FIPSER: Improving Fairness Testing of DNN by Seed Prioritization

Junwei Chen
51265902128@stu.ecnu.edu.cn
East China Normal University
Shanghai, China

Yueling Zhang*
ylzhang@sei.ecnu.edu.cn
East China Normal University
Shanghai, China

Lingfeng Zhang
lanford217@gmail.com
East China Normal University
Shanghai, China

Min Zhang
mzhang@sei.ecnu.edu.cn
East China Normal University
Shanghai, China

Chengcheng Wan
ccwan@sei.ecnu.edu.cn
East China Normal University
Shanghai, China

Ting Su
tsu@sei.ecnu.edu.cn
East China Normal University
Shanghai, China

Geguang Pu
ggpu@sei.ecnu.edu.cn
East China Normal University
Shanghai, China

ABSTRACT

As a rapidly evolving AI technology, deep neural networks are becoming increasingly integrated into human society, yet raising concerns about fairness issues. Previous studies have proposed a metric called causal fairness to measure the fairness of machine learning models and proposed some search algorithms to mine individual discrimination instance pairs (IDIPs). Fairness issues can be alleviated by retraining models with corrected IDIPs. However, the number of samples that are used as seeds for these methods is often limited due to the pursuit of efficiency. In addition, the quantity of IDIPs generated on different seeds varies, so it makes sense to select appropriate samples as seeds, which has not been sufficiently considered in past studies. In this paper, we study the imbalance in IDIP quantities for various datasets and sensitive attributes, highlighting the need for selecting and ranking seed samples. Then, we proposed FIPSER, a feature importance and perturbation potential-based seed prioritization method. Our experimental results show that, on average, when applied to the current state-of-the-art method of IDIP mining, FIPSER can improve its effectiveness by 45% and efficiency by 11%.

CCS CONCEPTS

• **Computing methodologies → Artificial intelligence**; • **Software and its engineering**;

KEYWORDS

Fairness Testing, Imbalance, Seed Prioritization, Feature Importance

*Corresponding author.

ACM Reference Format:
Junwei Chen, Yueling Zhang, Lingfeng Zhang, Min Zhang, Chengcheng Wan, Ting Su, and Geguang Pu. 2024. FIPSER: Improving Fairness Testing of DNN by Seed Prioritization . In *39th IEEE/ACM International Conference on Automated Software Engineering (ASE '24), October 27-November 1, 2024, Sacramento, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3691620.3695486

1 INTRODUCTION

Deep learning has emerged as a powerful technique that is becoming more and more integrated into people's daily lives and professional activities [29, 36, 48, 50]. As this trend progresses, concerns about fairness defects in machine learning systems have arisen. The recruiting tool used by Amazon is an example exposing fairness issues in machine learning systems. It was discovered that Amazon's ML hiring system was discriminating against female candidates, particularly for software development and technical positions [13]. One possible reason for this is that most recorded historical data were for male software developers [39]. To address these issues, some existing research aims to propose appropriate fairness metrics, while others seek to improve the fairness of machine learning systems from various stages and perspectives.

Fairness metric research aims to develop methods for measuring the fairness of machine learning systems. Recent research [33] categorizes fairness metrics into three groups: group fairness, individual fairness, and subgroup fairness. Equalized odds and equal opportunity [19] are two commonly used group fairness metrics. Subgroup fairness [28] is a representation of the subgroup fairness metric. For individual fairness, causal fairness [10, 17] is a popular and intuitive metric.

Fairness improvement research [7, 19, 27, 40] endeavors to ensure that machine learning systems make decisions not only with high accuracy but also with good fairness. Previous studies have attempted various perspectives to propose their methods, such as focusing on mitigating the influence of biased paths in neural networks [18] or adaptively choosing the fairness-improving method based on causality analysis [56].

Recently, some research [8, 17] argued that group fairness property might not detect bias in scenarios when the same amount of discrimination is made for each group, which led to the usage of individual fairness in many recent works [4, 8, 47]. So, we study causal fairness in this research. It refers to scenarios where, for a given individual and a machine learning model, if another individual differs only in sensitive attributes and the model gives different predictions, the model is considered discriminatory towards the previous individual. **Figure 1 (a)** provides an example of a simple individual discrimination instance pair (IDIP) mining process. Notice that simply removing sensitive attributes from training data does not effectively improve model fairness, as models may implicitly learn sensitive attributes and exhibit discriminatory behavior in predictions[15, 19]. Many past studies focused on mining individual discrimination instances from neural networks. ADF [57] adopted a two-stage search method inspired by AEQUITAS [47] and introduced techniques from adversarial sample mining, utilizing gradient information for more effective discrimination instance searching. EIDIG [55] extended ADF by reducing gradient calculations and introducing momentum for improved efficiency and effectiveness. NF [58] further improved search effectiveness by selecting and utilizing biased neurons based on neuron-level analysis results. DICE [34] introduced concepts from information theory to enhance the search process, achieving current state-of-the-art results.

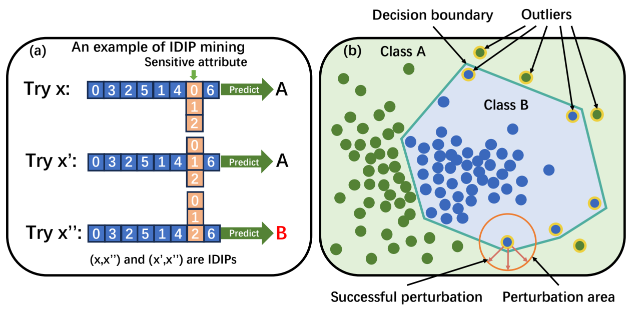

Figure 1: An explanation of IDIP mining and the intuition of outlier. (a) is an example of IDIP mining. x is a sample in the dataset. Its 7th attribute is a sensitive attribute, with three possible values: 0, 1, and 2. The initial value of the attribute is 0. We first compute the neural network's prediction for sample x. Then, we perturb its sensitive attribute values to 1 and 2, respectively, and observe the neural network's prediction for perturbed samples. In this example, the prediction for the perturbed sample x'' is different from x, so they form a pair of IDIP. (b) explains why outliers can enhance the effectiveness of IDIP mining. In the figure, the points with yellow borders represent some outliers, the teal line represents the decision boundary, and the orange circle represents a neighborhood around the outlier. Within the neighborhood, perturbations that cross the decision boundary, represented by pink arrows, successfully generate IDIPs.

However, these methods face challenges in conducting fairness testing on complete datasets due to the limitation of time, computing power, and other constraints. Current solutions typically involve sample clustering with a round-robin sampling strategy to choose a subset of seed samples and only conduct the mining process for the chosen samples. This simple approach does not fully leverage information in samples and neural networks, resulting in insufficient effectiveness. Moreover, our observations indicate that the quantity of IDIPs is imbalanced across datasets and sensitive attributes. Here, imbalance refers to the disparity in the number of IDIPs mined from different samples. For example, in terms of the age attribute of the diabetes dataset, 6039, 6081, and 5910 IDIPs are yielded on three randomly selected samples. This indicates a relatively balanced result. Conversely, on the age attribute of the heart dataset, three randomly selected samples yielded IDIPs with counts of 0, 1707, and 6468, respectively, which shows a significant imbalance. The existing sampling strategy of IDIP mining methods has not taken this imbalance into consideration, which could be a threat to their effectiveness and efficiency. To address these challenges, we aim to develop a better sampling strategy by further analyzing and leveraging information embedded in neural networks and samples. This strategy aims to select more suitable seed samples for IDIP mining, enhancing the overall efficiency and effectiveness of these IDIP mining methods.

In this paper, we propose a Feature Importance and Perturbation potential-based SEed prioRitization method (FIPSER). Our intuition is that outliers in datasets are likely to generate more IDIP. **Figure 1 (b)** gives a visual explanation for the intuition. The task of finding IDIP $(x, \tilde{x})$ for sample x is essentially trying to find at least one perturbed sample within a neighborhood of x, which has crossed the decision boundary. As outliers are closer to the model's decision boundary, they are more likely to generate IDIPs. Moreover, different dimensions of data have varying impacts on model decisions. Taking this into consideration, we propose a feature importance estimation method to fully utilize information in samples and neural networks to assess the impact of different features on decision-making. We first concatenate the tabular data with their output on each neuron to form the feature matrix and estimate their importance. These features will be processed by feature processing procedures, including feature selection and feature merging. In these procedures, features with high similarity or low importance will be merged or dropped, aiming to enhance the effectiveness of the remaining features. Furthermore, we propose a perturbation potential estimation method to estimate the possibility of seeds that generate IDIP through perturbation. Our observations indicate that seeds with high perturbation potential are more likely to succeed in the global search stage, and seeds with high outlier scores are more effective in the local search stage. Local search generates the majority of IDIP. Consequently, when determining the sorting result, we let the outlier score have a relatively large influence and the perturbation potential have a relatively small influence. We then calculate outlier scores based on processed features and divide samples into multiple sections according to their perturbation potential. Finally, we sort samples within each section based on their outlier scores and ensure that sections with high perturbation potential are prioritized at the front of the sorting order.

In summary, our contributions include:

(1) We observed and figured out the imbalance of IDIP quantity on different seeds for multiple datasets and sensitive

attributes, showing the necessity of seed prioritization for IDIP mining methods.

(2) We proposed methods to estimate feature importance and perturbation potential. Based on them, we have successfully designed the workflow of FIPSER.
(3) We have validated the performance of FIPSER with several experiments, including effectiveness, efficiency, imbalance affection, and model accuracy and fairness after retraining. The results show that FIPSER can successfully improve the effectiveness and efficiency of existing IDIP mining methods and perform well in handling imbalances. Additionally, using the IDIPs generated by FIPSER to retrain models would have a similar impact on model fairness and accuracy as other sampling strategies.
(4) Our implementation for FIPSER can be found on GitHub[1].

2 BACKGROUND

2.1 Individual Discrimination Instance Pair (IDIP)

Individual discrimination occurs when a machine learning model yields different predictions for two valid inputs that only vary in sensitive attributes. These two valid inputs are called Individual Discriminatory Instance Pairs (IDIPs). Let f represent a machine learning model, x represent the input data, and y represent the output. The prediction of a machine learning model for an input sample can be expressed as $y = f(x)$. An input x typically consists of several attributes, denoted by $x_a = \{a_1, a_2, \cdots\}$, where a_i represents the i-th attribute of x. Some attributes are sensitive attributes, denoted by x_s, and the remaining non-sensitive attributes are denoted by x_{ns}. For each sample x, we define its causal fairness set as **Equation 1**.

$$S_{cf}(x) = \{\hat{x} \mid \forall a_n \in x_{ns}, x[a_n] = \hat{x}[a_n]; \exists a_s \in x_s, x[a_s] \neq \hat{x}[a_s]\} \tag{1}$$

The $x[a_n]$ represents the value of x w.r.t attribute a_n. If $\exists \tilde{x} \in S_{cf}(x), f(x) \neq f(\tilde{x})$, then x and $\tilde{x}$ are combined to form an IDIP $(x, \tilde{x})$.

For example, the samples in the diabetes dataset consist of 8 attributes (Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction, Age), among which Age is the sensitive attribute. So for each sample in the dataset, its x_s is {Age}, and its x_{ns} is {Pregnancies, Glucose, BloodPressure, SkinThickness, Insulin, BMI, DiabetesPedigreeFunction}. For sample x = (4, 7, 6, 4, 1, 5, 3, **5**), assuming its prediction result is 0, if there exists any $\tilde{x}$ that differs from x only in the sensitive attribute, for example, $\tilde{x}$ = (4, 7, 6, 4, 1, 5, 3, **3**) and its prediction result is 1, then $(x, \tilde{x})$ is an IDIP.

2.2 Gini Coefficient and Lorenz Curve

The Lorenz curve and Gini coefficient are two indicators commonly used to observe imbalance. **Figure 2** is an example that illustrates income inequality using Lorenz curves and Gini coefficients. Plot a curve with the horizontal axis representing the percentage of the cumulative population, sorted by income from lowest to highest, and the vertical axis representing the percentage of cumulative

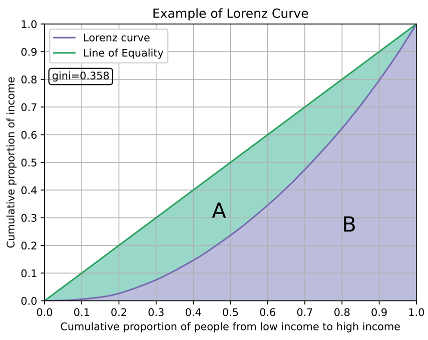

Figure 2: An example for Lorenz curve and Gini coefficient. The purple line is the Lorenz curve. The green line is called the line of equality, which means everyone's income is exactly the same.

income. That curve is the Lorenz curve. The diagonal line in the graph means that everyone has the same income, so this diagonal line is also called the line of equality (LoE). The larger the income inequality, the greater the gap between the Lorenz curve and the line of equality, bringing the Lorenz curve closer to the right bottom corner.

$$G = \frac{S_A}{S_A + S_B} \tag{2}$$

Based on the Lorenz curve, we can calculate the Gini coefficient. Denote the Gini coefficient as G. Its definition is given by **Equation 2**, where S_A represents the area of region A, and S_B represents the area of region B. The larger the Gini coefficient, the greater the income gap between low-income and high-income populations, thus representing a more imbalanced income among people.

2.3 Notations of Neural Network

There are many kinds of architectures for neural networks. Among them, the fully connected neural network is one of the most basic architectures.

$$y^c = f_{nn}(x^d; W; B) \tag{3}$$

The inference process of the fully connected network can be represented by **Equation 3**, where f_{nn} denotes the neural network. x is a d-dimensional row vector, which is an input sample of the network, W is a set of weight matrices, B is a set of bias vectors, and y is the prediction output, which is a c-dimensional row vector.

$$r^v_{k+1} = f_{op}(r^u_k \times w^{u\times v}_{k+1} + b^v_{k+1}) \tag{4}$$

In addition, there are numerous hidden layers inside a network. The $k+1$-th hidden layer takes the output of the k-th layer as its input, then computes and passes its output to the next layer. This process can be represented by **Equation 4**, where f_{op} represents the function of the layer. For most layers, f_{op} is the activation function; for the last layer, it is commonly the softmax function. r^u_k and r^v_{k+1} are the output of k-th and $k+1$-th layer, they are row vectors of dimension u and v. $w^{u\times v}_{k+1} \in W$ is a $u \times v$ weight matrix of layer $k+1$. And $b^v_{k+1} \in B$ is a v-dimensional row vector of the layer.

[1]https://github.com/light-chimes/FIPSER

2.4 Shannon Entropy

For a given discrete random variable X, its Shannon entropy is defined in **Equation 5**.

$$H(X) = -\sum_{x \in X} P(x) \log P(x) \tag{5}$$

Shannon entropy is a good metric for measuring the information of discrete data. Typically, the greater the diversity and dispersion of values in the data, the higher the Shannon entropy.

3 APPROACH

FIPSER can be divided into four phases: feature importance estimation, perturbation potential estimation, feature processing, and sample ranking. **Figure 3** shows the workflow of FIPSER.

In the feature importance estimation phase, we first combine each sample's attribute with the output values on each neuron to utilize information from samples and neural networks. Then, we propose a method to estimate feature importance and calculate the importance of each feature based on it. The purpose of computing feature importance is to enhance the effectiveness of outlier calculation because we can use importance-weighted features instead of original features. The importance-weighted features can express features' impacts on decision-making.

In the phase dedicated to estimating perturbation potential, our goal is to determine the possibility of samples undergoing label changes as a result of being perturbed. Existing methods typically use a two-stage search strategy to search IDIP, which includes global and local search stages. Our analysis of existing methods indicates that samples producing IDIP by only altering sensitive attributes are more effective seeds for the global search stage, which could be leveraged to enhance the IDIP mining effectiveness. However, it is too expensive to exhaustively check the model outputs of samples with all possible sensitive attribute values due to the voluminous value combinations of the sensitive attributes. So, we utilize gradient information, values, and bounds of sensitive attributes to estimate the possibility of each sample changing its prediction through gradient-guided perturbation in sensitive attributes.

The feature processing phase aims to improve the effectiveness of outlier score calculation. First, we standardize the feature columns to remove the effects of data scales. After that, each feature is weighted based on its importance. We then calculate the Shannon entropy for each feature and filter out features with low entropy based on a threshold. For the remaining features, we perform adaptive feature merging, computing distances between features and merging features whose distances are below a threshold. The threshold is determined by an adaptive adjustment strategy. The reason for doing so is to prevent the loss of information for important features. The value of merged features is the average of the original features, and the merged importance is the sum of the original importance values. After that, we will keep some of the most important features and drop others.

The remaining features are used to calculate outliers in the sample ranking phase. We use a distance-based method, computing the outlier score as the average distance from each data point to all other points. Then, we divide samples into multiple sections by their perturbation potential, ensuring that samples within each section are sorted by outlier scores while samples between different sections are sorted by their perturbation potential; this will finally yield the sorting result.

3.1 Feature Importance Estimation

Let $D^{l \times d}$ represent the tabular dataset with l samples and d attributes. $N^{l \times n}$ denotes neurons' output for each sample; it also has l rows, and it has n columns, which is the neuron number of the network. Then, we can concatenate $D^{l \times d}$ and $N^{l \times n}$, that will result in $F^{l \times (d+n)}$, which we called the feature of samples. Additionally, since the structure of $D^{l \times d}$ is similar to $N^{l \times n}$, we can treat $D^{l \times d}$ as the output value of neurons in a virtual "layer 0", behaving like the previous layer of layer 1. Next, we define f_{mod} in **Equation 6**.

$$f_{mod}(k, s^u, \Delta p) = f_{op}((r_k^u + s^u \times \Delta p) \times w_{k+1}^{u \times v} + b_{k+1}^v) \tag{6}$$

The s^u is a one-hot vector with u elements, which is used to select a particular neuron of layer k, and the Δp is a given perturbation value. The meaning of function f_{mod} is to fetch the output of layer $k+1$ with a perturbation given to a neuron of its previous layer k. With the help of f_{mod}, we define the impact of a neuron in layer k to its next layer $k+1$'s neurons. Our approach to calculating impact is given in **Equation 7**.

$$I_{s_j,k}^v = \frac{f_{mod}(k, s_j^u, \Delta p_{inc}) - f_{mod}(k, s_j^u, \Delta p_{dec})}{\Delta p_{inc} - \Delta p_{dec}} \tag{7}$$

Here, $\Delta p_{inc} > 0$ is a given positive perturbation value to the neuron, $\Delta p_{dec} \leq 0$ is a given negative or zero perturbation value to the neuron, s_j^u is a one-hot vector with the j-th element set to 1. The meaning of **Equation 7** is to catch the change of each neuron's value of layer $k+1$, given a positive and a non-positive perturbation to the j-th neuron of its previous layer k. Practically, we input each sample into neural networks and record the maximum and minimum values of each neuron. We then set Δp_{inc} to $max_{k,j} - \hat{r}_{k,j}$, where $max_{k,j}$ is the observed maximum output value of j-th neuron in layer k, $\hat{r}_{k,j}$ is its current value of that neuron, and similarly set Δp_{dec} to $min_{k,j} - \hat{r}_{k,j}$, where $min_{k,j}$ is the minimum output value of j-th neuron in layer k. **Figure 4** gives an example for impact calculation. Suppose we're calculating the impact of the first neuron in the k-th layer (denoted by $N_{k,1}$) of a neural network on two neurons in the $k+1$-th layer (denoted by $N_{k+1,1}$ and $N_{k+1,2}$), based on a sample x_e. First, we input x_e into the neural network to get the output values of each neuron. Then, we perturb $N_{k,1}$ to its maximum and minimum observed values. Suppose 4 is the maximum value and -2 is the minimum. Next, we compute the outputs of $N_{k+1,1}$ and $N_{k+1,2}$ after the perturbation. Then we can obtain $I_{s_1,k}^2$. It's a 2-dimensional row vector representing the impact of $N_{k,1}$ on $N_{k+1,1}$ and $N_{k+1,2}$.

Lastly, we define feature importance in **Equation 8**.

$$\varepsilon_{k,j} = \frac{1}{n_{k+1}} \times I_{s_j,k}^v \times \begin{bmatrix} \varepsilon_{k+1,1} \\ \varepsilon_{k+1,2} \\ \vdots \\ \varepsilon_{k+1,n} \end{bmatrix} \tag{8}$$

n_{k+1} is the number of neurons in layer $k+1$. Overall, **Equation 8** shows that the feature importance $\varepsilon_{k,j}$ of j-th neuron in layer k is equal to the average value of its impact to its next layer $k+1$,

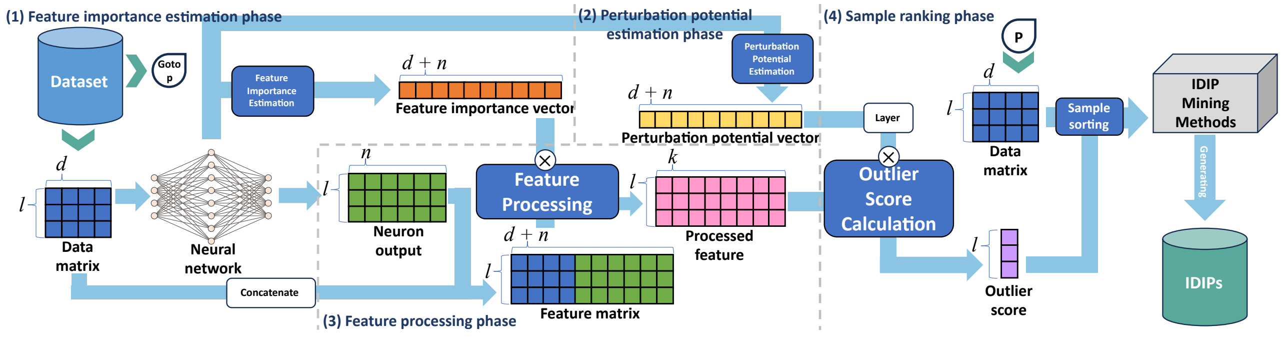
(1) Feature importance estimation phase
Dataset
Goto P
Feature Importance Estimation
d + n
Feature importance vector
(2) Perturbation potential estimation phase
Perturbation Potential Estimation
d + n
Perturbation potential vector
(4) Sample ranking phase
P
Layer
d
l
Data matrix
Sample sorting
IDIP Mining Methods
Generating
IDIPs
d
l
Data matrix
Neural network
n
l
Neuron output
Feature Processing
k
l
Processed feature
Outlier Score Calculation
l
Outlier score
d + n
l
Feature matrix
Concatenate
(3) Feature processing phase

Figure 3: The workflow of FIPSER.

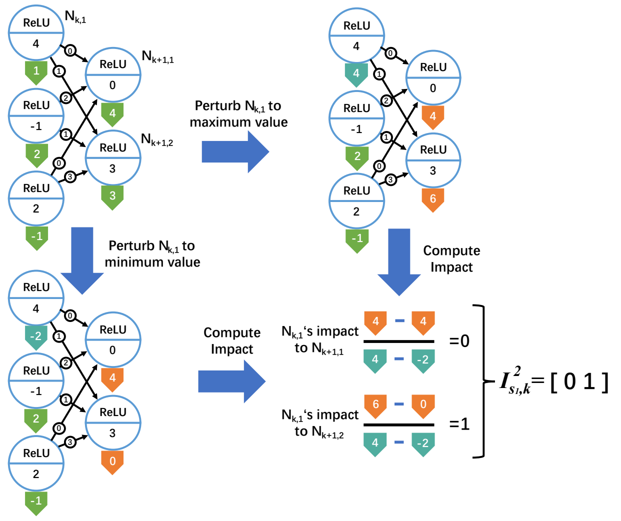
ReLU 4 N_{k,1}
ReLU 0 N_{k+1,1}
ReLU -1
ReLU 3 N_{k+1,2}
ReLU 2
Perturb N_{k,1} to maximum value
Perturb N_{k,1} to minimum value
Compute Impact
Compute Impact
N_{k,1}'s impact to N_{k+1,1}
N_{k,1}'s impact to N_{k+1,2}
=0
=1
I^2_{s_i,k} = [0 1]

Figure 4: This is an example of impact calculation. In the figure, each circle represents a neuron, which is divided into upper and lower semicircles. The upper semicircle records the neuron's activation function. The lower semicircle records the neuron's bias value. The arrow below each circle represents the output value of the neuron above it. The neuron's output for the original sample is indicated by green arrows. Teal arrows represent perturbing a neuron to its maximum or minimum value. Orange arrows represent the new output values of neurons that are affected by the perturbation.

weighted by their feature importance. The feature importance of neurons in layer k depends on the feature importance of its next layer $k+1$. So, we need to give an initial feature importance value to the neurons in the output layer of the network and compute the feature importance of neurons in the previous layers in a back-propagation manner. We concurrently compute feature importance based on each sample in the dataset. This will produce $\varepsilon_{all}^{l\times(d+n)}$, a feature importance matrix with l rows and $d+n$ columns. We take the average value of the feature importance of each row to get an estimation of the average importance of each feature. After that, we can obtain ε_{mean}^{d+n}, which is a $(d+n)$-dimensional row vector that represents the importance of each feature.

3.2 Perturbation Potential Estimation

The perturbation potential τ of a sample is defined in **Equation 9**.

$$\tau(x^d) = \frac{\delta f_{nn}(x^d)_m}{\delta x_{sa,i}} \times sign(\frac{\delta f_{nn}(x^d)_m}{\delta x_{sa,i}}) \times \begin{bmatrix} min(U_{sa,i} - x_{sa,i}, 1) \\ min(L_{sa,i} - x_{sa,i}, 1) \end{bmatrix} \tag{9}$$

x^d is the input sample, a d-dimensional row vector, $f_{nn}(x^d)_m$ represents the largest prediction output of x^d, $x_{sa,i}$ is the i-th sensitive attribute of x^d. The fraction containing δ represents the gradient of the DNN's largest output $f_{nn}(x^d)_m$ with respect to the i-th sensitive attribute $x_{sa,i}$ of the input. The *sign* function gives row vector $\begin{bmatrix}1 & 0\end{bmatrix}$ if it receives non-positive input. Otherwise, it will give row vector $\begin{bmatrix}0 & 1\end{bmatrix}$. $U_{sa,i}$ is the upper bound of sensitive attribute $x_{sa,i}$, and $L_{sa,i}$ is the lower bound of it. For samples whose sensitive attribute's distance to the boundary in the direction of gradient descent is greater or equal to 1, their perturbation potentials are equal to the derivative value of the largest prediction output w.r.t the sensitive attribute. Otherwise, the perturbation potential is blocked because the space for gradient descent on the sensitive attribute is relatively small.

3.3 Feature Processing

The computation process of this phase is shown in **Algorithm 1**. Firstly, in the second line, we standardize each feature (i.e., each column of $F^{l\times(d+n)}$) and multiply them with their corresponding feature importance ε_{mean}. The standardization aims to reduce the influence of varying data scales. From line 4 to line 6, we calculate the Shannon entropy of each feature and drop features whose entropy is smaller than threshold ρ_{en}. For feature merging, we proposed an adaptive feature merging strategy, presented in lines 8 to 21. The strategy first tries a relatively larger merging distance threshold ρ_{dist} as shown in line 8. Features with distances below the threshold ρ_{dist} will be merged. Then, in lines 13 and 14, we test whether merging features by ρ_{dist} would result in a merged feature importance larger than ρ_{imp}. The merged feature importance is calculated by adding the feature importance of merged features. This test aims to prevent the merging of important features from losing too much information. If all the merged importance is smaller than the threshold, features will be successfully merged in line 21; their merged feature values will be the average of the original features. Otherwise, as shown in lines 15 to 17, we will reduce ρ_{dist} and repeat the process. After that, we select several of the most important

Algorithm 1 Feature Processing

```
Input: $F$ is a matrix, represents the original feature of each sample, $\varepsilon_{mean}$ is a row vector, represents the importance of each feature, $\rho_{en}$ is the threshold used in entropy selection, $\rho_{dist}$ is the distance threshold of feature merging, $\rho_{imp}$ is the importance threshold of merged features, $r_{decay}$ is the decay rate of threshold of feature merging, $\rho_{acc}$ is the accumulate importance threshold of second feature selection
Output: $F'$ is the feature after processing
1:  for $col\_idx$ in $F$ do
2:      $F[col\_idx] \leftarrow standardize(F[col\_idx])$
3:      $F[col\_idx] \leftarrow F[col\_idx] * \varepsilon_{mean}[col\_idx]$
4:      if $entropy(F[col\_idx]) \leq \rho_{en}$ then
5:          $delete\ F[col\_idx]$
6:      end if
7:  end for
8:  $indices\_cluster \leftarrow merge\_column(F, \rho_{dist})$
9:  $merge\_success \leftarrow False$
10: while not $merge\_success$ do
11:     $merge\_success \leftarrow True$
12:     for $indices$ in $indices\_cluster$ do
13:         $merged\_importance \leftarrow sum(\varepsilon_{mean}[indices])$
14:         if $merged\_importance > \rho_{imp}$ then
15:             $merge\_success, \rho_{dist} \leftarrow False, \rho_{dist} * r_{decay}$
16:             $indices\_cluster \leftarrow merge\_column(F, \rho_{dist})$
17:             break
18:         end if
19:     end for
20: end while
21: $F, \varepsilon_{mean} \leftarrow merge(F, \varepsilon_{mean}, indices\_cluster)$
22: return $top\_k\_importance(F, \varepsilon_{mean}, \rho_{acc})$
```

features whose accumulated sum of feature importance reached threshold ρ_{acc} and return them.

3.4 Sample Ranking

The perturbation potential can serve as a good guide to improve the success rate of the global search stage, which is crucial to enabling the follow-up local search stage. Moreover, weighted features provided by feature processing phases are effective in finding samples that can generate more IDIPs. So, to fully utilize these two indicators, we proposed **Algorithm 2**.

First, we compute each sample's outlier scores by measuring their average distance to other samples based on their features. Then, we compute and decide every sample's potential section at line 2. The section number of each sample is equal to the logarithm of their potential score to the given base $b_{section}$. The larger the base, the fewer sections there will be, resulting in a smaller impact of perturbation potential on sample sorting. But, correspondingly, the influence of outlier scores on sample sorting will become more significant. Then, in lines 3 and 4, we use the section number to widen the difference of outlier scores between samples from different sections. This will result in samples with larger section numbers having larger outlier scores so that they will be put at the front of the sorted sample sequence. Besides, scores added to the samples in the same sections are identical, so the sample order inside a section will be determined only by their original outlier scores. In line 5, we sort samples by their outlier scores in descending order and finally obtain the sorted samples in line 6.

Algorithm 2 Sample sorting

```
Input: $b_{section}$ is the base of the logarithm, $\tau$ is the perturbation potential of each sample, $sample$ represents the original samples of the given dataset, and $F$ is a matrix, represents the feature of the dataset given by the feature processing phase
Output: $sample$ represents the sorted samples
1: $outlier\_score \leftarrow avg\_row\_distance(F)$
2: $potential\_section \leftarrow \log_{b_{section}}(\tau)$
3: $section\_score \leftarrow max(outlier\_score) * potential\_section$
4: $outlier\_score \leftarrow outlier\_score + section\_score$
5: $sample \leftarrow sort\ samples\ by\ outlier\_score$
6: return $sample$
```

Table 1: Basic information about datasets used in our experiments. Column Length is the number of samples in the dataset. Column #Dimension is the column number of datasets. Column Name represents names of sensitive attributes. Column Index represents indices of sensitive attributes (starting from 1). Column Range represents the range of values for sensitive attributes.

Dataset	Length	#Dimension	Name	Index	Range
census[30]	32561	13	Age	1	[1,9]
			Race	8	[0,4]
			Sex	9	[0,1]
compas[41]	7214	12	Sex	1	[0,1]
			Age	2	[0,2]
			Race	3	[0,1]
bank[35]	45211	16	Age	1	[1,9]
credit[23]	600	20	Sex	9	[0,1]
			Age	13	[1,8]
default[51]	13636	23	Sex	2	[0,1]
			Age	5	[2,7]
diabetes[26]	768	8	Age	8	[1,9]
heart[24]	297	13	Age	1	[1,7]
			Sex	2	[0,1]
meps15[1]	15830	137	Age	1	[1,9]
			Race	2	[0,1]
			Sex	10	[0,1]
meps16[1]	15675	137	Age	1	[1,9]
			Race	2	[0,1]
			Sex	10	[0,1]
students[12]	1044	32	Sex	2	[0,1]
			Age	3	[15,22]

4 EXPERIMENT

4.1 Dataset

To compare with existing methods, we chose the same datasets as DICE for experiments, which are also commonly used datasets in recent fairness research. The experimental datasets of ADF, EIDIG, and NF are also included. The basic information of these datasets is presented in **Table 1**. For the model retraining experiment, we split each dataset into a training set and a test set; 80% of the dataset goes into the training set, and 20% of the dataset goes into the testing set.

Table 2: The comparison of the quantity of IDIPs generated by different sampling methods. The Column Index refers to the indices of sensitive attributes, starting from 1. For each dataset and sensitive attributes, we conduct three seed selection methods M_r, M_c and M_f, based on 4 SOTA IDIP mining methods, ADF, EIDIG, NF, and DICE. M_r represents randomly sampling, M_c represents the clustering and sampling with round-robin strategy and M_f represents the FIPSER sampling method. For each experiment, we compare the number of effective seeds in the global search stage, which is denoted by #G, and the generated IDIP quantity in total, which is denoted by #T. Seeds here refer to the input samples, as each IDIP mining method uses them as seeds for the global search stage. In addition, since we use 100 samples as input, the maximum value of #G is 100.

Dataset	**Index**	ADF						EIDIG						NF						DICE					
		M_r		M_c		M_f		M_r		M_c		M_f		M_r		M_c		M_f		M_r		M_c		M_f	
		#G	#T	#G	#T	#G	#T	#G	#T	#G	#T	#G	#T	#G	#T	#G	#T	#G	#T	#G	#T	#G	#T	#G	#T
census	1	82	32459	70	26434	**99**	**44754**	49	19770	48	20245	**96**	**40725**	99	42560	**100**	41783	**100**	**42677**	96	408542	87	359943	**100**	**492163**
	8	63	22058	62	20310	**82**	**24888**	32	9780	34	10694	**73**	**25319**	**100**	27173	99	33451	**100**	**37216**	86	296316	81	270633	**97**	**330496**
	9	51	7634	47	6069	**73**	**11627**	20	3615	28	4507	**53**	**8527**	94	13672	92	13889	**98**	**16580**	87	155019	79	166588	**98**	**196996**
compas	1	82	10486	74	8553	**96**	**11329**	**87**	**8576**	80	8110	77	8190	**100**	9984	99	**10333**	99	9769	63	28864	67	29614	**80**	**44255**
	2	82	14701	92	17869	**100**	**18472**	76	**12216**	83	11045	**89**	10172	**100**	**14610**	**100**	12883	99	13012	70	78375	77	81715	**91**	**93770**
	3	96	9691	**99**	9018	**99**	**9809**	95	**8319**	**98**	7191	**98**	6484	**100**	6570	**100**	**7347**	**100**	7233	54	25731	71	32577	**81**	**43246**
bank	1	87	32716	90	31712	**98**	**43444**	33	12509	21	8797	**68**	**30845**	**100**	41591	**100**	37180	**100**	**44539**	98	472461	96	462924	**100**	**571408**
credit	9	77	13110	63	10662	**92**	**15288**	21	4151	13	2777	**62**	**11665**	97	13489	92	12133	**100**	**14885**	90	99430	78	76799	**96**	**123770**
	13	**100**	38980	98	39172	**100**	**41710**	88	35033	76	29728	**91**	**37354**	**100**	38388	**100**	38772	**100**	**39813**	97	283052	95	258443	**100**	**338849**
default	2	73	10566	83	12846	**94**	**16483**	19	2586	39	4833	**70**	**9764**	**99**	**12870**	**99**	12846	96	12256	98	133779	94	138649	**99**	**219869**
	5	57	20600	56	21238	**78**	**31735**	17	7269	**24**	8344	**24**	**10124**	**97**	**26169**	93	23910	**97**	26046	**100**	441064	99	434917	**100**	**552293**
diabetes	8	98	55565	96	54052	**100**	**56610**	79	42777	77	41246	**89**	**46640**	**100**	**56148**	**100**	56009	**100**	55015	**100**	431472	**100**	429749	**100**	**435879**
heart	1	68	23464	73	26013	**99**	**35519**	57	22889	70	27610	**84**	**31845**	**100**	**46003**	**100**	45058	**100**	42293	81	252277	80	247627	**100**	**356971**
	2	80	5867	83	**7661**	**86**	7488	**88**	**10082**	**88**	9515	85	8153	**100**	**8419**	**100**	8074	**100**	6815	**47**	82337	**47**	73707	44	**96005**
meps15	1	32	5268	23	3407	**67**	**9267**	22	2965	33	6145	**80**	**10594**	86	11573	88	11074	**93**	**13142**	63	31603	55	29736	**97**	**41573**
	2	61	5387	46	3973	**64**	**6597**	62	5743	53	4332	**64**	**6638**	**100**	7335	**100**	7841	**100**	**8523**	50	16842	56	19570	**93**	**31042**
	10	50	3135	56	3875	**85**	**6616**	47	4075	59	5302	**84**	**7656**	**100**	5766	**100**	5638	**100**	**6142**	54	22339	57	20757	**90**	**31813**
meps16	1	27	4653	25	4748	**65**	**9314**	24	4206	22	3442	**59**	**9851**	96	14344	96	13196	**98**	**14464**	59	44970	60	42223	**99**	**80626**
	2	85	11052	46	5240	**88**	**11917**	75	9182	63	7583	**83**	**12610**	**100**	**12761**	**100**	12193	**100**	11887	69	39667	60	33174	**99**	**56129**
	10	61	3938	71	5418	**86**	**7509**	41	3497	56	4033	**82**	**8598**	**100**	5727	**100**	6029	**100**	**7770**	60	22446	58	24649	**98**	**44329**
students	2	55	3228	61	4625	**91**	**7813**	24	1909	20	1887	**47**	**5052**	**100**	6288	**100**	6856	**100**	**8618**	38	18711	36	18409	**62**	**38536**
	3	80	14767	74	12364	**100**	**26151**	72	15759	68	12412	**98**	**25024**	**100**	17195	**100**	15765	**100**	**27908**	64	71008	53	70546	**81**	**96808**

4.2 Models

For the fairness testing, we use the same neural network structure as DICE, i.e., a fully connected neural network with layer widths (64,32,16,8,4,2). We use ReLU as the activation function. To correct the generated IDIP, we integrate 5 machine learning methods into one ensemble model to vote the corrected prediction. The methods includes k-NN [16], multilayer perceptron [20], support vector machine [11], random forest [22], and Gaussian naive Bayesian models [20].

4.3 Configurations

We set the initial feature importance F of the output layer neurons to 1.0. We randomly sample 3000 data points to compute the feature importance. For feature selection, we set the base of entropy computation to 2.0, the threshold ρ_{en} to 0.05, and ρ_{acc} to 0.3. For feature merging, we set ρ_{dist} to 1.5% of dataset length, r_{decay} to 0.9, ρ_{imp} to 0.15. The base $b_{section}$ used in the sample ranking phase is set to 32. We set epochs as 1000, batch size as 128, and learning rate as 0.01 to train networks. For the retraining, we keep all other training parameters unchanged and reduce the number of training epochs to 500. In addition, the number of IDIPs used for retraining is half the size of the training set. We mix these IDIPs with the training set to conduct retraining. For fairness evaluation, we randomly generate 100,000,000 sample pairs.

4.4 Environments

We implement FIPSER with Python 3.8.18, other key packages include: numpy v1.22.0, tensorflow v2.7.0, scikit-learn v0.22.2.post1, aif360 v0.4.0. We conducted our experiments on a Windows 11 computer with 16GB RAM, a 24-core Intel i7-13700F CPU, and an NVIDIA GeForce RTX 4060 Ti GPU.

4.5 Baselines

As FIPSER is a pre-processing[2] method, we choose to conduct our experiment based on four state-of-the-art IDIP mining methods: ADF, EIDIG, NF, and DICE. This will help us find out FIPSER's effectiveness and efficiency in improving the performance of existing IDIP methods. Moreover, we compare FIPSER (denoted by M_f) with two other sampling strategies: random sampling (denoted by M_r) and clustering strategy (denoted by M_c). Existing IDIP mining methods typically use the latter as their seed sampling strategy.

4.6 Research Questions

To further understand and evaluate our approach, we would like to answer the following questions:

- **RQ1**: How effective and efficient is FIPSER?
- **RQ2**: How imbalanced is the IDIP quantity? Can FIPSER counteract the imbalance?
- **RQ3**: How do the IDIPs generated by FIPSER affect the fairness and accuracy of neural networks after retraining?

4.7 Results

4.7.1 RQ1 (effectiveness and efficiency). To evaluate the effectiveness of FIPSER, we compare the number of effective seeds and the

[2]Pre-processing mentioned here differs from the concept introduced in the Introduction section. Here, it refers to FIPSER not altering the existing IDIP mining methods but just selecting appropriate seed samples to enhance their overall performance.

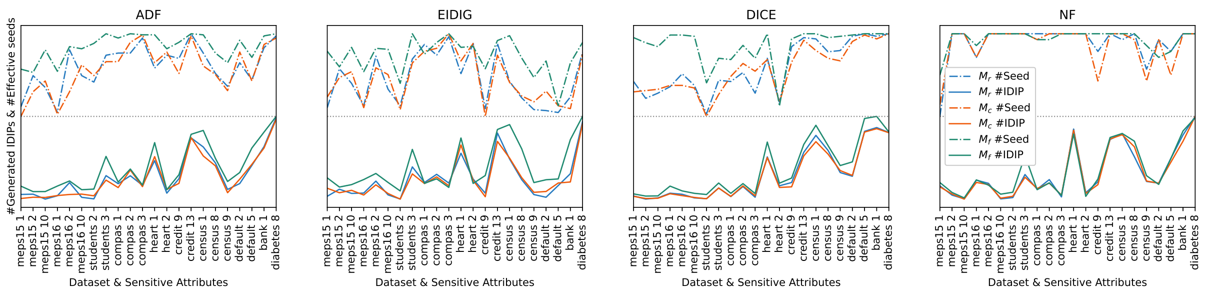

Figure 5: Comparison of the effectiveness of different sampling methods. Dash-dotted lines above the gray dotted line represent the number of effective global search seeds, with higher lines indicating a larger number of effective seeds. Solid lines below the gray dotted line represent the total number of generated IDIPs, with higher lines indicating a larger IDIP quantity.

Table 3: Comparison of IDIP generating speed (pairs per second). The Column Index refers to the indices of sensitive attributes, starting from 1. V_r, V_c, and V_f are IDIP generating speeds of random sampling, clustering strategy, and FIPSER, respectively. ADF, EIDIG, NF, and DICE represent the IDIP mining methods corresponding to the respective experiments.

Dataset	Index	ADF			EIDIG			NF			DICE		
		V_r	V_c	V_f	V_r	V_c	V_f	V_r	V_c	V_f	V_r	V_c	V_f
census	1	10.495	**24.379**	16.787	23.057	28.484	**30.143**	18.112	27.967	**28.903**	152.783	128.094	**166.722**
	8	13.473	**31.603**	16.918	14.570	30.376	**35.449**	10.985	**33.296**	27.172	140.701	71.615	**154.654**
	9	**23.614**	21.441	18.108	14.995	**26.575**	23.220	19.839	25.150	**28.450**	96.465	74.237	**109.931**
compas	1	**24.490**	22.191	22.375	12.554	21.327	**21.535**	20.327	**20.673**	19.325	53.651	52.229	**56.376**
	2	23.303	**28.891**	27.828	13.537	**20.410**	17.937	**22.015**	18.982	19.544	**90.294**	89.014	81.824
	3	14.927	18.223	**19.557**	15.061	**15.994**	14.787	13.477	**14.901**	14.695	**55.217**	52.628	48.976
bank	1	4.957	12.074	**18.538**	14.962	16.232	**18.624**	11.847	14.231	**18.583**	69.500	32.753	**74.132**
credit	9	9.455	20.743	**21.058**	17.170	22.165	**24.795**	17.695	16.092	**18.908**	85.789	**86.098**	82.513
	13	7.758	19.286	**20.735**	14.395	19.276	**21.161**	15.436	18.061	**19.108**	105.933	**108.226**	105.560
default	2	14.609	16.298	**17.761**	5.912	13.546	**13.945**	**14.219**	13.799	13.059	40.911	41.951	**55.438**
	5	16.231	17.533	**18.707**	12.402	**15.930**	15.244	**11.426**	11.371	11.012	72.058	75.902	**90.288**
diabetes	8	62.765	62.529	**63.130**	55.121	**62.357**	61.495	**64.251**	62.862	61.704	207.438	**215.629**	202.734
heart	1	29.465	33.009	**33.096**	16.426	**38.760**	34.927	**42.313**	38.988	36.101	121.755	**125.763**	125.209
	2	13.088	**16.744**	15.972	14.913	**21.678**	18.943	**15.972**	15.157	12.860	**119.156**	117.743	114.292
meps15	1	**0.975**	0.875	0.816	**0.763**	0.650	0.741	**0.730**	0.716	0.486	5.785	**8.388**	6.947
	2	1.622	1.629	**1.846**	**1.123**	0.633	0.799	1.401	1.482	**1.489**	3.671	**7.654**	6.213
	10	1.128	1.293	**1.457**	**1.341**	1.162	0.653	1.098	1.065	**1.148**	7.429	**8.156**	6.557
meps16	1	1.073	**1.172**	0.821	0.781	0.957	**1.028**	0.424	0.800	**0.865**	7.844	**7.967**	7.256
	2	2.396	2.151	**2.525**	2.340	2.271	**2.787**	1.087	**2.280**	2.240	5.938	**7.897**	6.729
	10	1.184	1.445	**1.632**	1.569	1.345	**1.954**	0.671	**1.131**	0.866	7.036	**7.093**	6.203
students	2	4.650	6.149	**7.072**	5.710	7.248	**8.615**	5.274	5.694	**7.191**	**40.413**	39.420	37.523
	3	4.657	4.280	**7.411**	5.395	4.784	**7.518**	4.468	3.989	**7.806**	59.371	**59.988**	56.779

number of IDIPs generated by each sampling method. The effective seed here means the seeds that were successfully used in the global search stage of IDIP mining. **Figure 5** provides a summary of effectiveness experiments. **Table 2** provides detailed records of the experimental results. In addition, due to the small enough time overhead of FIPSER computation (averaging about 0.778% of the IDIP mining time cost), the results do not include the time overhead of FIPSER. The result shows that FIPSER significantly improves the number of effective seeds in the global search stage compared to random sampling and the clustering strategy. For ADF, EIDIG, NF, and DICE, FIPSER boosts their $\#G$ by 34.05%, 81.94%, 0.62%, and 28.97%, respectively, with an average improvement of 36.40% over random sampling. When compared with the clustering strategy, improvements are 43.60%, 75.94%, 1.10%, and 31.46%, with an average of 38.03%. In terms of the total number of generated IDIPs, the improvements compared to random sampling are 46.02%, 88.14%, 9.44%, and 42.40%, averaging 46.50%. Compared to the clustering strategy, the rates are 53.77%, 84.06%, 10.35%, and 44.97%, averaging 48.29%. When combining FIPSER with the latest SOTA IDIP mining method, DICE, FIPSER increases the total IDIP count by about 45%. Its combination with EIDIG yields an even more impressive improvement, about 84%. Furthermore, instead of increasing the quantity of generated IDIP, the clustering strategy even reduces the quantity of IDIP in some scenarios when compared with random sampling. For efficiency, we compare the IDIP generating speed of each sampling method. **Table 3** provides the results of efficiency experiments. For ADF, EIDIG, NF, and DICE, FIPSER improved their IDIP generating speed by 40.79%, 38.16%, 27.66%, and 6.24%, respectively, averaging 28.21%, to the speed of random sampling. For clustering strategy, they are 4.28%, 6.82%, 4.33%, and 11.21%, averaging 6.66%. Overall, the clustering strategy can generate IDIPs faster than random sampling, but FIPSER is more efficient.

Answer RQ1: Generally, combining existing IDIP mining methods with FIPSER can enhance their effectiveness and efficiency. Compared to random sampling, FIPSER can, on average, increase the total generation number of IDIP by 46.50% and the generation speed by 28.21%. Compared to the clustering strategy, these values are 48.29% and 6.66%, respectively.

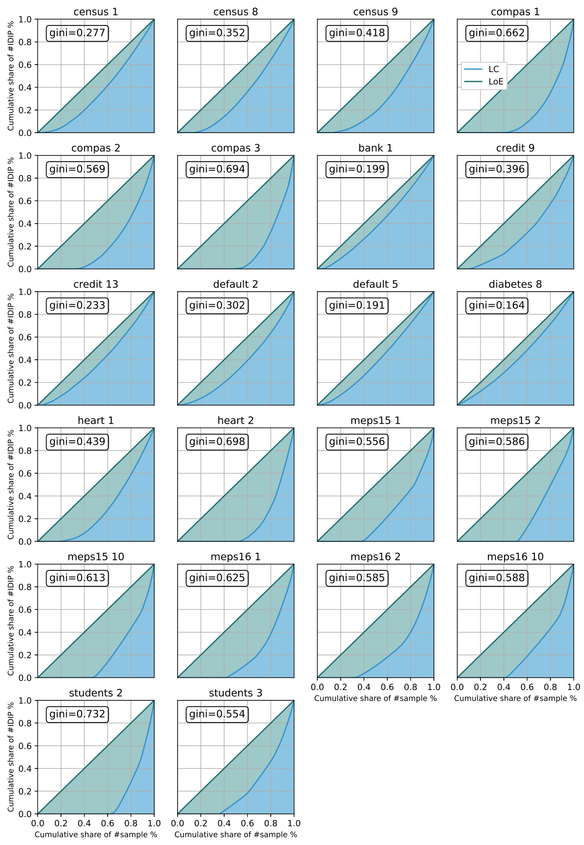

Figure 6: The Lorenz curve and Gini coefficient of each dataset and each sensitive attribute. The result in each figure is calculated based on the IDIP mining result of 100 randomly sampled seeds. The title of each plot indicates the datasets and the indices of one of their sensitive attributes used for IDIP mining.

4.7.2 RQ2 (imbalance). We use the Gini coefficient and Lorenz curve to describe the imbalance in the IDIP generation quantity of samples. We conduct IDIP mining with DICE based on random sampling. The results are shown in **Figure 6**. The size of the teal area and the Gini coefficient reflect the imbalance of IDIP quantity

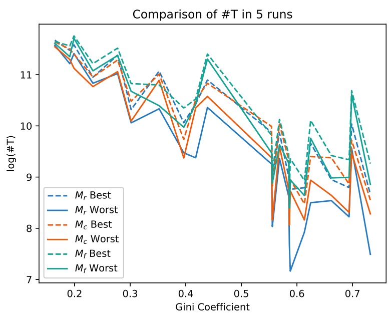

Figure 7: The dashed and solid lines represent the best and worst cases of the total number of IDIP (denoted by #T) generated by different sampling methods among 5 runs, respectively. For better visualization, the values on the vertical axis have been transformed by logarithm.

across different datasets and sensitive attributes. It can be observed that the imbalance degree varies across different datasets and sensitive attributes. Datasets such as compas and meps exhibit severe imbalances. Moreover, different sensitive attributes within the same dataset, such as credit 9 and credit 13, or the same sensitive attribute across different datasets, such as census 1 and compas 2 (both sensitive attributes are Age), exhibit varying degrees of imbalance.

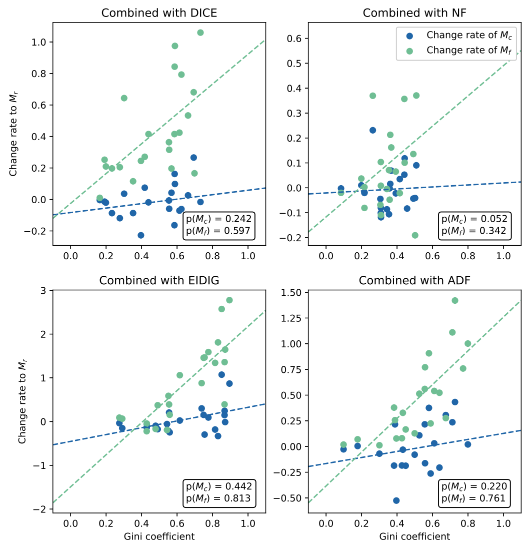

Figure 8: The correlation analysis between the Gini coefficient and the change rate of IDIP generation quantity, compared to random sampling. $p(M_c)$ and $p(M_f)$ denote the Pearson correlation coefficients between the change rate and the Gini coefficient of clustering and FIPSER, respectively.

To find out whether FIPSER is effective in counteracting imbalance, we first compare the number of IDIPs generated by different sampling methods under different Gini coefficients. We experimented for 5 runs using each sampling method and observed the best and worst cases of IDIP mining quantity across the 5 runs. The results are shown in **Figure 7**. It can be observed that in most cases, FIPSER performs better than random sampling and clustering strategies. It is also noticeable that as the Gini coefficient increases, the overall performance of all sampling methods decreases because it becomes harder to find samples that can generate more IDIPs. Additionally, though not very pronounced, it seems that as the Gini coefficient increases, there appears to be a widening performance gap between FIPSER and the other two sampling methods. We conducted a more in-depth analysis of this phenomenon. We observe the change rate of #T (the total IDIP generation quantity based on a particular sampling method) for different sampling methods compared to random sampling. We plot the correlation between this change rate and the Gini coefficient and calculate their Pearson correlation coefficient. The results are presented in **Figure 8**. It is shown that the clustering strategy shows a relatively weak correlation. When combining FIPSER with any IDIP mining method, we observe stronger correlations. This is reasonable because an effective sampling method should identify samples that can generate more IDIPs. If a method can handle imbalance better than random sampling, it should result in a higher change rate of #T in imbalanced datasets and sensitive attributes. In other words, the change rate should have a stronger positive correlation with the degree of imbalance represented by the Gini coefficient. This result suggests that FIPSER is a more effective sampling strategy for counteracting the imbalance.

Answer RQ2: The imbalance degree of IDIP generation quantity varies across datasets and sensitive attributes. Generally, the more imbalanced the dataset and sensitive attributes are, the lower the effectiveness of the sampling method. FIPSER can better counteract imbalance compared to the clustering method.

4.7.3 RQ3 (retraining). We randomly generate one hundred million sample pairs and evaluate the proportion of non-IDIPs among these pairs as the fairness metric. We also measure the model's accuracy and fairness before and after retraining. The experimental results are shown in **Table 4**. The result indicates that, on average, if we retrain the model with IDIP generated by clustering strategy, it leads to a decrease in accuracy of approximately 8.72% and an improvement in fairness of about 7.06%. For FIPSER, these values are 8.37% and 7.29%, respectively. This means that whether using IDIP generated by clustering method or FIPSER to retrain the model, the improvement in fairness and the decrease in accuracy are similar.

Answer RQ3: After retraining models using IDIPs generated by FIPSER, when compared to the clustering method, models retrained based on FIPSER show no significant difference in the impact on model fairness and accuracy.

Table 4: Comparison of model accuracy and fairness before and after retraining. Column Origin represents the accuracy and fairness of the original pre-trained model. Columns M_c retrained and M_f retrained represent the corresponding results of models retrained by clustering and FIPSER's generated IDIP, respectively. r_{acc} represents model accuracy, and r_{fair} represents the proportion of non-IDIPs among one hundred million randomly generated sample pairs.

Dataset	Origin		M_c retrained		M_f retrained	
	r_{acc}	r_{fair}	r_{acc}	r_{fair}	r_{acc}	r_{fair}
census	0.8394	0.9503	0.8179	0.9916	**0.8412**	**0.9944**
credit	0.6417	0.9351	**0.4167**	**1.0000**	**0.4167**	**1.0000**
bank	0.8888	0.9928	0.8794	0.9993	**0.8914**	**0.9999**
compas	0.9640	0.9445	**0.9716**	0.9433	0.9660	**0.9711**
default	0.8188	0.9995	0.8082	0.9999	**0.8155**	**1.0000**
heart	0.7667	0.9418	**0.8333**	**0.9732**	0.7500	0.9670
diabetes	0.6818	0.7414	**0.3506**	**1.0000**	**0.3506**	**1.0000**
students	0.9139	0.9356	0.8086	**1.0000**	**0.9234**	0.9861
meps15	0.8108	0.9188	**0.8268**	0.9872	0.8149	**0.9938**
meps16	0.8075	0.9280	**0.8186**	0.9858	0.8096	**0.9893**

5 RELATED WORK

5.1 Fairness Testing

Fairness testing aims to measure fairness and find out fairness defects in softwares [2, 3, 10, 17, 46, 54]. Recent research typically categorizes methods into pre-processing, in-processing, and post-processing methods based on their intervention stages during model development. Data pre-processing [27] is a commonly used pre-processing method nowadays. Pre-processing methods finish their jobs before model training. In-processing methods, such as convex regularizers [7], are designed to enhance the fairness of models during the training process. Post-processing methods [19, 40] fix models' fairness defects after they are trained. Themis [5] first utilized random sampling for fairness testing. It tries to measure the fairness of software by counting the frequency of individual discrimination instances in input space. But it is not efficient enough. AEQUITAS [47] proposed a two-stage random sampling method, which includes a global search stage and a local search stage, to generate more discrimination instances. SymbGen [3] combines symbolic execution along with local explainability for the generation of effective test cases. These methods are not designed for neural networks. Later, ADF [57] leverages adversarial sample mining to design an effective IDIP mining method for deep neural networks. To improve its efficiency, EIDIG [55] reduced the gradient computation cost and boosted the search with momentum. NF [58] tries to interpret neurons in neural networks to find out the biased neurons and enhance the searching of IDIP based on them. Recently, DICE [34] proposed an information-theoretic model to better mine IDIP and debug neural networks. Our method can be combined with these methods to improve their efficiency and effectiveness.

5.2 Causal Analysis

Causal analysis has been utilized to generate explanations for machine learning algorithms [25, 45, 49, 56]. Previous studies have focused on developing interpretable models that can provide insights into the decision-making process of machine learning models. For instance, Narendra et al. [37] treated deep neural networks as Structural Causal Models (SCMs) to estimate the causal impact of each model component on the output. Chattophadhyay et al. [9] proposed a scalable causal approach to estimate the individual causal effects of each feature on the model output. Causal inference has also been employed in studies related to fairness in machine learning models. Kusner et al. [31] introduced an approach to assess the fairness of a machine learning model based on counterfactual fairness. Zhang et al. [53] proposed a metric based on causal explanations to quantitatively measure the fairness of an algorithm. Sun et al. [45] proposed a causality-based technique for repairing neural networks for various properties. Zhang et al. [56] proposed an approach that adaptively chooses the fairness-improving method based on causality analysis. Unlike these works, our approach focuses on the imbalance in datasets and leverages information in samples, such as outliers, to improve the effectiveness.

5.3 Test Case Prioritization

Regression testing [32] is a commonly employed software testing technique, but the time overhead associated with a large amount of test cases may be unsustainable [44]. As a result, many studies have focused on devising non-full regression testing strategies, including test suite reduction (TSR) [38], test case selection (TCS) [14], and test case prioritization (TCP) [43] methods. TSR and TCS need to discard test cases, TCP, however, only adjusts the order of test cases in which they are executed. This makes TCP technology safer than TSR and TCS, making it a more popular technique [6, 52]. Rothermel et al. [42] first proposed the TCP problem. Subsequently, considering the necessary data, the dynamic or static execution conditions, and the availability of test cases, TCP can be categorized into white-box TCP and black-box TCP [21]. White-box TCP prioritizes test cases through dynamic or static analysis of the source code. In contrast, Black-box TCP prioritizes test cases based on their diversity when lacking access to the source code information. For our work, similar to white-box TCP, we extract information from neural networks to help seed prioritization. But unlike white-box TCP, our work focuses on analyzing and testing neural networks.

6 CONCLUSION

We propose a feature importance and perturbation potential-based seed prioritization method, FIPSER, to sort and select seed samples in IDIP generation. FIPSER consists of four phases. In the first phase, FIPSER utilizes the information from samples and neural networks to estimate the importance of features. In the second phase, FIPSER utilizes the gradient of the model output w.r.t sensitive attributes to estimate the perturbation potential of samples. In the third phase, features are processed by weighting, selection, and merging procedures. In the last phase, FIPSER utilizes the outputs of the previous three stages to enhance the effectiveness of outlier score calculation and yields sorting results based on the outlier scores. Experimental results demonstrate that FIPSER achieves state-of-the-art performance in seed selection for IDIP search algorithms, such as ADF, EIDIG, and DICE. Notably, FIPSER enhances both the effectiveness and efficiency of these algorithms, resulting in the discovery of a greater number of IDIPs than previously possible.

ACKNOWLEDGMENTS

This work is supported by the National Natural Science Foundation of China for Young Scientists (Grant 62202166, NSFC), the Shanghai Pujiang Program (Category D) (Grant 22PJD021), the CCF-Huawei Populus Grove Fund (Grant CCF-HuaweiTC202304, CCF-Huawei), and the Chenguang Program of Shanghai Education Development Foundation and Shanghai Municipal Education Commission (Grant 23CGA33).

REFERENCES

[1] 2014. *Medical expenditure panel survey.* https://meps.ahrq.gov/mepsweb/

[2] Julius Adebayo and Lalana Kagal. 2016. Iterative orthogonal feature projection for diagnosing bias in black-box models. *arXiv preprint arXiv:1611.04967* (2016).

[3] Aniya Agarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. 2018. Automated test generation to detect individual discrimination in AI models. *arXiv preprint arXiv:1809.03260* (2018).

[4] Aniya Aggarwal, Pranay Lohia, Seema Nagar, Kuntal Dey, and Diptikalyan Saha. 2019. Black box fairness testing of machine learning models. In *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering.* 625–635.

[5] Rico Angell, Brittany Johnson, Yuriy Brun, and Alexandra Meliou. 2018. Themis: Automatically testing software for discrimination. In *Proceedings of the 2018 26th ACM Joint meeting on european software engineering conference and symposium on the foundations of software engineering.* 871–875.

[6] Anu Bajaj and Om Prakash Sangwan. 2018. A survey on regression testing using nature-inspired approaches. In *2018 4th International Conference on Computing Communication and Automation (ICCCA).* IEEE, 1–5.

[7] Richard Berk, Hoda Heidari, Shahin Jabbari, Matthew Joseph, Michael Kearns, Jamie Morgenstern, Seth Neel, and Aaron Roth. 2017. A convex framework for fair regression. *arXiv preprint arXiv:1706.02409* (2017).

[8] Sumon Biswas and Hridesh Rajan. 2023. Fairify: Fairness verification of neural networks. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 1546–1558.

[9] Aditya Chattopadhyay, Piyushi Manupriya, Anirban Sarkar, and Vineeth N Balasubramanian. 2019. Neural network attributions: A causal perspective. In *International Conference on Machine Learning.* PMLR, 981–990.

[10] Zhenpeng Chen, Jie M Zhang, Max Hort, Federica Sarro, and Mark Harman. 2022. Fairness testing: A comprehensive survey and analysis of trends. *arXiv preprint arXiv:2207.10223* (2022).

[11] Corinna Cortes and Vladimir Vapnik. 1995. Support-vector networks. *Machine learning* 20 (1995), 273–297.

[12] Cortez and Paulo. 2008. Student Performance. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5TG7T.

[13] Jeffrey Dastin. 2022. Amazon scraps secret AI recruiting tool that showed bias against women. In *Ethics of data and analytics.* Auerbach Publications, 296–299.

[14] Priyanka Dhareula and Anita Ganpati. 2015. Prevalent criteria's in regression test case selection techniques: An exploratory study. In *2015 International Conference on Green Computing and Internet of Things (ICGCIoT).* IEEE, 871–876.

[15] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. 2012. Fairness through awareness. In *Proceedings of the 3rd innovations in theoretical computer science conference.* 214–226.

[16] Evelyn Fix and Joseph Lawson Hodges. 1989. Discriminatory analysis. Nonparametric discrimination: Consistency properties. *International Statistical Review/Revue Internationale de Statistique* 57, 3 (1989), 238–247.

[17] Sainyam Galhotra, Yuriy Brun, and Alexandra Meliou. 2017. Fairness testing: testing software for discrimination. In *Proceedings of the 2017 11th Joint meeting on foundations of software engineering.* 498–510.

[18] Xuanqi Gao, Juan Zhai, Shiqing Ma, Chao Shen, Yufei Chen, and Qian Wang. 2022. FairNeuron: improving deep neural network fairness with adversary games on selective neurons. In *Proceedings of the 44th International Conference on Software Engineering.* 921–933.

[19] Moritz Hardt, Eric Price, and Nati Srebro. 2016. Equality of opportunity in supervised learning. *Advances in neural information processing systems* 29 (2016).

[20] Trevor Hastie, Robert Tibshirani, Jerome H Friedman, and Jerome H Friedman. 2009. *The elements of statistical learning: data mining, inference, and prediction.* Vol. 2. Springer.

[21] Christopher Henard, Mike Papadakis, Mark Harman, Yue Jia, and Yves Le Traon. 2016. Comparing white-box and black-box test prioritization. In *Proceedings of the 38th International Conference on Software Engineering.* 523–534.

[22] Tin Kam Ho. 1998. The random subspace method for constructing decision forests. *IEEE transactions on pattern analysis and machine intelligence* 20, 8 (1998), 832–844.

[23] Hofmann and Hans. 1994. Statlog (German Credit Data). UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5NC77.

[24] Janosi, Andras, Steinbrunn, William, Pfisterer, Matthias, and Robert Detrano. 1989. Heart Disease. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C52P4X.

[25] Zhenlan Ji, Pingchuan Ma, Yuanyuan Yuan, and Shuai Wang. 2023. Cc: Causality-aware coverage criterion for deep neural networks. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 1788–1800.

[26] Kahn and Michael. [n. d.]. Diabetes. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5T59G.

[27] Faisal Kamiran and Toon Calders. 2012. Data preprocessing techniques for classification without discrimination. *Knowledge and information systems* 33, 1 (2012), 1–33.

[28] Michael Kearns, Seth Neel, Aaron Roth, and Zhiwei Steven Wu. 2018. Preventing fairness gerrymandering: Auditing and learning for subgroup fairness. In *International conference on machine learning.* PMLR, 2564–2572.

[29] Diksha Khurana, Aditya Koli, Kiran Khatter, and Sukhdev Singh. 2023. Natural language processing: State of the art, current trends and challenges. *Multimedia tools and applications* 82, 3 (2023), 3713–3744.

[30] Kohavi and Ron. 1996. Census Income. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5GP7S.

[31] Matt J Kusner, Joshua Loftus, Chris Russell, and Ricardo Silva. 2017. Counterfactual fairness. *Advances in neural information processing systems* 30 (2017).

[32] Yiling Lou, Junjie Chen, Lingming Zhang, and Dan Hao. 2019. A survey on regression test-case prioritization. In *Advances in Computers.* Vol. 113. Elsevier, 1–46.

[33] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2021. A survey on bias and fairness in machine learning. *ACM Computing Surveys (CSUR)* 54, 6 (2021), 1–35.

[34] Verya Monjezi, Ashutosh Trivedi, Gang Tan, and Saeid Tizpaz-Niari. 2023. Information-theoretic testing and debugging of fairness defects in deep neural networks. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE).* IEEE, 1571–1582.

[35] S. Moro, P. Rita, and P. Cortez. 2014. Bank Marketing. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5K306.

[36] Sajjad Mozaffari, Omar Y Al-Jarrah, Mehrdad Dianati, Paul Jennings, and Alexandros Mouzakitis. 2020. Deep learning-based vehicle behavior prediction for autonomous driving applications: A review. *IEEE Transactions on Intelligent Transportation Systems* 23, 1 (2020), 33–47.

[37] Tanmayee Narendra, Anush Sankaran, Deepak Vijaykeerthy, and Senthil Mani. 2018. Explaining deep learning models using causal inference. *arXiv preprint arXiv:1811.04376* (2018).

[38] Kohei Nishino, Takashi Kitamura, Tomoji Kishi, and Cyrille Artho. 2020. Toward an Encoding Approach to Interaction-based Test Suite Minimization. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW).* IEEE, 211–212.

[39] Dana Pessach and Erez Shmueli. 2022. A review on fairness in machine learning. *ACM Computing Surveys (CSUR)* 55, 3 (2022), 1–44.

[40] Geoff Pleiss, Manish Raghavan, Felix Wu, Jon Kleinberg, and Kilian Q Weinberger. 2017. On fairness and calibration. *Advances in neural information processing systems* 30 (2017).

[41] ProPublica. 2021. *Compas software ananlysis.* https://github.com/propublica/compas-analysis

[42] Gregg Rothermel, Roland H. Untch, Chengyun Chu, and Mary Jean Harrold. 2001. Prioritizing test cases for regression testing. *IEEE Transactions on software engineering* 27, 10 (2001), 929–948.

[43] Golmei Shaheamlung, Ketusezo Rote, et al. 2020. A comprehensive review for test case prioritization in software engineering. In *2020 International Conference on Intelligent Engineering and Management (ICIEM).* IEEE, 331–336.

[44] Dima Suleiman, Marwah Alian, and Amjad Hudaib. 2017. A survey on prioritization regression testing test case. In *2017 8th International Conference on Information Technology (ICIT).* IEEE, 854–862.

[45] Bing Sun, Jun Sun, Long H Pham, and Jie Shi. 2022. Causality-based neural network repair. In *Proceedings of the 44th International Conference on Software Engineering.* 338–349.

[46] Florian Tramer, Vaggelis Atlidakis, Roxana Geambasu, Daniel Hsu, Jean-Pierre Hubaux, Mathias Humbert, Ari Juels, and Huang Lin. 2017. Fairtest: Discovering unwarranted associations in data-driven applications. In *2017 IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE, 401–416.

[47] Sakshi Udeshi, Pryanshu Arora, and Sudipta Chattopadhyay. 2018. Automated directed fairness testing. In *Proceedings of the 33rd ACM/IEEE international conference on automated software engineering.* 98–108.

[48] Jian Wang, Hengde Zhu, Shui-Hua Wang, and Yu-Dong Zhang. 2021. A review of deep learning on medical image analysis. *Mobile Networks and Applications* 26, 1 (2021), 351–380.

[49] Zhaoyu Wang, Pingchuan Ma, and Shuai Wang. 2023. Towards practical federated causal structure learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases.* Springer, 351–367.

[50] Cody Watson, Nathan Cooper, David Nader Palacio, Kevin Moran, and Denys Poshyvanyk. 2022. A systematic literature review on the use of deep learning in software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 31, 2 (2022), 1–58.

[51] Yeh and I-Cheng. 2009. Default of Credit Card Clients. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C55S3H.

[52] Shin Yoo and Mark Harman. 2012. Regression testing minimization, selection and prioritization: a survey. *Software testing, verification and reliability* 22, 2 (2012), 67–120.

[53] Junzhe Zhang and Elias Bareinboim. 2018. Fairness in decision-making—the causal explanation formula. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 32.

[54] Jie M Zhang and Mark Harman. 2021. " Ignorance and Prejudice" in Software Fairness. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 1436–1447.

[55] Lingfeng Zhang, Yueling Zhang, and Min Zhang. 2021. Efficient white-box fairness testing through gradient search. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*. 103–114.

[56] Mengdi Zhang and Jun Sun. 2022. Adaptive fairness improvement based on causality analysis. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 6–17.

[57] Peixin Zhang, Jingyi Wang, Jun Sun, Guoliang Dong, Xinyu Wang, Xingen Wang, Jin Song Dong, and Ting Dai. 2020. White-box fairness testing through adversarial sampling. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*. 949–960.

[58] Haibin Zheng, Zhiqing Chen, Tianyu Du, Xuhong Zhang, Yao Cheng, Shouling Ji, Jingyi Wang, Yue Yu, and Jinyin Chen. 2022. Neuronfair: Interpretable white-box fairness testing through biased neuron identification. In *Proceedings of the 44th International Conference on Software Engineering*. 1519–1531.