than a factor of 4 when lower values of I were used with small sampling windows.

$$P_u \sim W + W^{-1} + I + I^2 \tag{2}$$

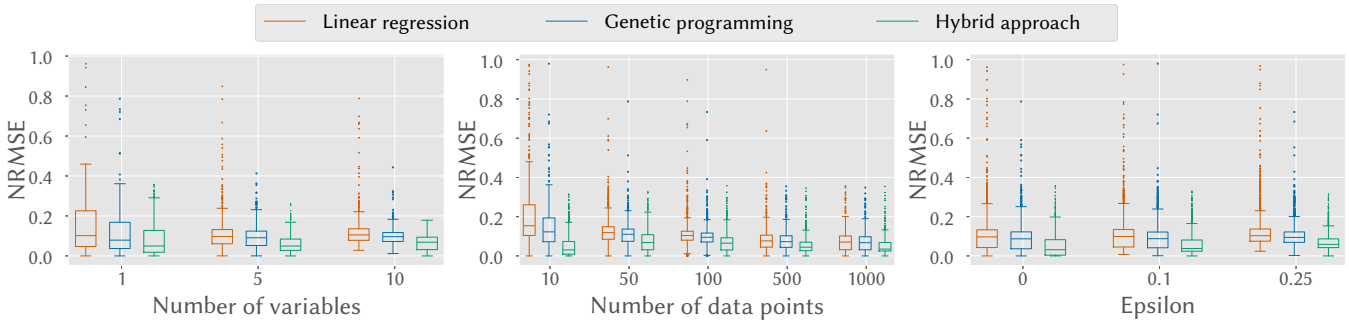
We used the causal effect estimates and test outcomes from [5] as a gold standard to which we compared our technique and the two baseline approaches. We measured the reliability of test outcomes using the balanced classification rate (BCR), defined as $2 \times \text{sensitivity} \times \text{specificity} / \text{sensitivity} + \text{specificity}$. This gives the harmonic mean of the sensitivity (the proportion of tests which should pass that do pass) and the specificity (the proportion of tests which should fail that do fail), with a higher BCR indicating more reliable test outcomes. We use BCR, rather than simply the proportion of passing tests, since some of our test cases are expected to fail due to known inconsistencies. That is, the small proportion of Covasim countries that Clark et al. [5] could not estimate to within a 5.5% margin of error, and the small sampling windows of Poisson for which doubling the intensity did not yield the expected increase in polygons per unit area.

4.2 Results and Discussion

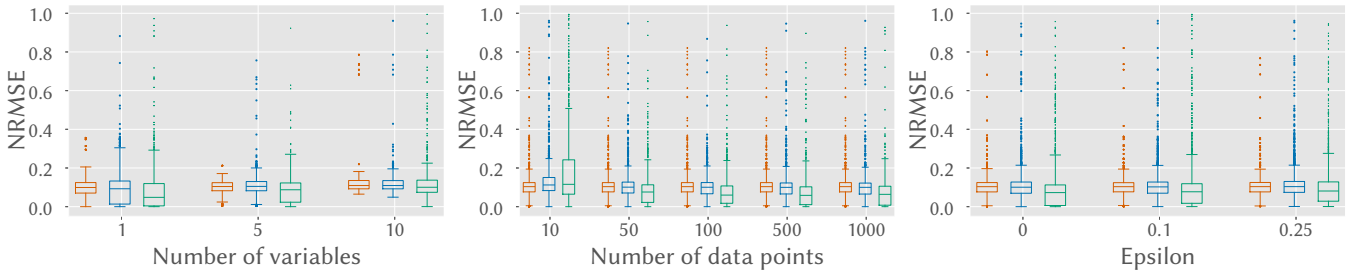
We now present the results of our studies. Our full dataset, processing code, and plots are available as part of our replication package.

4.2.1 RQ1 (Accuracy). Figure 2 shows how the number of variables, size of training dataset, and noise factor ϵ affect the NRMSE of both the fit and counterfactual predictions for our technique and the two baselines. In the interest of visual clarity, we only show the outcomes for 1, 5, and 10 variable equations, since the other variables tell a similar story. We also omit outliers with NRMSE greater than 1 (the highest typical value for NRMSE), as some of these were extremely large, resulting in distorted plots. For the linear regression plot in Figure 2a, we removed 246 of our 4500 total data points. In Figure 2b, we removed 270 linear regression outliers, 235 GP outliers, and 293 hybrid outliers.

[2]We omitted minus and divide to mitigate the risk of generating trivial expressions like $x_i - x_i$ and x_i/x_i that syntactically reference x_i but do not use it meaningfully. We omitted exponentials and powers higher than 3 to mitigate numerical overflow. This limits the generality of our results, but is a necessary mitigation when generating random expressions.



(a) Boxplots showing how the number of variables, size of dataset, and ϵ affect the fit to the training data.



(b) Boxplots showing how the number of variables, size of training dataset, and ϵ affect the counterfactual outcome predictions.

Figure 2: Graphs showing how the number of variables, size of dataset, and ϵ affect NRMSE.

Figure 2a shows that our hybrid approach typically achieves the best fit to the data, with a Friedman test, and subsequent post-hoc Nemenyi tests confirming the statistical significance (i.e. $p < 0.05$) of this. A Spearman R test reveals a very weak (although statistically significant) positive correlation between the number of variables and the NRMSE for all three techniques. That is, as the number of variables increases, the fit becomes very slightly worse. There is a similar weak correlation between ϵ and NRMSE. For the amount of data, the correlation is negative, indicating that the fit improves as we collect more training data. While this correlation is slightly stronger than the other two, it is still relatively weak.

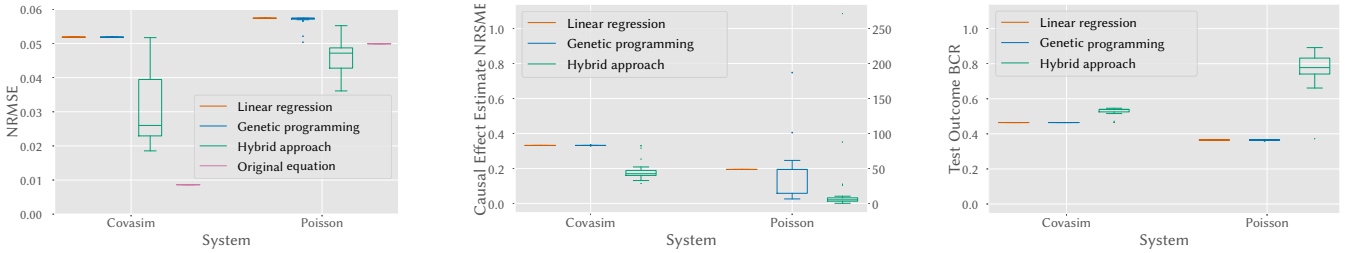
Figure 2b tells a similar story for counterfactual predictions, with Friedman and Nemenyi tests indicating that our technique tends to produce significantly better results. Spearman R tests indicate similar weak correlations for all three techniques in terms of number of variables, data size, and ϵ. This indicates that our technique is more robust to small datasets, noise and large numbers of variables than either of the baseline techniques. However, our technique did occasionally produce less accurate counterfactual predictions than the baselines, especially for the smallest datasets. When taken in conjunction with Figure 2a, this indicates that our technique may be more susceptible to overfitting than the baseline techniques.

Answer to RQ1: Our technique achieves a better fit to the data and a better predictive accuracy (i.e. a more accurate causal effect estimate) than the baseline techniques. Although it is slightly more susceptible to overfitting, our technique is generally more robust to small datasets, noise and large numbers of variables.

4.2.2 RQ2 (Test outcomes). Figure 3 shows how the three techniques performed when executing causal test cases for Covasim and the Poisson process. Our technique typically achieved the best fit to the test data, leading to more accurate causal effect estimates and more reliable test outcomes. Friedman and Nemenyi tests confirm the statistical significance of these relationships.

Figure 3a shows that no technique managed to fit the Covasim data as well as the original hand-crafted equation used by Clark et al. [5]. This is not surprising, since the original equation is extremely complex, and we did not include splines in the set of operations considered by either our technique or baseline GP. For Poisson, our technique typically managed to find an even better fit than the original equation used by Clark et al. [5]. In both cases, the baseline GP approach was rarely able to evolve an equation with a better fit than the seeded baseline regression equation, which seems to represent a local optimum. This explains its extremely narrow interquartile range. By contrast, our hybrid approach has a much wider interquartile range as it was often able to find a much better fitting equation, but occasionally, like the baseline GP, got stuck in the local optimum of the seeded baseline regression equation.

Figure 3b shows that our technique produced more accurate causal effect estimates than both baseline techniques. For the Poisson process model, neither of the baseline techniques was able to fit a model that gave an NRMSE in the typical [0, 1] range, indicating that these models did not fit the data well at all and gave predictions well outside of the observed range of values. The reason for this is that the Poisson process is highly non-linear and has a lot of interaction between variables, meaning that standard linear regression is not a suitable estimator here. While the baseline GP



(a) NRMSE of fit to the data.

(b) Causal effect estimate NRMSE with respect to the original estimates from [5].

(c) Test outcome BCR with respect to the original outcomes from [5]

Figure 3: Test outcomes for the causal tests from [5].

was able to evolve some of these characteristics into the population, the coefficients from our repair operator are critical here.

Figure 3c shows that the more accurate causal effect estimates from our technique translate to a higher BCR, indicating more reliable causal test outcomes. For Covasim, our technique achieved a BCR of just above 0.5. However, for the Poisson model, our technique typically achieved a BCR above 0.75 and up to 0.9 in some cases. This difference in performance is likely due to the fact that Covasim has much more complex relationships between more variables. There are also many more test cases to consider here: 156 in comparison to just 40 for Covasim.

Answer to RQ2: For both the Covasim and Poisson test cases, our technique produced equations that fit the data better than either of the baselines. This led to more accurate causal effect estimates and more reliable test outcomes.

4.2.3 RQ3 (Practicality). Looking back at Figure 2, while all three techniques scale quite well with respect to the number of variables, the amount of training data, and noisy data, our hybrid approach tends to scale better. As noted for RQ1, the exception to this is for datasets containing just 10 software runs, where our technique has a tendency to overfit the data. However, once our technique has sufficient data — somewhere between 10 and 50 datapoints in our evaluation — it tends to outperform the baseline approaches, even for relatively small datasets.

While a controlled benchmarking study is outside the scope of this evaluation, we did record the time taken by each technique to fit the data[3]. In all cases, baseline linear regression has a negligible runtime in comparison to the other two techniques. In most cases, our approach was slightly slower than baseline GP, which is explained by the extra step of our repair operator.

Figure 4 shows that the runtime of both baseline GP and our hybrid approach increases exponentially with the size of the data set, and Spearman R tests show a statistically significant strong positive correlation here. While this does limit the scalability of our approach, Figure 2 shows that we can typically obtain reasonably accurate fits and predictions even from relatively small datasets. Since causal testing is typically applied in situations where data collection is very expensive and time-consuming [5], it is likely that this would be the limiting factor rather than the fitting time.

[3]We ran our experiments on HPC using 4GB of memory and a single 2GHz CPU core per run.

Figure 4 also shows that runtime increases slightly with the number of variables. However, while Spearman R tests show that this correlation is statistically significant for both baseline GP and our technique, it is relatively weak. There is no statistically significant correlation between ϵ and runtime.

Figure 5 shows the runtime for the two causal test cases from study 2. Since the Poisson dataset contained 1000 data points, the results are comparable to Figure 4. For Covasim, both our technique and baseline GP took around 5 minutes to fit the data from [5], which contained over 4500 points. While this is a long time in comparison to the original regression used by Clark et al. [5], we still deem it to be acceptable given the amount of time and domain expertise required to formulate their equation.

Answer to RQ3: Although it took the longest, our approach still ran in a reasonable time, especially in the context of the amount of time and expertise required to hand-craft a regression equation.

4.3 Threats to Validity

The main threat to validity is that we only evaluated our technique on two real software systems. Consequently, our results may not generalise to other kinds of software systems. This threat is mitigated somewhat by the fact that the two real systems differ greatly in size, complexity, and application, but further investigation on a wider variety of systems and operations is desirable future work. In the same vein, our 300 randomly generated equations were constructed using a limited set of common mathematical operations. While this was necessary in order to ensure the feasibility of our evaluation, our results may not generalise to more complex equations. An investigation into how our technique performs with different operations is left to future work.

Another threat to validity is that the baseline GP and our hybrid approach are subject to configuration parameters that we did not spend time optimising. While this mitigates the internal threat to validity of overfitting the results to the systems considered here, there may be more suitable configurations, meaning the behaviour of the techniques is misrepresented. Without an extensive parameter-optimisation investigation, it is impossible to mitigate this.

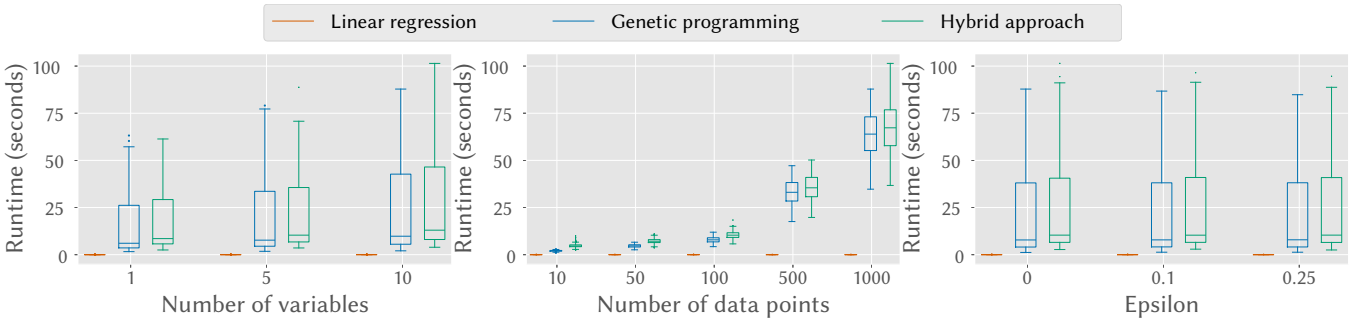


Figure 4: Boxplots showing how the number of variables, size of dataset, and ϵ affect the fitting time for the random datasets of study 1.

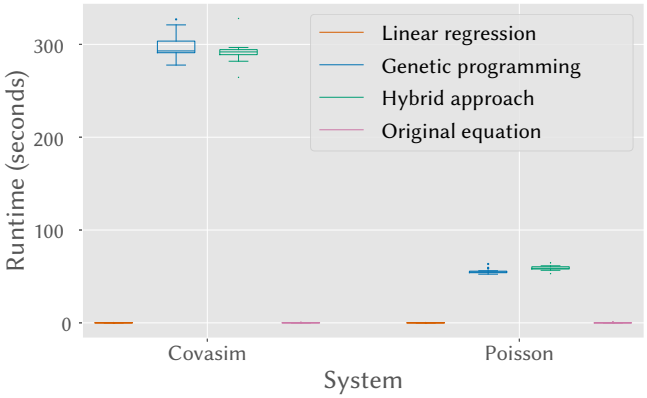


Figure 5: Fitting time for the causal test cases from study 2.

5 Related Work

There is a wealth of literature on the prediction of software outputs from existing runtime data using standard techniques such as linear regression [2], support vector regression [4], and ensemble models [10]. There are also several existing works that hybridise linear regression with GP. Iba et al. [11] use a repair operator similar to ours. However, they only consider polynomial expressions, and do so in a feedforward manner, meaning the order of components will affect their inferred coefficients. Our approach does not have these limitations. Arnaldo et al. [1] learn compound models that involve both symbolic GP execution and numeric multiple regression transforms. While this supports arbitrary operations, the coefficients for each component are not directly learnt in the same way.

6 Conclusions and Future Work

Causal testing is a promising and increasingly popular way to test complex software systems by using statistical models to estimate causal effects from pre-existing runtime data. For the most accurate and explainable results, users should ideally know the equational form of the relationships between the variables they are testing. However, this is often infeasible for the kinds of system that causal testing is typically applied to.

In this work, we presented a hybrid approach between GP and linear regression that is able to automatically fit non-linear equations to data and discover interactions between variables. We implemented this as an extension to the CTF [5] and evaluated it using 300 random target expressions and two existing causal test cases from real computational models in the literature. Our results show that our hybrid approach fits more accurate functions than either GP or linear regression alone and is more robust to noisy data, small datasets, and large numbers of variables.

Future work includes an investigation into the robustness of our technique to unknown operators, and a more comprehensive evaluation on more real systems with different characteristics. Furthermore, a study into how the inferred functions can be validated, and the burden on the user in comparison to hand-crafting functions themselves is highly desirable. Finally, the ability to perform more complex regression, such as spline regression, is highly desirable.

References

[1] Ignacio Arnaldo, Krzysztof Krawiec, and Una-May O'Reilly. 2014. Multiple regression genetic programming. In *Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation (GECCO)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/2576768.2598291

[2] 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 *International Conference on Automation of Software Test (AST)*. IEEE.

[3] 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). https://doi.org/10.1109/TSE.2014.2372785

[4] Yuqi Chen, Christopher M. Poskitt, Jun Sun, Sridhar Adepu, and Fan Zhang. 2020. Learning-guided network fuzzing for testing cyber-physical system defences. In *International Conference on Automated Software Engineering (ASE)*. IEEE. https://doi.org/10.1109/ASE.2019.00093

[5] 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 Transactions on Software Engineering and Methodology* 33, 1 (2023). https://doi.org/10.1145/3607184

[6] 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)*. https://doi.org/10.1109/ICST57152.2023.00023

[7] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner Gardner, Marc Parizeau, and Christian Gagné. 2012. DEAP: evolutionary algorithms made easy. *J. Mach. Learn. Res.* 13, 1 (2012).

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

[9] Ralph Guderlei and Johannes Mayer. 2007. Statistical Metamorphic Testing Testing Programs with Random Output by Means of Statistical Hypothesis Tests

and Metamorphic Testing. In *International Conference on Quality Software (QSIC)*. IEEE. https://doi.org/10.1109/QSIC.2007.4385527

[10] Fitash Ul Haq, Donghwan Shin, and Lionel Briand. 2022. Efficient Online Testing for DNN-Enabled Systems Using Surrogate-Assisted and Many-Objective Optimization. In *International Conference on Software Engineering (ICSE)*. ACM, New York, NY, USA. https://doi.org/10.1145/3510003.3510188

[11] Hitoshi Iba, Hugo deGaris, and Taisuke Sato. 1995. A Numerical Approach to Genetic Programming for System Identification. *Evolutionary Computation* 3, 4 (1995). https://doi.org/10.1162/evco.1995.3.4.417

[12] Upulee Kanewala and James M Bieman. 2013. Using machine learning techniques to detect metamorphic relations for programs without test oracles. In *International Symposium on Software Reliability Engineering (ISSRE)*. IEEE.

[13] Cliff C Kerr, Robyn M Stuart, Dina Mistry, Romesh G Abeysuriya, Katherine Rosenfeld, Gregory R Hart, Rafael C Núñez, Jamie A Cohen, Prashanth Selvaraj, Brittany Hagedorn, et al. 2021. Covasim: an agent-based model of COVID-19 dynamics and interventions. *PLOS Computational Biology* 17, 7 (2021).

[14] John R. Koza. 1992. *Genetic programming: On the programming of computers by means of natural selection*. MIT Press.

[15] Brad L Miller, David E Goldberg, et al. 1995. Genetic algorithms, tournament selection, and the effects of noise. *Complex systems* 9, 3 (1995).

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

[17] M. H. Stone. 1948. The Generalized Weierstrass Approximation Theorem. *Mathematics Magazine* 21, 4 (1948). https://doi.org/10.2307/3029750