Causality-Based Neural Network Repair

Bing Sun
Singapore Management University

Long H. Pham
Singapore Management University

Jun Sun
Singapore Management University

Jie Shi
Huawei Singapore

ABSTRACT

Neural networks have had discernible achievements in a wide range of applications. The wide-spread adoption also raises the concern of their dependability and reliability. Similar to traditional decision-making programs, neural networks can have defects that need to be repaired. The defects may cause unsafe behaviors, raise security concerns or unjust societal impacts. In this work, we address the problem of repairing a neural network for desirable properties such as fairness and the absence of backdoor. The goal is to construct a neural network that satisfies the property by (minimally) adjusting the given neural network’s parameters (i.e., weights). Specifically, we propose CARE (CAusality-based REpair), a causality-based neural network repair technique that 1) performs causality-based fault localization to identify the ‘guilty’ neurons and 2) optimizes the parameters of the identified neurons to reduce the misbehavior. We have empirically evaluated CARE on various tasks such as backdoor removal, neural network repair for fairness and safety properties. Our experiment results show that CARE is able to repair all neural networks efficiently and effectively. For fairness repair tasks, CARE successfully improves fairness by 61.91% on average. For backdoor removal tasks, CARE reduces the attack success rate from over 98% to less than 1%. For safety property repair tasks, CARE reduces the property violation rate to less than 1%. Results also show that thanks to the causality-based fault localization, CARE’s repair focuses on the misbehavior and preserves the accuracy of the neural networks.

ACM Reference Format:
Bing Sun, Jun Sun, Long H. Pham, and Jie Shi. 2022. Causality-Based Neural Network Repair. In 44th International Conference on Software Engineering (ICSE '22), May 21–29, 2022, Pittsburgh, PA, USA. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3510003.3510080

1 INTRODUCTION

Neural networks have had discernible achievements in a wide range of applications, ranging from medical diagnosis [66], facial recognition [53], fraud detection [19] and self-driving [6]. While neural networks are demonstrating excellent performance, there has been a growing concern on whether they are reliable and dependable. Similar to traditional decision-making programs, neural networks inevitably have defects and need to be repaired at times. Neural networks are usually inherently black-boxes and do not provide explanations on how and why decisions are made in certain ways. As a result, these defects are more “hidden” compared to traditional decision-making programs [5, 17, 23]. It is thus crucial to develop systematic ways to make sure defects in a neural network are repaired and desirable properties are satisfied.

Similar to the activity known as debugging for traditional software programs, there is often a need to modify a neural network to fix certain aspects of its behavior (whilst maintaining other functionalities). Existing efforts to repair the unexpected behavior of neural networks often focus on retraining with additional data [43, 44]. Although retraining is natural and often effective, retraining a neural network model could be difficult and expensive for real-world applications [23]. More importantly, unlike debugging traditional software program (where we can be reasonably certain that the bug is eliminated after the ‘fix’), there is no guarantee that the retrained model eliminates the unwanted behavior. Therefore, techniques for modifying an existing model without retraining will be preferable in certain scenarios. That is, we sometimes would like to apply a small modification on an existing neural network to remove unexpected behaviors whilst retaining most of its well-trained behaviors.

Neural networks may misbehave in different ways. Given a model trained primarily for accuracy, it could be discriminative, i.e., the prediction is more favourable to certain groups thus violating fairness property. In this situation, small adjustment can be applied to the trained network without retraining, i.e., to improve its fairness whilst maintaining the model’s accuracy. In another scenario, malicious hidden functionalities (backdoor) could be easily embedded into a neural network if the training data is poisoned [42, 67]. Such a backdoor produces unexpected behavior if a specific trigger is added to an input, whilst presenting normal behavior on clean inputs. In this scenario, one would like to repair the neural network by removing the backdoor while maintaining the correct behavior on clean inputs. A further scenario is that a trained model could violate safety-critical properties on certain inputs, e.g., the output is not within the expected range [33, 34]. In this case, we would like to repair the network by making adjustments to its parameters so that the repaired model satisfies the specified property.

For both traditional software programs and neural networks, debugging and repairing are essentially reasoning over causality. For traditional software programs, the notion of causality is natural and well defined [9, 28, 32], e.g., based on (control and data) dependency. However, the same is not true for neural networks, i.e., a wrong decision could easily be the collective result of all neurons in the network and yet considering that all of them are ‘responsible’ and should be modified is unlikely to be helpful. Although causality analysis has been applied to interpret machine learning models [8]
and verification [71], existing causality analysis typically focuses on the causal relation from the input to the model prediction, and not the hidden neurons. Hence it is still not clear how to apply existing definitions of causality for repairing neural networks. In other words, how to repair a neural network to remedy defects based on causality is still largely an open problem.

In this work, we introduce CARE (Causality-based REpair), a general automatic repair algorithm for neural networks. Instead of retraining, CARE applies minor modifications to a given model’s parameters (i.e., weights) to repair known defects whilst maintaining the model’s accuracy. The defects are defined based on desirable properties that the given neural network fails to satisfy. CARE is a search-based automated program repair technique that first performs fault localization based on causal attribution of each neuron, i.e., it locates those neurons that have the most contribution to the model’s defects. Secondly, CARE performs automatic repair by adjusting model parameters related to the identified neurons until the resultant model satisfies the specified property (whilst maintaining accuracy).

We summarize our contribution as follows.

- We propose and implement CARE as a general automatic repair algorithm, that applies causality-based fault localization and PSO optimization to all layers of neural networks.
- We demonstrate the effectiveness of CARE in the context of three different tasks: 1) fairness improvement, 2) backdoor removal and 3) safety property violation repair.
- We empirically evaluate CARE on multiple neural networks including feed-forward neural networks (FFNNs) and convolutional neural networks (CNNs), trained on multiple benchmark datasets. The results indicate that CARE improves the models’ performance over the specified properties significantly and CARE outperforms existing approaches proposed for neural network repair.

The remainder of this paper is organized as follows. In Section 2, we review relevant background and define our problem. In Section 3, we present each step of our approach in detail. In Section 4, we evaluate our approach through multiple experiments to answer multiple research questions. We review related work in Section 5 and conclude in Section 6.

2 PRELIMINARY

In this section, we review relevant background and define our research problem.

2.1 Neural Network Properties

There are many desirable properties we would like a trained neural network to satisfy besides meeting its accuracy requirement. In this work, we assume \( \phi \) to be one of the following three kinds of properties and show that CARE can handle all of them.

**Fairness:** Fairness is a desirable property that potentially has significant societal impact [47]. Due to the fact that machine learning models are data-driven and the training data could be biased (or imbalanced), models trained based on such data could be discriminative [4, 63]. In this work, we define independence-based fairness following [61] as follows.

**Definition 2.1 (Independence-based Fairness).** Let \( N \) be a neural network and \( \xi \) be a positive real-value constant. We write \( Y \) as the prediction of \( N \) on a set of input features \( X \) and \( L \) as the prediction set. We further write \( F \subseteq X \) as a feature encoding some protected characteristics such as gender, age and race. \( N \) satisfies independence-based fairness, with respect to \( \xi \), if and only if, \( \forall i \in L, \forall f_i, f_j \in F \) such that \( i \neq j \),

\[
| P(Y = l | F = f_i) - P(Y = l | F = f_j) | \leq \xi
\]

Intuitively, Definition 2.1 states that, \( N \) is fair as long as the probability difference of a favourable prediction for instances with different values of protected feature is within the threshold \( \xi \).

**Absence of backdoor:** With the wide adoption of neural networks in critical decision-making systems, sharing and adopting trained models become very popular. On the other hand, this gives attackers new opportunities. Backdoor attack is one of the neural network attacks that often cause significant threats to the system. Backdoor is a hidden pattern trained into a neural network, which produces unexpected behavior if and only if a specific trigger is added to an input [67]. In classification tasks, the backdoor misclassifies an arbitrary input to the same target label, when the associated trigger is applied to the input. Hence another desirable property of a neural network would be ‘backdoor free’, where the backdoor attack success rate is kept below a certain threshold.

**Definition 2.2 (Backdoor Attack Success Rate).** Let \( N \) be a backdoored neural network, \( t \) be the target label, \( X \) be a set of adversarial inputs with the trigger and \( N(x) \) be the prediction on \( x \). We say attack success rate (SR) is:

\[
SR(t) = P(N(x) = t | x \in X)
\]

Intuitively, attack success rate is the percentage of adversarial inputs classified into the target label. Next, we define backdoor-free neural network.

**Definition 2.3 (Backdoor-free Neural Network).** Let \( N \) be a backdoored neural network and \( \xi \) be a positive real-value constant. \( N \) satisfies backdoor-free property with respect to \( \xi \) and \( t \), if \( SR(t) < \xi \).

**Safety:** For neural networks applied on safety-critical systems such as the Airborne Collision Avoidance [33] system, all safety properties must be satisfied strictly. However, due to reasons like limited training data and insufficient training, those critical properties could be violated. Hence, another desirable behavior of a neural network is to satisfy all safety properties or the violation rate is kept below a certain threshold.

**Definition 2.4 (Safety Property Violation Rate).** Let \( N \) be a neural network, \( X \) be a set of inputs and \( N(x) \) be the prediction on \( x \). Let \( \rho \) be the critical safety property that \( N \) is expected to satisfy. We say the property violation rate (VR) is:

\[
VR(\rho) = P(N(x) \neq \rho | x \in X)
\]

Intuitively, violation success rate is the percentage of inputs that violate the property \( \rho \). Next, we define safety property violation-free neural network.

**Definition 2.5 (Safety Property Violation-free Neural Network).** Let \( N \) be a neural network and \( \xi \) be a positive real-value constant.
constant. \( N \) is safety property violation-free with respect to \( \xi \) and \( \rho \), if \( VR(\rho) < \xi \).

**Example 2.1.** We train a feed-forward neural network \( N \) on Census Income dataset (refer to details of the dataset in Section 4.1) to predict whether an individual’s income exceeds $50K per year. The neural network is configured to have 5 hidden layers. We use this network as the running example in this paper. In this example, we focus on a fairness property \( \phi \) w.r.t. protected feature gender, i.e., the model is considered unfair if the probability difference of a favourable prediction for females and males is greater than \( 1\% \). Note that this is for illustration purpose only and such a threshold is probably too strict for real-world applications. Given this neural network, the problem is to find a repaired network \( M \) with minimal adjustment to \( N \)’s weight parameters so that \( M \) satisfies the fairness property whilst maintaining its accuracy.

### 2.2 Our Problem

We are now ready to define our problem.

**Definition 2.6 (Neural Network Repair Problem).** Let \( N \) be a neural network, \( S \) be the input space, and \( \phi \) be one of the above-mentioned desirable properties. The neural network repair problem is to construct a neural network \( M \) such that \( M \) satisfies \( \phi \) in the input space \( S \) and the semantic distance between \( N \) and \( M \) is minimized.

We define semantic distance between two neural networks \( N \) and \( M \) as follows.

**Definition 2.7 (Semantic Distance).** Let \( N(x) \) be the prediction of neural network \( N \) on input \( x \) where \( x \in S \), the semantic distance between \( N \) and \( M \) is defined as:

\[
P(N(x) \neq M(x))
\]  

(4)

Intuitively, the semantic distance between two neural networks is the probability that the predictions of the two models are different on the same input. In this work, we use the model accuracy difference of \( N \) and \( M \) as a measure of their semantic distance.

### 3 OUR APPROACH

In this section, we present the details of our approach. An overview of our framework is as shown in Figure 1 and Algorithm 1 shows the details of our approach. The first step is to verify whether the given neural network \( N \) satisfy property \( \phi \). If \( \phi \) is satisfied, CARE terminates immediately. Otherwise, CARE proceeds to the next step. It performs causality-based fault localization on all hidden neurons of \( N \) and identifies those neurons that have the most contribution to \( N \)’s unwanted behavior. The third step is to optimize these neurons’ weights and generate a repaired network \( M \) such that \( M \) satisfies the property \( \phi \) and is semantically close to \( N \).

### 3.1 Property Verification

In this step, we verify neural network \( N \) against given property \( \phi \). If \( N \) satisfies \( \phi \), CARE returns \( N \) unchanged and terminates. Otherwise, CARE proceeds to repair the neural network. Property verification is not the focus of this work and we adopt existing neural network verification techniques. Recently there have been multiple tools and approaches for solving the above-mentioned neural network verification problems. We omit the details on how different properties are verified and refer the readers to [5, 34, 35, 61]. In our implementation, CARE’s fairness verification algorithm is based on [61], backdoor success rate verification is based on the method proposed in [67] and safety property verification is based on the approach proposed in [59].

**Example 3.1.** In our running example, CARE verifies \( N \) against the fairness property \( \phi \). The resultant fairness score (i.e., the probability difference of a favourable prediction for females and males) is 1.30\%. Therefore, \( N \) fails to satisfy the fairness property (which requires the fairness score to be within 1\%).

![Figure 1: An overview of our framework](image)
3.2 Causality Analysis

In this step we perform causality-based fault localization. The goal is to identify a relatively small set of neurons to repair. Note that in this work, we restrict ourselves to repair by adjusting the parameters of existing neurons only, i.e., without introducing or pruning neurons. The number of parameters in a neural network is often huge and even relatively small neural network may consist of thousands of weight and bias parameters. Therefore, attempting to optimize all neural weights would be costly and hard to scale. CARE thus adopts a fault localization method to identify neurons that are likely to be responsible for the undesirable behaviors. This method is based on causality analysis carried out over all hidden neurons of \( N \).

An alternative method for fault localization is gradient-based method, which is to measure how much perturbing a particular neuron would affect the output [57]. CARE is designed to capture the causal influence of a hidden neuron on the performance on the given property. In contrast, gradient-based method draws conclusion based on statistical correlations, which is not ideal for model repair (since our aim is to identify neurons that cause the defect). Next, we describe our causality-based fault localization in detail.

In recent years, causality has gained increasing attention in interpreting machine learning models [8, 27]. Multiple approaches have been designed to explain the importance of the components in a machine learning model when making a decision, based on causal attributions. Compared with traditional methods, causal approaches identify causes and effects of a model’s components and thus facilitates reasoning over its decisions.

In the following, we review some concepts which are necessary for the causality analysis in this work.

**Definition 3.1 (Structural Causal Models (SCM) [48]).** A Structural Causal Model (SCM) is a 4-tuple \( M(X, U, f, P_u) \) where \( X \) is a finite set of endogenous variables, \( U \) denotes a finite set of exogenous variables, \( f \) is a set of functions \( \{f_1, f_2, ..., f_n\} \) where each function represents a causal mechanism such that \( x_i \in X, f_i(Pa(x_i), u_i) \) where \( Pa(x_i) \) is a subset of \( X \setminus \{x_i\}, u_i \in U \) and \( P_u \) is a probability distribution over \( U \).

SCM serves as a framework for representing what we know about the world, articulating what we want to know and connecting the two together in a solid semantics [49]. It plays an important role in causality analysis and is commonly applied in many studies [8, 45, 46, 73].

Figure 2 shows an example causal graph to study the efficiency of a medication on a disease, where the nodes represent variables and the edges represent cause-effect relations intuitively. In this graph, age is considered as an exogenous variable (i.e., confounder), patient’s heart rate, cholesterol level and whether the medication is applied or not are endogenous variables (i.e., whether the medication is applied or not is often considered as the treatment). The outcome is the recovery rate of a patient. As illustrated in the graph, age affects the patient’s health conditions such as heart rate and level of cholesterol. Furthermore, the need or feasibility of applying this medicine on patients is affected by age, i.e., young people may not necessarily take the medicine and patients above 70 years old are too risky to take the medicine. Patient’s health condition and the application of medication affect the recovery rate. Furthermore, age can affect the recovery rate directly since younger patient often recover faster than the elderly.

In this work, we model neural networks as SCMs to analyze the causal relationship between hidden neurons and model’s predictions. To debug and repair a neural network, we would like to measure the ‘contribution’ of each neuron to the network misbehavior, which is referred as the causal effect.

**Definition 3.2 (Average Causal Effect).** The Average Causal Effect (ACE) of a binary random variable \( x \) (e.g., treatment), on another random variable \( y \) (e.g., outcome) is defined as follows.

\[
ACE = E[y|do(x=1)] - E[y|do(x=0)]
\]

where \( do(\cdot) \) operator denotes the corresponding interventional distribution defined by SCM.

Intuitively, ACE is used to measure the causal effect of \( x \) on \( y \) by performing intervention on \( x \). There are many other causal effect metrics in the literature such as

- average treatment effect (ATE), i.e., \( ATE = E[y(1) - y(0)] \), where \( y(1) \) represents the potential outcome if a treatment is applied exogenously and \( y(0) \) represents the corresponding potential outcome without treatment. ATE measures the effect of the treatment at the whole population level [29],
- conditional average treatment effect, i.e., \( CATE = E[y(1) - y(0)|x_0] \) [1], which is the average treatment effect conditioned on a particular subgroup of \( X \), i.e., \( X = x_0 \),
- and effect of treatment on the treated, i.e., \( ETT = E[y|x_1, x_0] - E[y|x_0] \) which measures the probability of \( Y \) would be \( y \) had \( X \) been \( x_1 \) counterfactually, given that in the actual world \( X = x_0 \) [71].

In this work, we focus on the ACE metric since we are interested in measuring the causal effect of the hidden neurons on certain output value.

3.3 Fault Localization

In the following, we present details on how causality analysis is used for fault localization in neural networks. Firstly, the neural network \( N \) is modeled as an SCM. As proposed in [38], neural networks can be interpreted as SCMs systematically. In particular, feed-forward neural networks (FFNNs) and convolutional neural networks (CNNs) can be represented as directed acyclic graphs with
 edges from an earlier (i.e., closer to the input layer) layer to the next layer until the output layer. The following is a proposition.

**Proposition 3.1.** An \( n \)-layer FFNN or CNN \( (x_1, x_2, ..., x_n) \) where \( x_i \) represents the set of neurons at layer \( i \), can be interpreted by SCM \( M([x_1, x_2, ..., x_n], U, [f_1, f_2, ..., f_n], P_U) \), where \( x_1 \) represents neurons at input layer and \( x_n \) represents neurons at output layer. Corresponding to every \( x_i, f_i \) represents the set of causal functions for neurons at layer \( i \). \( U \) represents a set of exogenous random variables that act as causal factors for input neurons \( x_1 \) and \( P_U \) is a probability distribution over \( U \).

**Proof.** In the scenario of FFNN, the proof of Proposition 3.1 follows that provided in [8]. In the scenario of CNN, similar to FFNN, neurons at each layer can be written as functions of neurons at its previous layer, i.e., \( \forall i : \forall x_{ij} \in x_i : x_{ij} = f_i(x_{i-1}) \), where \( x_{ij} \) represents the \( j \)th neuron at layer \( i \). Neurons at input layer \( x_1 \) can be assumed to be functions of independent noise variables \( U \) such that \( \forall x_{1j}, \in x_1 \) and \( u_j \in U : x_{1j} = f_1(u_j) \). Thus a CNN can be equivalently expressed by a SCM \( M([x_1, x_2, ..., x_n], U, [f_1, f_2, ..., f_n], P_U) \).

Figure 3 depicts the SCM of a 3-layer FFNN. The dotted circles represent exogenous random variables which act as causal factors for the input neurons. In this work, we assume that neurons at the input layer are not causally related to each other but can be jointly caused by a latent confounder.

Next we define the attribution problem, i.e., what is the causal influence of a particular hidden neuron on model defect. Recall that in Definition 3.2, we define the ACE of a binary variable \( x \) on output variable \( y \). However, we cannot apply the definition directly for two reasons. First, the domain of neural networks’ neurons is mostly continuous, not binary-valued. Second, we are interested in finding causal effects on the model defect rather than certain output variable. Hence we propose the following definition.

**Definition 3.3 (Causal Attribution).** We denote the measure of the undesirable behavior of given neural network \( N \) as \( y \). The Causal Attribution of a hidden neuron \( x \) to \( N \)’s defect \( y \) is:

\[
ACE_{do(x=\beta)}^y = \mathbb{E}[y|do(x=\beta)]
\]  

(6)

Next, we calculate the interventional expectation of \( y \) given intervention \( do(x=\beta) \) and we thus have the following.

\[
\mathbb{E}[y|do(x=\beta)] = \int_y yp(y|do(x=\beta))dy
\]  

(7)

Intuitively, causal attribution measures the effect of neuron \( x \) being value \( \beta \) on \( y \). We evaluate Equation 7 by sampling inputs according to their distribution whilst keeping the hidden neuron \( x = \beta \), and computing the average model undesirable behavior \( y \).

In this work, we calculate \( y \) according to the desirable property \( \phi \). Let \( N_t(x_{ip}) \) be the prediction value of class \( t \) on input \( x_{ip} \) we have:

- For fairness repair, we measure model unfairness \( y \) by taking the difference of the prediction value on favourable class \( t \) w.r.t. samples that only differ by the sensitive feature. Let \( x_{ip} \) and \( x_{ip}' \) be a pair of discriminatory instances [72] (that only differ by the sensitive feature). We have \( y_{fair} = | N_t(x_{ip}) - N_t(x_{ip}') | \) and we calculate \( ACE_{do(x=\beta)}^{y_{fair}} \) as the causal attribution of neuron \( x \) w.r.t. fairness property.
- For backdoor removal, we measure \( y \) by calculating the prediction value on target class \( t \), i.e., \( y_{back} = N_t(x_{ip}) \). Thus we aim to calculate \( ACE_{do(x=\beta)}^{y_{back}} \) for all hidden neurons. Note \( x_{ip} \) can be clean inputs or adversarial inputs.
- For safety property violation repair, we measure \( y \) by calculating the prediction value on desirable property \( t \), i.e., \( y_{safe} = N_t(x_{ip}) \). We calculate \( ACE_{do(x=\beta)}^{y_{safe}} \) as the causal attribution for hidden neuron \( x \).

For other properties, CARE is able to calculate the causal attribution of each neuron on \( y \) as long as the corresponding \( y \) is specified.

Thus by calculating the causal attribution of each hidden neuron on \( y \), we are able to identify neurons that are most likely the cause to the unexpected behavior. Afterwards, the identified neurons are used as candidates for model repair in our next step. The time spent on this step depends on the total number of neurons \( n \) to analyze in the given neural network. The time complexity for causality-based fault localization is thus \( O(n) \).

**Example 3.2.** In our running example, CARE conducts causality-based fault localization on all hidden neurons of \( N \) to identify neurons that are the most “responsive” to \( N \)’s prediction bias. CARE generates 12.4k samples to analyze all neurons and the total time taken is 120s. It is observed that a few neurons (such as the 1st neuron at 3rd hidden layer and the 3rd neuron at 5th hidden layer) have a contribution which is “outstanding”. This is good news to us as it implies that it is possible to improve the fairness by modifying the weights of only a few neurons. CARE then selects the top 10\% of total number of neurons (i.e., 13 in total for this network) to be used as repair candidates.

### 3.4 Network Repair

Next, we present how CARE performs neural network repair. Similar to traditional software programs, neural networks can go wrong and incorrect behaviors should be fixed. While existing efforts mainly focus on how to retrain the neural network to repair the unexpected behaviors, there are a few recent works [3, 23, 65] that address this problem by modifying the parameters of the network.
We write $\mathcal{R}$ since the large number of elements undergoing mutation works as demonstrated in [10]. Their heuristics (i.e., focusing on their approach has limited effectiveness in repairing neural networks as demonstrated in [10]. Their heuristics (i.e., focusing on neurons in the output layer) is not justified from our point of view as the neurons causing the problematic behaviors may not be in the output layer. In [3], an approach is proposed to repair a model through inductive synthesis. Their approach has scalability issues as a large model size leads to a large search space. While NNRepair [65] provides a constraint-based approach to fix the logic of a neural network at an intermediate layer or output layer, its fault localization and repair applies to a single layer only but the actual fault could be across multiple layers. Thus its repair performance may not be ideal and we provide a comparison in Section 4.2.

In our approach, we propose to address the repair problem through optimization, i.e., based on the Particle Swarm Optimization (PSO) algorithm. Besides PSO, there are many other optimization algorithms. Genetic optimization algorithms (genetic algorithm for example) usually do not handle complexity in an efficient way [34] since the large number of elements undergoing mutation causes a considerable increase in the search space. In comparison, PSO requires smaller number of parameters and thus has lower number of iterations. Another type of optimization is stochastic optimization algorithm. Markov Chain Monte Carlo (MCMC) as an representative uses a sampling technique for global optimization. However, MCMC often encounter either slow convergence or biased sampling [18]. Hence, in this work we select intelligence based algorithm PSO.

The idea is to search for small adjustments in weight parameters of the neurons identified in the previous step such that the specified property is satisfied. PSO simulates intelligent collective behavior of animals such as schools of fish and flocks of birds. It is known to be particularly effective for optimization in continuous domains [52]. In PSO, multiple particles are placed in the search space. At each time step, each particle updates its location $\vec{x}_i$ and velocity $\vec{v}_i$ according to an objective function. That is, the velocity is updated based on the current velocity $\vec{v}_i$, the previous best location found locally $\vec{p}_i$, and the previous best location found globally $\vec{g}$. Its location is updated based on the current location and velocity. We write $R(0, c)$ to denote a random number uniformly sampled from the range of $[0, c]$. Formally, the PSO update equation is as follows [55].

$$\vec{v}_i^t \leftarrow \omega \vec{v}_i^{t-1} + R(0, c_1)(\vec{p}_i - \vec{x}_i^t) + R(0, c_2)(\vec{g} - \vec{x}_i^t)$$  \hspace{1cm} (8)

$$\vec{x}_i^t \leftarrow \vec{x}_i^{t-1} + \vec{v}_i^t$$  \hspace{1cm} (9)

where $\omega$, $c_1$ and $c_2$ represent inertia weight, cognitive parameter and social parameter respectively. In PSO, the fitness function is used to determine the best location.

In CARE, the weights of the identified neurons are the subject for optimization and thus are represented by the location of the particles in PSO. The initial location of each particle is set to their original weight and the initial velocity is set to zero.

As defined in Section 2, our problem is to repair a given neural network $N$ against property $\phi$ while minimizing the accuracy drop. Therefore, two essential components need to be considered in the optimization process: 1) model performance on property $\phi$ and 2) model accuracy. In CARE, we measure the model performance based on the property $\phi$ specified. Formally, the fitness function of PSO is defined as follows.

$$\text{fitness} = (1 - \alpha) UB + \alpha (1 - \text{accuracy})$$  \hspace{1cm} (10)

where UB (undesirable behavior) is a quantitative measure on the degree of violating property $\phi$; constant parameter $\alpha \in (0, 1)$ determines the relative importance of the accuracy. For fairness improvement task, we randomly sample a large set of instances and UB is measured by the percentage of individual discriminatory instances within the set, i.e., let $N(x)$ be the prediction of $x$ and $(x, x')$ be a pair of discriminatory instances in the sample set, $UB = P(N(x) \neq N(x'))$; for backdoor removal task, we add backdoor trigger (if it is known) to all samples in testing set and measure $UB = \mathcal{SR}(t)$ following Definition 2.2; for safety task, we randomly sample a large set of instances and measure $UB = \mathcal{VR}(\rho)$ following Definition 2.4. Intuitively, our optimization target is to make the repaired model satisfy given property $\phi$ while maintaining the accuracy. Note that, for other properties, CARE is able to conduct repair as long as a performance indicator is specified based on the given property properly.

**Example 3.3.** In our running example, CARE applies PSO to generate a repair network $M$ that will improve the fairness performance of $N$. The weights of the identified 13 neurons in the previous step are the subjects for the optimization. We set $\alpha$ to 0.8 so that PSO can optimize for fairness without sacrificing model accuracy too much. PSO terminates at 17th iteration where no better location is found in the last 10 consecutive iterations. By adjusting neural weights accordingly, $M$ is generated. CARE performs fairness and accuracy check on $M$ and the maximum probability difference is 0.007 with accuracy of 0.86 (original accuracy is 0.88). Thus, CARE successfully repairs $N$ with a fairness improvement of 46.1% and the model accuracy is mildly affected.

**4 IMPLEMENTATION AND EVALUATION**

In the following, we conduct a set of experiments to evaluate CARE. We demonstrate the technique in the context of three different tasks: 1) neural network fairness improvement, 2) neural network backdoor removal and 3) neural network safety property violation

![Figure 4: Causal Attribution of Neurons in the Example](image)
repair. All experiments are conducted on a machine with 1 Dual-Core Intel Core i5 2.9GHz CPU and 8GB system memory. To reduce the effect of randomness, all experimental results are the average of five runs if randomness is involved.

4.1 Experiment Setup
In the following, CARE is evaluated over multiple datasets by answering multiple research questions (RQs). The details of the datasets are summarized as follows:

For fairness repair, CARE is evaluated over three datasets that are commonly used in machine learning model fairness testing [2, 21, 64, 72],
- **Census Income [12]:** This dataset consists of 32,561 training instances containing 13 features and is used to predict whether an individual income exceeds $50K per year. Among all attributes, gender, age and race are three protected characteristics. The labels are if an individual income exceeds $50K per year or not.
- **German Credit [13]:** This dataset consists of 1000 instances with 20 features and is used to assess an individual’s credit. Here, age and gender are the protected features. The labels are whether a person’s credit is good or not.
- **Bank Marketing [11]:** This dataset consists of 45,211 instances and there are 17 features. Among them, age is the protected feature. The labels are whether the client will subscribe a term deposit or not.

We train three feed-forward neural networks following standard practice and run CARE to repair the unfair behavior of each model against the corresponding protected features.

For backdoor removal, CARE is evaluated over three datasets:
- **German Traffic Sign Benchmark Dataset (GTSRB) [60]:** This dataset consists of 39.2K training instances and 12.6K testing instances of colored images. The task is to recognize 43 different traffic signs. We train a CNN network consists of 6 convolutional layers and 2 dense layers.
- **MNIST [41]:** This dataset consists of 70K instances of handwritten digits as gray-scale images. We train a standard 4-layer CNN network to recognize the 10 digits (0-9).
- **Fashion-MNIST [69]:** This data set consists of 70K instances of 10 fashion categories, e.g., dress, coat and etc. Each sample is a 28x28 grayscale image. We train a CNN network consists of 3 convolutional layers and 2 dense layers.

For safety violation repairing, CARE is evaluated over three ACAS Xu [33] networks. ACAS Xu contains an array of 45 DNNs that produces horizontal maneuver advisories of unmanned version of Airborne Collision Avoidance System X. As discovered in [34, 50], some DNNs violates certain safety properties, e.g., DNN N2,9 violates the safety property $\phi_s$. We apply CARE on 3 sub networks N2,9, N3,3 and N1,9 against 3 properties $\phi_s$, $\phi_2$ and $\phi_7$ respectively, aiming to repair the misbehavior. Table 1 shows the details of the trained networks used in our experiment.

In PSO, we follow the general recommendation in [51] and set parameter $\omega = 0.8$, $c_1 = c_2 = 0.41$ and number of particles to 20. The maximum number of iteration is set to 100. To further reduce the searching time, we stop the search as soon as the property is satisfied or we fail to find a better location in the last 10 consecutive iterations. Note that CARE performs accuracy check after PSO finds an $M$. If the accuracy drop is significant, i.e., bigger than a threshold of 3%, CARE returns error message and a larger value of $\alpha$ shall be set in such scenarios.

4.2 Research Questions and Answers
In the following, we report our experiment results and answer multiple research questions.

**RQ1:** Is CARE successful in neural network repair? To answer this question, we systematically apply CARE to the above-mentioned neural networks.

For **fairness repair tasks**, to better evaluate the performance of CARE, we set a strict fairness requirement (i.e., $\xi = 1\%$) to make sure all models fail the fairness property and CARE performs fairness repair on all of them (note that some of models fail fairness even if $\xi$ is set to be a realistic value of 5%). Table 2 summarizes the results, where the columns show model, the protected feature, the unfairness (maximum probability difference) before and after repair and model accuracy before and after repair. Note that in this experiment, input features are assumed to be independent and normal distribution is followed in our sampling process. The number of neurons to repair is set to be no more than 10% of total number of neurons in the given network. As illustrated in Table 2, among all cases, $N2$ and $N3$ show alarming fairness concern (i.e., with unfairness above 5%). CARE successfully repairs all neural networks, with an average fairness improvement of 61.9% and maximum of 99.2%. In terms of model accuracy, either the accuracy is unchanged ($N2$) or has only a slight drop ($N1$ and $N3$).

We further compare the performance of CARE with the state-of-the-art work [61] for this task. The method proposed in [61], relies on learning a Markov Chain from original network and performing sensitivity analysis on it. Then optimization is applied to find small adjustments to weight parameters of sensitive neurons for better fairness. We run experiments on $N1$, $N2$ and $N3$ and the performance comparison is shown in Table 5 (DTMC represents the method proposed in [61]). CARE is able to improve fairness by 61.9% on average and the model accuracy drops by 1.7%, while DTMC only improves fairness by 45.1% at a higher cost of 2.2%.

For **backdoor removal tasks**, we train neural networks $N4$, $N5$ and $N6$ following the attack methodology proposed in BadNets [26]. We randomly chose a target label for each network and

| Table 1: Neutral Networks Used in Our Experiments |
|--------------------------------------------------|
| Model Dataset | Architecture | #Neuron | Accuracy |
|---------------|--------------|---------|----------|
| $N1$ Census    | 6-layer FFNN  | 139     | 0.8818   |
| $N2$ Credit    | 6-layer FFNN  | 146     | 1.0      |
| $N3$ Bank      | 6-layer FFNN  | 143     | 0.9226   |
| $N4$ GTSRB     | 3-Conv + 2 Dense CNN | 107,595 | 0.9657   |
| $N5$ MNIST     | 2-Conv + 2 Dense CNN | 31,610  | 0.9900   |
| $N6$ Fashion   | 3-Conv + 2 Dense CNN | 67,226  | 0.9136   |
| $N7$ ACAS N2,9 | 6-layer FFNN  | 305     | -        |
| $N8$ ACAS N3,3 | 6-layer FFNN  | 305     | -        |
| $N9$ ACAS N1,9 | 6-layer FFNN  | 305     | -        |
ACAS Xu networks as our... of the above mentioned three ACAS Xu networks. The average result
of property repair tasks, we use \( N_{29}, N_{33} \) and \( N_{19} \) of ACAS Xu networks as our \( N_{7}, N_{8} \) and \( N_{9} \). These networks take 5-dimensional inputs of sensor readings and output 5 possible maneuver advises. Katz et al. [34] and Long et al. [50], demonstrate that \( N_{29} \) violates the safety property \( \phi_{6} \), \( N_{33} \) violates property \( \phi_{2} \) and \( N_{19} \) violates property \( \phi_{7} \). Therefore, we apply CARE on these 3 networks to improve their performance on the corresponding property. In this experiment, for each network, we randomly sample 10k counterexamples to the property as the evaluation set and 10k instances that are correctly classified by the original neural network as the drawdown set. We measure the violation rate (VR) on each set to evaluate the performance. As shown in Table 4, for all the three networks, CARE successfully brings down the violation rate from 1.0 to < 1% in evaluation set, while the performance in drawdown set is not affected.

To further evaluate the performance of CARE on safety property repair tasks, we compare CARE with the state-of-the-art approach proposed in [59] named PRDNN. PRDNN [59] introduces decoupled neural network architecture and solves neural network repair problem as a linear programming problem. We apply PRDNN on the above mentioned three ACAS Xu networks. The average result of \( N_{7}, N_{8} \) and \( N_{9} \) is shown in Table 5. Both tools are able improve the performance effectively while CARE outer-performs PRDNN by 2.2% although the cost is slightly higher (still below 1%)\(^4\).

Thus to answer RQ1, we say CARE is able to repair given buggy neural network successfully for various properties while maintaining high model accuracy. Furthermore, CARE outer-performs existing works for different tasks.

RQ2: What is the effect of parameter \( \alpha \), i.e., the trade-off between the degree of property violation and accuracy? Recall that the value of \( \alpha \) controls the trade-off between fairness and accuracy. To understand the effect of \( \alpha \)’s value, we perform another set of experiments.

Figure 5 illustrates the result on all neural networks, where the first plot shows the performance improvement of repaired network compared to original network and the second plot shows the cost measured by model accuracy drop. As \( \alpha \) increases from 0.1 to 0.9, the importance of model accuracy over repair objective in PSO fitness function increases. As shown in the plots for \( N_{7} \), the model accuracy is quite sensitive with respect to the value of \( \alpha \), i.e., when \( \alpha = 0.2 \) model accuracy drops by 0.05 (from 0.88 to 0.83). Although a smaller \( \alpha \) results in better fairness, we select a larger \( \alpha \) to keep high model accuracy. As shown in the plots for \( N_{8} \), model accuracy is stable over different \( \alpha \) values. Similar to the previous case, smaller \( \alpha \) results in better fairness. Therefore, we select a small \( \alpha \) for more effective fairness repair. As for \( N_{9} \), a large \( \alpha \) improves model accuracy significantly. When \( \alpha \) is greater than 0.7, the model accuracy is even higher than the original network. That is because PSO tries to optimize for model accuracy as well. However, the effectiveness of fairness improvement drops, i.e., model unfairness is reduced to 0.001 for \( \alpha = 0.1 \) but at \( \alpha = 0.9 \) the unfairness is as high as 0.0368. For \( N_{7}, N_{8} \) and \( N_{9} \) costs for all \( \alpha \) values are small (< 1%). This is because fixing the backdoor itself will improve model accuracy as well. For \( N_{7} \) and \( N_{9} \) the performance improvement (drop in SR) is quite stable over different \( \alpha \). For \( N_{8} \), the performance improvement drops as \( \alpha \) increases, hence, we select small \( \alpha \) to let the optimization focus on backdoor removal. As for \( N_{7}, N_{8} \) and \( N_{9} \), the performance improvement is stable over different \( \alpha \)’s values. For \( N_{7} \), the cost drops from 0.25% to 0.02% when \( \alpha \) increases from 0.1 to > 0.2. While for \( N_{8} \) and \( N_{9} \), value of \( \alpha \) does not affect the cost much. Hence, we select small \( \alpha \) value for \( N_{7} \) and \( N_{9} \) and select \( \alpha > 0.1 \) for \( N_{8} \). As shown by the experiment results, the value of \( \alpha \) balances the trade-off between performance improvement and cost. A smaller \( \alpha \) often results in more effective property repair but model accuracy may be affected. In our experiments, we set the value of \( \alpha \) as shown in Table 6.

Thus to answer RQ2, we say that \( \alpha \) does have a significant impact on the repairing result. In practice, the users can decide its value based on the importance of the property. In the future, we aim to study systematic ways of identifying the best \( \alpha \) value automatically.

RQ3: How effective is the causality-based fault localization? This question asks whether our causality-based fault localization is useful. We answer this question by comparing the repair effectiveness through optimizing the weight parameters that are selected in four different ways, i.e., 1) selected based on our approach; 2) selected...\(^1\)A bug was found in the initial implementation for the sampling range of \( N_{9} \) and all the related results are updated accordingly after the fix.

\(^2\)PRDNN hangs with our initial experimental setup but we resolved this issue by setting up another Ubuntu virtual machine with more disk size.

\(^3\)We use the same validation set generated by CARE to evaluate both tools.
Table 2: Fairness Repair Results

| Model | P. Feat. | Fairness Score Before | After | Accuracy Before | After | Time(s) Loc | Tot |
|-------|---------|------------------------|-------|-----------------|-------|------------|-----|
| NN1   | Race    | 0.0431                 | 0.0119| 0.88            | 0.85  | 118        | 314 |
| NN1   | Age     | 0.0331                 | 0.0230| 0.88            | 0.86  | 173        | 276 |
| NN1   | Gender  | 0.0130                 | 0.0070| 0.88            | 0.86  | 120        | 240 |
| NN2   | Race    | 0.0659                 | 0.0005| 1.0             | 1.0   | 170        | 221 |
| NN2   | Gender  | 0.0524                 | 0.0374| 1.0             | 1.0   | 103        | 129 |
| NN3   | Age     | 0.0544                 | 0.0028| 0.92            | 0.91  | 174        | 435 |

Table 3: Backdoor Removal Results

| Model | Attack SR Before | After | Accuracy Before | After | Time(s) Loc | Tot |
|-------|-----------------|-------|-----------------|-------|------------|-----|
| NN4   | 0.9808          | 0     | 0.9657          | 0.9654| 80         | 493 |
| NN5   | 0.9868          | 0     | 0.9909          | 0.9867| 330        | 541 |
| NN6   | 0.9977          | 0.0007| 0.9136          | 0.9199| 108        | 492 |

Table 4: Safety Property Repair Results

| Model | Counterexample VR Before | After | Drawdown VR Before | After | Time(s) Loc | Tot |
|-------|--------------------------|-------|--------------------|-------|------------|-----|
| NN7   | 1.0                      | 0.0065| 0.0                | 0     | 33         | 503 |
| NN8   | 1.0                      | 0     | 0.0                | 0     | 31         | 168 |
| NN9   | 1.0                      | 0.0001| 0.0                | 0     | 39         | 325 |

Table 5: Comparison with Existing Works

| Tech. | Fairness Imp. Cost | Backdoor Imp. Cost | Safety Imp. Cost |
|-------|--------------------|--------------------|------------------|
| CARE  | 61.9%              | 1.7%               | 99.9%            |
| DTMC  | 45.1%              | 2.2%               | -                |
| NNRREPAIR | -                  | 19.3%              | 5.9%             |
| PRDNN | -                  | -                  | -                |

Table 6: Parameter $\alpha$ Setting

| Model | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| $\alpha$ | 0.1   | 0.7   | 0.8   | 0.1   | 0.1   | 0.3   |       |       |       |

| Model | $N_1$ | $N_2$ | $N_3$ | $N_4$ | $N_5$ | $N_6$ | $N_7$ | $N_8$ | $N_9$ |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| $\alpha$ | 0.2   | 0.2   | 0.2   | 0.3   | 0.1   | 0.2   |       |       |       |

Figure 5: Effect of Parameter $\alpha$

While CARE improves the performance by 80% and in average the improvement is only 36.6%, although the model accuracy is not affected much. Therefore, with randomly selected parameters, the repair is not effective and the performance is improved at the cost of disrupting correct behavior of the original model. On the other side, as described in Section 3, our fault localization is based on the causal influence of each neuron on the targeted misbehavior without altering other behaviors.

Secondly, we optimize all weight parameters of the given neural network, i.e., no fault localization. In this setting, the search space in the PSO algorithm increases significantly (compared to the case of repairing 10% of parameters in CARE). Therefore, we limit the time taken by PSO so that time allowed to spend in this step is the same as that taken by CARE with fault localization. The results are shown in columns All. For fairness repair tasks the fairness is improved by 67.8% on average. But the model accuracy drops significantly, i.e., the accuracy drops by 20.1% on average and 42.0% in the worst case. For backdoor removal, the performance improvement is only 36.2% on average, while CARE reduces the SR by 99.5%. For safety repair tasks, the performance improvement is 55.6% on

randomly; 3) selected based gradients and 4) include all the parameters. The results are shown in Table 7.

Firstly, we perform network repair with randomly selected neurons to optimize. We follow the same configuration of CARE as the one we used in RQ1, i.e., with the same $\alpha$ value and the number of neurons to fix. The performance improvement and cost of the repair is shown in $Random1$ columns in Table 7. Adjusting randomly selected parameters results in an average performance improvement of 47.6%. While CARE improves the performance by 80.8% on all tasks. For fairness repair tasks, the improvement is significant with randomly selected neurons. However, the model accuracy drops sharply. The overall accuracy drop is 8.9% on average with the worst case of 19.0% for $NN_2$ w.r.t. protected feature gender. For backdoor removal tasks, the performance improvement is 6.4% on average and model accuracy is affected. Especially for $NN_6$, accuracy is reduced by 6.3% after the repair. In terms of safety repair
average and CARE manages to reduce the VR in evaluation set by 99.8%. Although the cost is below 1% for these two tasks, the repair is not that effective. In all of the cases without fault localization, PSO is not able to converge within the time limit and as a result, neither performance improvement nor cost is optimized when PSO terminates.

In the literature, gradient is a commonly used technique that guides the search in program fault localization, program repair and fairness testing [20, 57, 72]. Therefore, we conduct experiments to compare the performance of CARE with gradient-guided fault localization. That is, instead of calculating causal attribution of the hidden neurons, the fault localization step is based on \( \frac{\partial v}{\partial \theta} \) where \( y \) represents the favourable output value and \( v \) represents hidden neuron value. We use the gradient-based method to identify the same amount of "responsible" neurons and perform optimization with the same setting as CARE. The results of gradient-based localization method is illustrated in columns Gradient of Table 7. Compared with CARE, for fairness repair tasks, the gradient-based method is able to find a repair that satisfies fairness criteria but fails to maintain model accuracy. Overall fairness improves by 73.6% but average accuracy drops by 17.8%, which is not desirable. For backdoor removal, the gradient-based method is not able to find an effective repair where the overall performance improvement is below 30%. Especially for \( NN_4 \), the SR is still as high as 96.4% after the repair. For safety repair, gradient-based method is able to find a repair for \( NN_8 \) and \( NN_9 \) with performance improvement above 90%, while CARE archives 100% backdoor removal. For \( NN_7 \), gradient-based method is not useful at all where VR in evaluation set is as high as 99% after the repair. Hence the performance of gradient-based method is not stable and often not effective.

Thus to answer RQ3, we say our causality-based fault localization is effective in identifying candidates for parameter optimization. It guides CARE to focus on fixing undesirable behavior of the model while keeping the correct behavior unaffected.

### 5 RELATED WORK

This work is broadly related to works on neural network verification, repair and causal interpretation.

**Neural network verification.** There have been an impressive line of methods proposed recently for neural network verification. These includes solving the problem using abstraction techniques [16, 22, 56], SMT solver [30, 34, 35], MILP and LP [15, 62], symbolic execution [68] and many others [7, 14, 25, 37]. There have also been attempts to verify neural network fairness properties, including [4] and [5] based on numerical integration, [31] based on non-convex optimization and [61] based on probabilistic model checking. Unlike these works, our approach focus on neural network repair instead and we leverage some existing verification methods [59, 61, 67] in our work for property verification.

**Machine learning model repair.** There have been multiple attempts on repairing machine learning models to remove undesirable behaviors. In [36], Kauschke et al. suggest a method to learn a model to determine error regions of the given model, which allows users to patch the given model. In [58], Sotoudeh et al. propose a method to correct a neural network by applying small changes to model weights based on SMT formulation of the problem. In [23], Goldberger et al. leverage recent advances in neural network verification and presented an approach to repair a network with minimal change in its weights. This approach aims to find minimal layer-wise fixes for a given point-wise specification and the performance is restricted by the underlying verification method used. NNR repair [65] performs constraint-based repair on neural networks to fix undesirable behaviors but only applies to a single layer. Sotoudeh et al. proposed a method in [59] to solve a neural network repair problem as a linear programming problem. Unlike our approach, both NNR repair and the method proposed in [59] perform layer-wise repair but the actual fault in a buggy network could be across multiple layers. Furthermore, based on our experiment results, our approach is more effective in neural network repair against different properties.
Machine learning causal interpretation. Causality analysis has been applied to generate explanations for machine learning algorithms. Existing works focus on causal interpretable models that can explain why machine learning models make certain decisions. Narendra et al. [46] model DNNs as SCMs and estimate the causal effect of each component of the model on the output. In [8], Chattopadhyay et al. proposed a scalable causal approach to estimate individual causal effect of each feature on the model output. Causal inference has been applied in machine learning model fairness studies. Kusner et al. [40] proposed an approach to measure the fairness of a machine learning model based on counterfactuals where a fair model should have the same prediction for both actual sample and counterfactual sample. In [71], Zhang et al. propose a causal explanation based metric to quantitatively measure the fairness of an algorithm. Our work utilizes SCMs and do(·) calculus to measure the causal attribution of hidden neurons on model undesirable behaviors. The results are used as a guideline for fault localization.

6 CONCLUSION
We present CARE, a causality-based technique for repairing neural networks for various properties. CARE performs fault localization on a given neural network model and utilizes PSO to adjust the weight parameters of the identified neurons. CARE generates repaired networks with improved performance over specified property while maintaining the model’s accuracy. CARE is evaluated empirically with multiple neural networks trained on benchmark datasets and experiment results show that CARE is able to repair buggy model efficiently and effectively, with minimally disruption on existing correct behavior.

7 DATA AVAILABILITY
A Prove of Concept (PoC) realization of this work (CARE) is implemented on top of SOCRATES [50] as a causality-based neural network repair engine. The source code of CARE is publicly available at [56].

8 ACKNOWLEDGEMENTS
We thank anonymous reviewers for their constructive feedback. This research is supported by Huawei International (Grant Number TC20210714014).
[70] Bin Yang, Honglei Guo, and Enguo Cao. Chapter two - design of cyber-physical-social systems with forensic-awareness based on deep learning. In Ali R. Hurson and Sheng Wu, editors, *AI and Cloud Computing*, volume 120 of *Advances in Computers*, pages 39–79. Elsevier, 2021.

[71] Junzhe Zhang and Elias Bareinboim. Fairness in decision-making - the causal explanation formula. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 2037–2045. AAAI Press, 2018.

[72] P. Zhang, J. Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, J. Dong, and Dai Ting. White-box fairness testing through adversarial sampling. pages 1–12, 2020.

[73] Qingyuan Zhao and Trevor Hastie. Causal interpretations of black-box models. *Journal of Business & Economic Statistics*, 39(1):272–281, 2021.