Fix Fairness, Don't Ruin Accuracy: Performance Aware Fairness Repair using AutoML

Giang Nguyen
gnguyen@iastate.edu
Dept. of Computer Science
Iowa State University
Ames, Iowa, USA

Sumon Biswas
sumonb@cs.cmu.edu
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA, USA

Hridesh Rajan
hridesh@iastate.edu
Dept. of Computer Science
Iowa State University
Ames, Iowa, USA

ABSTRACT

Machine learning (ML) is increasingly being used in critical decision-making software, but incidents have raised questions about the fairness of ML predictions. To address this issue, new tools and methods are needed to mitigate bias in ML-based software. Previous studies have proposed bias mitigation algorithms that only work in specific situations and often result in a loss of accuracy. Our proposed solution is a novel approach that utilizes automated machine learning (AutoML) techniques to mitigate bias. Our approach includes two key innovations: a novel optimization function and a fairness-aware search space. By improving the default optimization function of AutoML and incorporating fairness objectives, we are able to mitigate bias with little to no loss of accuracy. Additionally, we propose a fairness-aware search space pruning method for AutoML to reduce computational cost and repair time. Our approach, built on the state-of-the-art *Auto-Sklearn* tool, is designed to reduce bias in real-world scenarios. In order to demonstrate the effectiveness of our approach, we evaluated our approach on four fairness problems and 16 different ML models, and our results show a significant improvement over the baseline and existing bias mitigation techniques. Our approach, *Fair-AutoML*, successfully repaired 60 out of 64 buggy cases, while existing bias mitigation techniques only repaired up to 44 out of 64 cases.

CCS CONCEPTS

• **Software and its engineering** → **Search-based software engineering**; • **Computing methodologies** → **Machine learning**.

KEYWORDS

Software fairness, bias mitigation, fairness-accuracy trade-off, machine learning software, automated machine learning

ACM Reference Format:
Giang Nguyen, Sumon Biswas, and Hridesh Rajan. 2023. Fix Fairness, Don't Ruin Accuracy: Performance Aware Fairness Repair using AutoML. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '23), December 3–9, 2023, San Francisco, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3611643.3616257

ESEC/FSE '23, December 3–9, 2023, San Francisco, CA, USA

ACM ISBN 979-8-4007-0327-0/23/12.
https://doi.org/10.1145/3611643.3616257

1 INTRODUCTION

Recent advancements in machine learning have led to remarkable success in solving complex decision-making problems such as job recommendations, hiring employees, social services, and education [2, 10, 11, 21, 22, 24, 38, 39, 51, 53, 55, 56, 63]. However, ML software can exhibit discrimination due to unfairness bugs in the models [3, 6]. These bugs can result in skewed decisions towards certain groups of people based on protected attributes such as race, age, or sex [30, 31].

To address this issue, the software engineering (SE) community has invested in developing testing and verification strategies to detect unfairness in software systems [1, 8, 30, 31, 62]. Additionally, the machine learning literature contains a wealth of research on defining different fairness criteria for ML models and mitigating bias [12, 20, 26, 37, 48, 50, 64, 65]. Various bias mitigation methods have been proposed to build fairer models. Some approaches mitigate data bias by adapting the training data [15, 16, 52]; some modify ML models during the training process to mitigate bias [19, 32, 41, 58, 60], and others aim to increase fairness by changing the outcome of predictions [1, 62, 66].

Despite these efforts, current bias mitigation techniques often come at the cost of decreased accuracy [6, 42]. Their effectiveness varies based on datasets, fairness metrics, or the choice of protected attributes [18, 25, 26, 37]. Hort et al. proposed Fairea [42], a novel approach to evaluate the effectiveness of bias mitigation techniques, which found that nearly half of the evaluated cases received poor effectiveness. Moreover, evaluations by Chen *et al.* also showed that in 25% of cases, bias mitigation methods reduced both ML performance and fairness [18].

Recent works [34, 42, 60] have shown that parameter tuning can successfully fix fairness bugs without sacrificing accuracy. By finding the best set of parameters, parameter tuning can minimize the error between the predicted values and the true values to reduce bias. This helps to ensure that the model is not overly simplified or too complex, which can lead to underfitting (high bias) or overfitting (low accuracy), respectively. By tuning the parameters, we can find the right balance between bias and accuracy, which leads to a model that generalizes well to different data or fairness metric. However, it is challenging to identify which parameter setting achieves the best fairness-accuracy trade-off [34].

Recent advancements in AutoML technology [28, 29, 43] have made it possible for both experts and non-experts to harness the power of machine learning. AutoML proves to be an effective option for discovering optimal parameter settings; however, currently there is a lack of focus on reducing bias within the AutoML techniques. Thus, we pose the following research questions: *Is it possible*

to utilize AutoML for the purpose of reducing bias? Is AutoML effective in mitigating bias? Does AutoML outperform existing bias reduction methods? Is AutoML more adaptable than existing bias mitigation techniques?

We introduce *Fair-AutoML*, a novel technique that utilizes AutoML to fix fairness bugs in machine learning models. Unlike existing bias mitigation techniques, *Fair-AutoML* addresses their limitations by enabling efficient and fairness-aware Bayesian search to repair unfair models, making it effective for a wide range of datasets, models, and fairness metrics. The key idea behind *Fair-AutoML* is to use AutoML to explore as many configurations as possible in order to find the optimal fix for a buggy model. Particularly, *Fair-AutoML* enhances the potential of AutoML for fixing fairness bugs in two novel techniques: by generating a new optimization function that guides AutoML to fix fairness bugs without sacrificing accuracy, and by defining a new search space based on the specific input to accelerate the bug-fixing process. Together, these contributions enable *Fair-AutoML* to effectively fix fairness bugs across various datasets and fairness metrics. We have implemented *Fair-AutoML* on top of *Auto-Sklearn* [29], the state-of-the-art AutoML framework.

Fair-AutoML aims to effectively address the limitations of existing bias mitigation techniques by utilizing AutoML to efficiently repair unfair models across various datasets, models, and fairness metrics. We conduct an extensive evaluation of *Fair-AutoML* using 4 widely used datasets in the fairness literature [1, 31, 62] and 16 buggy models collected from a recent study [6]. The results demonstrate the effectiveness of our approach, as *Fair-AutoML* successfully repairs 60 out of 64 buggy cases, surpassing the performance of existing bias mitigation techniques which were only able to fix up to 44 out of 64 bugs in the same settings and training time.

Our main contributions are the following:

- We have proposed a novel approach to fix unfairness bugs and retain accuracy at the same time.
- We have proposed methods to generate the optimization function automatically based on an input to make AutoML fixing fairness bugs more efficiently.
- We have pruned the search space automatically based on an input to fix fairness bugs faster using AutoML.
- We have implemented our approach in a SOTA AutoML, *Auto-Sklearn* [29]. **The artifact is available here [33]**.

The paper is organized as follows: §2 describes the background, §3 presents a motivation, §4 indicates the problem definition, §5 shows the *Fair-AutoML* approaches, §6 presents the our evaluation, §7 discusses the limitations and future directions of *Fair-AutoML*, §8 discusses the threats to validity of *Fair-AutoML*, §9 concludes, and §10 describes the artifact.

2 BACKGROUND

We begin by providing an overview of the background and related research in the field of software fairness.

2.1 Preliminaries

2.1.1 ML Software. Given an input dataset D split into a training dataset D_{train} and a validation dataset D_{val}, a ML software system can be abstractly viewing as mapping problem $M_{\lambda,c} : x \rightarrow y$ from inputs x to outputs y by learning from D_{train}. ML developers aims to search for a hyperparameter configuration $\lambda*$ and complementary components $c*$ for model M to obtain optimal fairness-accuracy on D_{val}. The complementary components can be ML algorithms combined with a classifier i.e., pre-processing algorithms.

2.1.2 AutoML. Given the search spaces Λ and C for hyperparameters and complementary components, AutoML aims to find $\lambda*$ and $c*$ to obtain the lowest value of the cost function (Equation 1):

$$M = \underset{\lambda \in \Lambda, c \in C}{\arg\min} Cost(M_{\lambda*,c*}, D_{val}) \quad (1)$$

$$(\lambda*, c*) = \underset{(\lambda,c)}{\arg\min} Loss(M_{\lambda,c}, D_{train}) \quad (2)$$

2.1.3 Measures. We consider a problem, where each individual in the population has a true label in $y = \{0, 1\}$. We assume a protected attribute $z = \{0, 1\}$, such as race, sex, age, where one label is privileged (denoted 0) and the other is unprivileged (denoted 1). The predictions are $\hat{y} \in \{0, 1\}$ that need to be not only accurate with respect to y but also fair with respect to the protected attribute z.

Accuracy Measure. Accuracy is given by the ratio of the number of correct predictions by the total number of predictions.

Accuracy = (# True positive + # True negative) / # Total

Fairness Measure. We use four ways to define group fairness metrics, which are widely used in fairness literature [4, 5, 30]:
The *Disparate Impact (DI)* is the proportion of the unprivileged group with the favorable label divided by the proportion of the privileged group with the favorable label [26, 64].

$$DI = \frac{Pr[\hat{y}=1|z=0]}{Pr[\hat{y}=1|z=1]}$$

The *Statistical Parity Difference (SPD)* quantifies the disparity between the favorable label's probability for the unprivileged group and the favorable label's probability for the privileged group [12].

$$SPD = Pr[\hat{y} = 1|z = 0] - Pr[\hat{y} = 1|z = 1]$$

The *Equal Opportunity Difference* (EOD) measures the disparity between the true-positive rate of the unprivileged group and the privileged group.

$$TPR_u = Pr[\hat{y} = 1|y = 1, z = 0]; TPR_p = Pr[\hat{y} = 1|y = 1, z = 1]$$
$$EOD = TPR_u - TPR_p$$

The Average Absolute Odds Difference (AOD) is the mean of the difference of true-positive rate and false-positive rate among the unprivileged group and privileged group [37].

$$FPR_u = P[\hat{y} = 1|y = 0, z = 0]; FPR_p = P[\hat{y} = 1|y = 0, z = 1]$$
$$AOD = \frac{1}{2} * |FPRu - FPRp| + |TPR_u - TPR_p|$$

To use all the metrics in the same setting, DI has been plotted in the absolute value of the log scale, and SPD, EOD, AOD have been plotted in absolute value [16, 42]. Thus, the bias score of a model is measured from 0, with lower scores indicating more fairness.

2.2 Related Work

2.2.1 Bias mitigation. SE and ML researchers has developed various bias mitigation methods to increase fairness in ML software divided into three categories [30, 40]:

Pre-processing approaches reduce bias by pre-processing the training data. For instance, *Fair-SMOTE* [15] addresses data bias by removing biased labels and balancing the distribution of positive

and negative examples for each sensitive attribute. *Reweighing* [48] decreases bias by assigning different weights to different groups based on the degree of favoritism of a group. *Disparate Impact Remover* [26] is a pre-processing bias mitigation technique that aims to reduce bias by editing feature values.

In-processing approaches reduce bias by modifying ML models during the training process i.e., *Parfait-ML* [60] present a search-based solution to balance fairness and accuracy by tuning hyperparameter to approximate the twined Pareto curves. *MAAT* [19] is an ensemble approach aimed at improving the fairness-performance trade-off in ML software. Instead of combining models with the same learning objectives as traditional ensemble methods, *MAAT* merges models that are optimized for different goals.

Post-processing approaches change the outcome of prediction to reduce bias. This technique unfavors privileged groups' instances and favors those of unprivileged groups lying around the decision boundary. For example, *Equalized Odds* [37] reduces the value of EOD by modifying the output labels. *Fax-AI* [35] eliminates direct discrimination in machine learning models by limiting the use of certain features, thereby preventing them from serving as surrogates for protected attributes. *Reject Option Classification* [49] prioritizes instances from the privileged group over those from the unprivileged group that are situated on the decision boundary with high uncertainty.

Previous efforts have made significant progress in reducing bias; however, they come at the cost of decreased accuracy and their results can vary depending on the datasets and fairness metrics. Our proposal, *Fair-AutoML*, aims to strike a balance between accuracy and bias reduction and demonstrate generalizability across various datasets and metrics.

2.2.2 Search space pruning. Search space pruning involves reducing the size or complexity of the search space in optimization or machine learning tasks. Pruning techniques are employed to accelerate the optimization process of AutoML by eliminating unpromising or redundant options, thus focusing computational resources on more promising areas of the search space. For example, Feurer *et al.* [29] introduce Auto-Sklearn 2.0, a novel approach aimed at enhancing the performance of Auto-Sklearn. This advancement involves constraining the search space to exclusively comprise iterative algorithms, while eliminating feature preprocessing. This strategic adjustment streamlines the implementation of successive halving, as it reduces the complexity to a single fidelity type: the number of iterations. Otherwise, the incorporation of dataset subsets as an alternative fidelity would require additional consideration. Another innovative contribution comes from Cambronero *et al.*, who introduces AMS [13]. This method capitalizes on the wealth of source code repositories to streamline the search space for AutoML. Notably, AMS harnesses the power of unspecified complementary and functionally related API components. By leveraging these components, the search space for AutoML is pruned effectively. Diverging from prior research efforts, *Fair-AutoML* distinguishes itself by leveraging data characteristics to effectively trim down the search space. Notably, existing techniques in search space pruning primarily target accuracy enhancement within AutoML. In contrast, our innovative pruning methodology within *Fair-AutoML* is uniquely directed towards repairing unfair models.

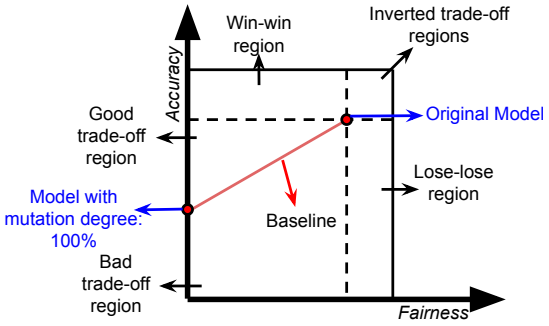

Figure 1: Baseline fairness-accuracy trade-off [42]

2.2.3 AutoML extension. AutoML aims to automate the process of building a high-performing ML model, but it has limitations. It can be costly, time-consuming to train, and produces complex models that are difficult to understand. To address these limitations, software engineering researchers have developed methods to enhance AutoML performance, such as *AMS* [13] and *Manas* [54]. *AMS* utilizes source code repositories to create a new search space for AutoML, while *Manas* mines hand-developed models to find a better starting point for AutoML. The goal of these methods is to improve AutoML to maximize the accuracy. Different from these methods, *Fair-AutoML*, built on top of *Auto-Sklearn* [29], is the first to focus on repairing unfair models.

3 MOTIVATION

The widespread use of machine learning in software development has brought attention to the issue of fairness in ML models. Although various bias mitigation techniques have been developed to address this issue, they have limitations. These techniques suffer from a poor balance between fairness and accuracy [42], and are not applicable to a wide range of datasets, metrics, and models [25, 26, 37]. To gain a deeper understanding of these limitations, we evaluate six different bias mitigation techniques using four fairness metrics, four datasets, and six model types. The evaluation criteria are borrowed from Fairea [42] and are presented in Table 1.

Fairea is designed to assess the trade-off between fairness and accuracy of bias mitigation techniques. The methodology of *Fairea* is demonstrated in Figure 1, where the fairness and accuracy of a bias mitigation technique on a dataset are displayed in a two-dimensional coordinate system. The baseline is established by connecting the fairness-accuracy points of the original model and the mitigation models on the dataset. *Fairea* evaluates the performance of the mitigation technique by altering the original model predictions and replacing a random subset of the predictions with other labels. The mutation degree ranges from 10% to 100% with a step-size of 10%. The baseline classifies the fairness-accuracy trade-off of a bias mitigation technique into five regions: lose-lose trade-off (**lose**), bad trade-off (**bad**), inverted trade-off (**inv**), good trade-off (**good**), and win-win trade-off (**win**). A technique reducing both accuracy and fairness would fall into the lose-lose trade-off region. If the trade-off is worse than the baseline, it would fall into the bad trade-off region. If the trade-off is better than the baseline, it would fall into the good trade-off region. If a bias mitigation method simultaneously decreases both bias and accuracy, it would fall into the

Table 1: Mean proportions of mitigation cases that that fall into each mitigation region

Criteria		Lose	Bad	Inv	Good	Win
Metric	DI	7%	31%	5%	43%	14%
	SPD	4%	36%	6%	40%	14%
	EOD	23%	15%	14%	40%	8%
	AOD	9%	30%	5%	40%	16%
Dataset	Adult [44]	18%	6%	14%	55%	7%
	Bank [45]	9%	44%	7%	23%	17%
	German [46]	6%	36%	2%	46%	10%
	Titanic [47]	11%	26%	3%	43%	17%
Mean		**11%**	**28%**	**7%**	**41%**	**13%**

Bad: bad trade-off region, Lose: lose-lose trade-off region, Inv: inverted trade-off region, Good: good trade-off region, Win: win-win trade-off region.

inverted trade-off region. If the technique improves both accuracy and fairness, it would fall into the win-win trade-off region.

The results of the region classification of six bias mitigation techniques - *Reweighing* [48], *Disparate Impact Remover* [26], *Parfait-ML* [60], *Equalized Odds* [37], *FaX-AI* [35], *Reject Option Classification* [49] - are shown in Table 1. The evaluation was conducted on 64 buggy cases using different criteria such as fairness metrics and datasets. The case is identified as buggy when it falls below the *Fairea* baseline. The mean percentage of each technique falling into the corresponding regions is listed in each cell. The mean results provide a general overview of the current state of bias mitigation techniques. Further details on the performance of each individual bias mitigation technique can be found in Table 3 of our evaluation.

Table 1 illustrates that the majority of existing bias mitigation techniques have a poor fairness-accuracy trade-off across different datasets, fairness metrics, and classification models. Specifically, 39% of the cases show that these techniques perform worse than the original model, with 28% of the cases resulting in a poor trade-off and 11% resulting in a decrease in accuracy and an increase in bias. Additionally, Table 1 shows that the performance of these techniques varies depending on the input, as demonstrated by the different results obtained when using different datasets or fainess metrics [25, 26, 37]. For example, the bias mitigation techniques had a high performance in 62% of the cases using the Adult dataset (55% for good trade-off region and 7% for win-win trade-off region), but only achieved 40% good effectiveness in the Bank dataset.

Hort *et al.* [42] have demonstrated that through proper parameter tuning, it is possible to address fairness issues in machine learning models without sacrificing accuracy. However, determining the optimal fairness-accuracy trade-off can be a challenge. Although AutoML can be effective in finding the best parameter settings, it does not specifically address bias reduction. This motivates the development of *Fair-AutoML*, a novel approach that utilizes Bayesian optimization to tune parameters and address fairness issues without hindering accuracy. *Fair-AutoML* is evaluated for its generality across different fairness metrics and datasets, and unlike other bias mitigation methods, it can be applied to any dataset or metric.

This work focuses on improving fairness quantitatively of buggy models instead of targeting a specific type of datasets and models. Our method is general since we utilize the power of AutoML to try as many configurations as possible to obtain the optimal fix; therefore, our method can work on various types of datasets and metrics. The rest of this work describes our approach, *Fair-AutoML*, that addresses the limitations of both existing bias mitigation methods and AutoML. As a demonstration, *Fair-AutoML* achieved good performance in 100% of the 16 buggy cases in the Adult dataset, while 75% of the mitigation cases showed a good fairness-accuracy trade-off, and the remaining 25% exhibited an improvement in accuracy without sacrificing bias reduction.

4 PROBLEM DEFINITION

This work aims to utilize AutoML to address issues of unfairness in ML software by finding a new set of configurations for the model that achieves optimal fairness-accuracy trade-off. Because fairness is an additional consideration beyond accuracy, the problem becomes a multi-objective optimization problem, requiring a new cost function that can optimize both fairness and accuracy simultaneously. To achieve this, we use a technique called weighted-sum scalarization (Equation 3) [23], which allows us to weigh the importance of different objectives and create a single scalar cost function.

$$A = \sum_{i=1}^{n} c_i * \beta_i \tag{3}$$

where, β_i denotes the relative weight of importance of c_i:

$$\sum_{i=1}^{n} \beta_i = 1 \tag{4}$$

In this work, we use a cost function (or objective function) that is a weighted-sum scalarization of two decision criteria: bias and accuracy. This cost function, as shown in Equation 5, assign weights to bias and accuracy in the cost function allow us to adjust the trade-off between the two criteria according to the specific problems:

$$Cost(M_{\lambda,c}, \mathcal{D}(z)) = \beta * f + (1-\beta) * (1-a) \tag{5}$$

We analyze the output of the buggy ML software (including bias and accuracy) to create a suitable cost function for each input. By analyzing the output, we are able to automatically estimate the weights of the cost function in order to balance fairness and accuracy for a specific problem. To the best of our knowledge, this is the first work that applies output analysis of the software to AutoML to repair unfair ML models.

However, using AutoML can be costly and time-consuming. To address this issue, we propose a novel method that automatically create new search spaces $\Lambda*$ and $C*$ based on different inputs to accelerate the bug-fixing process of AutoML. These new search spaces are smaller in size compared to the original ones, $|\Lambda*| < |\Lambda|$ and $|C*| < |C|$. Particularly, as shown in Equation 6, *Fair-AutoML* takes as input a ML model and a dataset with a protected attribute z, and aims to find $\lambda*$ and $c*$ in the smaller search space, in order to minimize the cost value.

$$M = \underset{\lambda* \in \Lambda*, c* \in C*}{\arg\min} \; Cost(M_{\lambda*,c*}, D_{val}(z)) \tag{6}$$

The technique of search space pruning in *Fair-AutoML* utilizes data characteristics to enhance bug-fixing efficiency. By shrinking the search spaces based on input analysis, *Fair-AutoML* can find better solutions more quickly. A set of predefined modifications to the ML model are pre-built and used as a new search space for new input datasets, reducing the time needed to fix buggy models. Our approach is based on previous works in AutoML [29], but updated and modified to tackle bias issues. To the best of our knowledge,

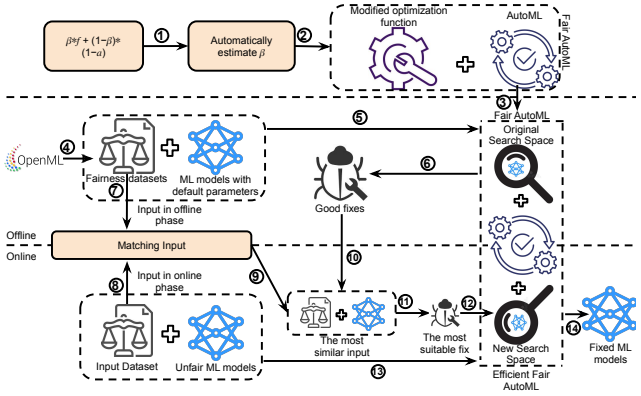

Figure 2: An Overview of *Fair-AutoML* Approach

we are the first to propose a search space pruning technique for fairness-aware AutoML.

5 FAIR-AUTOML

This section describes a detailed description of key components of *Fair-AutoML* (Figure 2): the dynamic optimization function (steps 1-3) and the search space pruning (steps 4-13).

5.1 Dynamic Optimization for Bias Elimination

We strive to eliminate bias in unfair models by utilizing Equation 5 as the objective function and determining the optimal value of β to minimize the cost function. In this section, we propose an approach to automatically estimate the optimal value of β for a specific dataset and a targeted model. This method ensures efficient correction of fairness issues while maintaining high predictive accuracy.

5.1.1 Upper bound of the cost function. To estimate the optimal value of β, the first step is to determine the upper bound of the cost function. This can be done by using a "pseudo-model", which is the 100% mutation degree model [42], as shown in the Figure 1. In other words, the pseudo-model always achieves the accuracy on any binary classification problem as follows:

$$a_0 = max(P(Y=1), P(Y=0)) \tag{7}$$

Given an input, the pseudo-model achieves an accuracy of a_0 and a bias value of f_0 on that input. We define the cost function, $Cost$, of the buggy ML model with accuracy a and bias value f on the input. As AutoML tries different hyperparameter configurations to fix the model, the values of a and f may change over time. The upper bound of the cost function is defined as Equations 8 and 9:

$$Cost(M_{\lambda,c}, \mathcal{D}(z)) < \beta * f_0 + (1-\beta) * (1-a_0) \tag{8}$$

$$\Leftrightarrow \beta * f + (1-\beta) * (1-a) < \beta * f_0 + (1-\beta) * (1-a_0) \tag{9}$$

The upper bound of the cost function is defined with the goal of repairing a buggy model so that its performance falls within a good/win-win trade-off region of fairness and accuracy. In other words, the accuracy of the repaired model must be higher than the accuracy of the pseudo-model. The repaired model must be better than the pseudo-model in terms of the cost function's value. Since the pseudo-model has zero bias ($f_0 = 0$), the upper bound of the cost function is defined as follows (Equation 10):

$$\beta * f + (1-\beta) * (1-a) < (1-\beta) * (1-a_0) \tag{10}$$

5.1.2 Lower bound of β. In this work, we desire to optimize the value of β in order to minimize bias as much as possible. The cost function used by *Fair-AutoML* is designed to balance accuracy and fairness, and increasing β will place more emphasis on reducing bias. However, simply setting β to its highest possible value is not a viable option, as it may lead to low predictive accuracy and overfitting. We cannot accept models with poor predictive accuracy regardless of their low bias [36, 57]. To overcome this challenge, we aim to find the lower bound of β, which can be done based on the upper bound of the cost function. From Equation 10, we get:

$$\beta < \frac{a - a_0}{a - a_0 + f} \tag{11}$$

However, if the value of β is smaller than $\frac{a-a_0}{a-a_0+f}$, the optimization function $Cost$ will always meet its upper bound condition. If the value of β always satisfies the upper bound condition of the cost function regardless of accuracy and fairness, we can obtain a better optimization function by either increasing accuracy or decreasing bias. In this case, we cannot guide AutoML to produce a lower bias. Therefore, to guide AutoML produces an output with improved fairness, we set a lower bound for β as Equation 12:

$$\beta \geq \frac{a - a_0}{a - a_0 + f} \tag{12}$$

The intuition being that our method aims to increase the chance for AutoML to achieve better fairness. However, by setting $\beta < \frac{a-a_0}{a-a_0+f}$ and $a > a_0$ (we aim to find a model which has better accuracy than the pseudo-model), any value of bias (f) can satisfy upper bound condition of the cost function, which lower chance to obtain fairer models of AutoML. To increase this chance, we set $\beta \geq \frac{a-a_0}{a-a_0+f}$ and $a > a_0$. In this case, AutoML need to find better models that has lower bias to satisfy Equation 10. In other words, this lower bound condition indirectly forces bayesian optimization to search for lower bias models.

5.1.3 β estimation. The final step is estimating the value of β based on its lower bound condition. Suppose that the buggy model achieves an accuracy of a_1 and a bias value of f_1 on that input. From the begining, we have: $a = a_1$ and $f = f_1$. In that time, the lower bound of β is $L = \frac{a_1-a_0}{a_1-a_0+f_1}$, so we have:

$$\beta = L + k, k \in [0, 1-L] \tag{13}$$

We present a greedy algorithm for estimating the value of β, which is detailed in Algorithm 1. Given a dataset D with a protected attribute z and a buggy model M (Line 1), we start by measuring the lower bound of β. Next, we run *Fair-AutoML* on the input under time constraint t with a value of β set to $\frac{a_1-a_0}{a_1-a_0+f_1}$ (Line 2-8). As the algorithm searches, whenever *Fair-AutoML* finds a candidate model that meets the condition $Cost < Cost_0$ (Lines 10-12), the value of β is slightly increased by α (Line 10-12). If after N tries, *Fair-AutoML* cannot find a model that satisfies the condition, the final value of β is set to $\beta = \beta - \alpha$ for the remaining search time to prevent overfitting from an excessively high value of β (Lines 13-15). The algorithm returns the best model found (Line 16).

Algorithm 1 Greedy Weight Identifier

Input: a dataset D with protected attribute z, buggy model M hyperparametered by λ, the increment value α, the searching time t and the threshold N
$\beta = \frac{a_1 - a_0}{a_1 - a_0 + f_1}$
$Cost(M_{\lambda,c}, \mathcal{D}(z)) = \beta * f + (1 - \beta) * (1 - a)$
$Cost_0(M_{\lambda,c}, \mathcal{D}(z)) = (1 - \beta) * (1 - a_0)$
count = 0
checker = False
while t **do**
    $M_{\lambda*,c*} = \arg\min_{\lambda \in \Lambda} Cost(M_{\lambda,c}, \mathcal{D}(z))$
    count = count + 1
    if $Cost(M_{\lambda,c}, \mathcal{D}(z)) < Cost_0(M_{\lambda,c}, \mathcal{D}(z))$ **then**
        if checker = False **then**
            $\beta = \beta + \alpha$
            count = 0
    if $count \geq N$ and checker = False **then**
        β = β - α
        checker = True
return $M_{\lambda}*$

5.2 Search Space Pruning for Efficient Bias Elimination

We propose a solution to speed up the Bayesian optimization process in *Fair-AutoML* by implementing search space pruning. This technique takes advantage of data characteristics to automatically reduce the size of the search space in AutoML, thus improving its efficiency. Our approach includes two phases: the offline phase and the online phase. The offline phase trains a set of inputs multiple times to gather a collection of hyperparameters and complementary components for each input, forming a pre-built search space. In the online phase, when a new input is encountered, it is matched against the inputs stored in our database to find a matching pre-built search space, which is then utilized to repair the buggy model. This approach effectively replaces the original search space of *Fair-AutoML*, making the Bayesian optimization process much faster. Search space pruning has already been successfully applied before [13, 28]; however, this is the first application of data characteristics to prune the search space for fairness-aware AutoML.

5.2.1 Offline Phase. This phase constructs a set of search spaces for *Fair-AutoML* based on different inputs. It is important to note that the input format in the offline phase must match that of the online phase, which includes a dataset with a protected attribute and a ML model. This ensures that the pre-built search spaces created in the offline phase can be effectively utilized in the online phase.

Input. In the offline phase, we collect a set of inputs to build search spaces for *Fair-AutoML*. The inputs are obtained as follows. Firstly, we mine machine learning datasets from *OpenML*, considering only the 3425 active datasets that have been verified to work properly. Secondly, to ensure that the mined datasets are relevant to the fairness problem, we only collect datasets that contain at least one of the following attributes: *age*, *sex*, *race* [17]. In total, we collected 231 fairness datasets. Thirdly, for each mined dataset, we use all available protected attributes. For example, when dealing with datasets that contain multiple protected attributes, such as the *Adult* dataset that includes *sex* and *race* as protected attributes, we treat them as distinct inputs for the dataset. Finally, we use the default values for the hyperparameters of the input ML model in the offline phase, as we do not know the specific values that will be used in the online phase.

Algorithm 2 Database Building

1: **Input:** a dataset D with protected attribute z, a model M with default hyperparameters λ. Running time t.
2: d = $\emptyset$
3: dev = 1
4: database = {}
5: space = {}
6: count = 0
7: **while** count ≤ n **do**
8: count = count + 1
9: **while** t **do**
10: $M_{\lambda}* = \arg\min Cost(M_{\lambda}, \mathcal{D}(z))$
11: $d = d \cup M_{\lambda}*$
12: kBestPipelines = top_k(d)
13: mBestComponents = top_m(kBestPipelines)
14: **for** model ∈ kBestPipelines **do**
15: **for** para ∈ model **do**
16: space[para] = space[para]∪[para.val]
17: **for** para ∈ space **do**
18: **if** para is numerical **then**
19: no_outliers = $\emptyset$
20: **for** i ∈ space[para] **do**
21: **if** $|i - \overline{space[para]}| < dev * \sigma(space[para])$ **then**
22: no_outliers = no_outliers ∪ space[para][i]
23: space[para] = [min(no_outliers), max(no_outliers)]
24: database[input] = (space, mBestComponents)
25: **return** database

Database building. To build a pre-defined search space database, we use the algorithm outlined in Algorithm 2 to obtain a pre-built search space for each collected input in order to fix the buggy model. This process involves training a fairness dataset with a specific protected attribute and ML model multiple times using *Fair-AutoML*, collecting the top k best pipelines found, and extracting parameters from these pipelines. In particular, we use *Fair-AutoML* to train the fairness dataset with a specific protected attribute and a ML model for n iterations (Line 7-11). We then gather the top k best pipelines, including a classifier and complementary components, found by *Fair-AutoML* according to the optimization function's value (Line 12). This results in $k * n$ total pipelines. From these pipelines, we extract and store the m most frequently used complementary components in the database (Line 13). For each classifier parameter, we also store its value (Lines 14-16). This results in k∗n values being stored for each hyperparameter. If a hyperparameter is categorical and its values are sampled from a set of different values, we store all its unique values in the database. If a hyperparameter is numerical and its values are sampled from a uniform distribution, we remove any outliers and store the range of values from the minimum to the maximum in the database (Lines 17-23). After this process, we have collected the pre-built search space for the input (Lines 24-25). We believe that two similar inputs may have similar buggy models and

Algorithm 3 Input Matching

1: **Input:** a input dataset D with the protected attribute z, the number of data points p, the number of features f, lower bound L, a buggy model M, and a database.
2: dist = {}
3: **for** d_i in database **do**
4: dist[d_i] = $|f_i - f| + |p_i - p|$
5: similarDataset = min(dist, key=dist.get)
6: dist = {}
7: **for** z_i in similarDataset **do**
8: dist[d_i] = $|L_i - L|$
9: similarAttribute = min(dist, key=dist.get)
10: similarModel = M with default parameter
11: **return** similarDataset, similarAttribute, similarModel

fixes, so the pre-built search space is built based on the best models found by *Fair-AutoML* from similar inputs, making it a reliable solution for fixing buggy models.

5.2.2 Online Phase. This phase utilizes a pre-built search space from the database to fix a buggy model for a given dataset by replacing the original search space with the pre-built one.

Search space pruning. Our approach of search space pruning in *Fair-AutoML* improves the bug fixing performance by reducing the size of the hyperparameter tuning space. Algorithm 3 is used to match the input dataset, protected attribute, and ML model to the most similar input in the database. Firstly, data characteristics such as the number of data points and features are used to match the new dataset with the most similar one in the database [28]. L1 distance is computed between the new dataset and each mined dataset in the space of data characteristics to determine the closest match. We consider that the most similar dataset to the new dataset is the nearest one (Line 2-5). Secondly, we compute the lower bound $L = \frac{a_1 - a_0}{a_1 - a_0 + f_1}$ of β of the new input. We then estimate the lower bound of β of all the protected attributes of the matched dataset and select the attribute whose lower bound is closest to L (Line 6-9). Lastly, two similar inputs must use the same ML algorithm (Line 10). The matching process is carried out in the order of dataset matching, protected attribute matching, and ML algorithm matching. The pre-built search space of the similar input is then used as the new search space for the new input.

6 EVALUATION

In this section, we describe the design of the experiments to evaluate the efficient of *Fair-AutoML*. We first pose research questions and discuss the experimental details. Then, we answer research questions regarding the efficiency and adaptability of *Fair-AutoML*.

RQ1: Is *Fair-AutoML* effective in fixing fairness bugs? To answer this question, we quantify the number of fairness bugs that *Fair-AutoML* is able to repair compared to existing methods, allowing us to assess the capability of an AutoML system in fixing fairness issues.

RQ2: Is *Fair-AutoML* more adaptable than existing bias mitigation techniques? The adaptability of a bias mitigation technique indicates its performance across a diverse range of datasets/metrics. So, we analyze the effectiveness of *Fair-AutoML* and existing bias mitigation techniques on different dataset/metrics to assess the adaptability of an AutoML system on fix fairness bugs.

RQ3: Are dynamic optimization function and search space pruning effective in fixing fairness bugs? To answer this question, we assess the performance of *Auto-Sklearn*, both with and without the dynamic optimization function and search space pruning, to demonstrate the impact of each proposed approach.

6.1 Experiment

6.1.1 Benchmarks. We evaluated our method using real-world fairness bugs sourced from a recent empirical study [6], with our benchmark consisting of 16 models collected from Kaggle covering five distinct types: XGBoost (**XGB**), Random Forest (**RF**), Logistic Regression (**LRG**), Gradient Boosting (**GBC**), Support Vector Machine (**SVC**). We use four popular datasets for our evaluation [10, 61, 62]:

The **Adult Census** (race) [44] comprised of 32,561 observations and 12 features that capture the financial information of individuals from the 1994 U.S. census. The objective is to predict whether an individual earns an annual income greater than 50K.

The **Bank Marketing** (age) [45] has 41,188 data points with 20 features including information on direct marketing campaigns of a Portuguese banking institution. The classification task aims to identify whether the client will subscribe to a term deposit.

The **German Credit** (sex) [46] has 1000 observations with 21 features containing credit information to predict good or bad credit.

The **Titanic** (sex) [47] has 891 data points with 10 features containing individual information of Titanic passengers. The dataset is used to predict who survived the Titanic shipwreck.

6.1.2 Evaluated Learning Techniques. We examined the performance of *Fair-AutoML* and other supervised learning methods addressing discrimination in binary classification including all three types of bias mitigation techniques and Auto-ML techniques.

Bias mitigation methods. We investigate all three types of bias mitigation methods: pre-processing, in-processing, post-processing. We select widely-studied bias mitigation methods for each category:

- The **pre-processing** includes *Reweighing* (**R**) [48], *Disparate Impact Remover* (**DIR**) [26].
- The **in-processing** includes *Parfait-ML* (**PML**) [60].
- The **post-processing** includes *Equalized Odds* (**EO**) [37], *FaX-AI* (**FAX**) [35], *Reject Option Classification* (**ROC**) [49].

Auto-Sklearn. We explore the efficiency of *Auto-Sklearn* (**AS**) [29] on mitigating bias in unfair model. Although, *Auto-Sklearn* does not seek to decrease bias, we compare its performance with *Fair-AutoML* to demonstrate the efficient of our techniques in guiding Auto-ML to repair fairness bugs.

Fair-AutoML. We create 4 versions of *Fair-AutoML* in this evaluation representing for *Fair-AutoML* with different cost functions:

- **T1** uses $\beta * DI + (1 - \beta) * (1 - accuracy)$ as a cost function.
- **T2** uses $\beta * SPD + (1 - \beta) * (1 - accuracy)$ as a cost function.
- **T3** uses $\beta * EOD + (1 - \beta) * (1 - accuracy)$ as a cost function.
- **T4** uses $\beta * AOD + (1 - \beta) * (1 - accuracy)$ as a cost function.

6.1.3 Experimental Configuration. Experiments were conducted using Python 3.6 on Intel Skylake 6140 processors. *Fair-AutoML* leverages the capabilities of *Auto-Sklearn* [29], taking advantage of

Table 2: Trade-off assessment results of *Fair-AutoML*, *Auto-Sklearn*, and mitigation techniques

	Model	Metric	T1	T2	T3	T4	AS	R	DIR	PML	EO	FAX	ROC	Model	Metric	T1	T2	T3	T4	AS	R	DIR	PML	EO	FAX	ROC
Adult Census	RF	Acc	0.010	0.001	0.005	-0.003	0.016	0.009	0.004	0.008	-0.006	0.002	-0.047	LRG	Acc	-0.013	-0.022	-0.009	-0.032	-0.001	-0.056	-0.055	-0.054	-0.057	-0.057	-0.061
		DI	**0.096**	**0.011**	**0.118**	0.095	**0.058**	0.337	inv	inv	0.292	**0.000**	0.445		DI	0.190	0.410	0.212	0.179	0.040	0.569	0.326	lose	0.375	bad	0.318
		SPD	**0.023**	**0.024**	**0.038**	0.048	**0.016**	0.055	inv	inv	0.047	inv	0.055		SPD	0.054	0.075	0.029	0.059	0.005	0.097	0.079	bad	0.083	0.076	0.072
		EOD	0.019	inv	**0.014**	0.020	**0.008**	inv	inv	inv	0.041	inv	lose		EOD	0.048	0.020	0.044	0.049	lose	0.057	0.056	0.060	0.067	0.052	0.059
		AOD	**0.028**	inv	0.030	0.035	**0.021**	**0.005**	**0.001**	inv	0.051	**0.007**	bad		AOD	0.085	0.075	0.079	0.089	0.039	0.096	0.094	0.091	0.103	0.094	0.095
	XGB	Acc	-0.019	-0.052	-0.017	-0.015	-0.001	-0.001	-0.018	-0.005	-0.035	0.001	-0.056	GBC	Acc	-0.023	-0.046	-0.014	-0.034	0.002	-0.006	-0.011	-0.034	-0.018	-0.001	-0.059
		DI	0.183	0.148	0.143	0.156	0.004	0.378	lose	lose	0.330	inv	0.456		DI	0.124	0.188	0.104	0.147	inv	0.387	0.027	lose	0.227	0.076	0.438
		SPD	0.028	0.074	0.023	0.030	0.003	0.058	lose	lose	0.051	inv	0.054		SPD	bad	0.055	0.012	0.029	inv	0.058	bad	lose	0.035	0.011	0.047
		EOD	0.036	0.041	0.037	0.030	lose	lose	lose	0.003	0.044	inv	lose		EOD	bad	0.025	0.013	0.024	inv	lose	lose	lose	0.037	0.010	lose
		AOD	0.055	0.064	0.047	0.053	0.017	0.009	lose	0.010	0.066	0.020	bad		AOD	0.037	0.055	0.041	0.050	0.018	0.031	0.031	lose	0.063	0.041	bad
Bank Marketing	RF	Acc	-0.012	-0.023	-0.001	-0.007	0.000	-0.008	-0.014	-0.008	-0.034	0.001	-0.082	XGB2	Acc	-0.001	-0.008	0.000	0.002	0.000	0.001	-0.080	-0.005	-0.075	0.000	-0.143
		DI	0.103	0.224	0.031	0.038	0.000	0.210	lose	bad	bad	0.129	bad		DI	0.158	0.236	**0.166**	**0.097**	lose	0.312	bad	bad	bad	**0.016**	bad
		SPD	0.035	bad	0.007	0.027	lose	0.065	bad	0.031	bad	0.021	bad		SPD	0.032	0.051	**0.028**	**0.021**	lose	0.053	bad	bad	bad	**0.002**	bad
		EOD	lose	bad	lose	lose	lose	lose	lose	0.029	bad	0.001	bad		EOD	lose	lose	**0.003**	0.012	lose	inv	inv	0.054	bad	inv	inv
		AOD	0.033	0.039	0.032	0.032	0.016	bad	0.020	0.046	bad	0.031	bad		AOD	0.023	0.018	**0.026**	0.027	lose	inv	inv	0.040	bad	**0.014**	bad
	XGB1	Acc	-0.002	0.002	0.007	0.000	-0.001	-0.007	-0.019	-0.005	-0.058	0.003	-0.018	GBC	Acc	-0.003	-0.007	0.014	0.011	-0.002	0.000	-0.066	-0.034	-0.058	0.001	-0.104
		DI	0.098	0.348	**0.114**	**0.092**	lose	0.222	0.174	bad	bad	**0.067**	0.183		DI	0.010	0.069	**0.011**	**0.014**	lose	0.332	bad	bad	bad	inv	bad
		SPD	0.023	0.062	**0.023**	**0.020**	lose	0.042	bad	0.029	bad	**0.013**	bad		SPD	lose	0.028	inv	inv	lose	0.052	bad	bad	bad	inv	bad
		EOD	0.021	inv	**0.011**	0.018	lose	lose	lose	0.064	bad	inv	lose		EOD	lose	lose	inv	inv	lose	inv	lose	bad	bad	inv	inv
		AOD	0.043	**0.040**	**0.043**	0.046	lose	lose	0.038	0.054	bad	**0.020**	0.043		AOD	0.014	0.023	0.027	**0.025**	lose	**0.017**	lose	bad	bad	**0.012**	bad
German Credit	RF	Acc	-0.020	-0.027	-0.012	-0.012	-0.011	-0.016	-0.007	-0.016	-0.529	-0.004	-0.446	SVC	Acc	-0.011	-0.022	-0.019	-0.015	-0.007	-0.013	-0.009	-0.044	-0.042	-0.012	-0.203
		DI	bad	0.076	lose	0.060	lose	0.066	0.039	0.076	0.095	bad	bad		DI	0.109	0.130	0.103	0.108	0.035	bad	bad	bad	bad	0.111	bad
		SPD	bad	0.052	lose	0.039	lose	0.044	0.025	0.052	0.068	bad	bad		SPD	0.077	0.092	0.070	0.078	0.021	bad	bad	bad	bad	0.078	bad
		EOD	0.044	0.062	0.045	0.059	0.033	0.054	0.042	0.079	0.064	0.023	bad		EOD	0.101	0.101	0.101	0.081	0.068	0.039	0.039	bad	0.107	0.092	bad
		AOD	lose	bad	lose	lose	lose	lose	lose	0.015	0.044	lose	bad		AOD	0.019	0.034	bad	0.027	lose	lose	lose	bad	0.059	0.016	bad
	XGB	Acc	0.003	-0.009	-0.014	-0.017	-0.016	-0.028	-0.026	0.005	-0.043	-0.006	-0.443	KNN	Acc	0.001	0.000	0.010	-0.005	0.000	0.005	0.009	-0.010	-0.002	-0.049	-0.420
		DI	0.070	0.091	0.119	0.101	bad	bad	bad	**0.051**	bad	0.035	lose		DI	0.112	**0.104**	**0.065**	0.123	0.038	**0.071**	**0.046**	0.160	0.150	0.135	bad
		SPD	0.050	0.065	0.082	0.069	bad	0.060	0.048	**0.038**	bad	0.025	bad		SPD	**0.072**	0.075	**0.043**	0.090	0.027	**0.047**	**0.028**	0.120	0.111	0.098	bad
		EOD	**0.069**	0.074	0.073	0.083	0.036	0.064	0.064	0.103	0.091	0.055	bad		EOD	0.104	**0.080**	**0.085**	0.076	0.066	**0.066**	**0.066**	0.128	0.115	0.002	bad
		AOD	0.020	0.020	0.041	0.037	lose	bad	bad	**0.015**	0.064	bad	bad		AOD	0.017	**0.011**	inv	0.034	lose	inv	inv	0.072	0.065	0.035	bad
Titanic	RF	Acc	-0.098	-0.130	-0.129	-0.128	-0.014	-0.166	-0.021	-0.010	-0.179	0.005	-0.178	GBC	Acc	-0.035	-0.076	-0.136	-0.126	0.065	-0.139	-0.015	-0.048	-0.189	-0.023	-0.165
		DI	1.549	1.864	1.848	1.849	lose	0.536	0.160	lose	2.024	0.449	2.303		DI	0.501	0.885	1.411	1.303	0.092	0.385	0.038	bad	1.769	bad	1.991
		SPD	0.395	0.571	0.551	0.545	lose	bad	bad	lose	bad	0.153	0.651		SPD	0.121	0.275	0.462	0.447	inv	bad	bad	0.285	bad	0.187	0.641
		EOD	0.274	0.404	0.445	0.446	lose	lose	lose	lose	0.481	0.045	bad		EOD	bad	bad	bad	bad	0.058	lose	lose	0.280	bad	0.116	0.426
		AOD	0.477	0.556	0.534	0.601	0.062	bad	0.133	0.097	0.618	0.336	bad		AOD	0.183	0.305	0.467	0.445	0.176	bad	0.081	0.374	0.568	0.306	bad
	LRG	Acc	-0.021	-0.084	-0.091	0.000	-0.007	-0.159	0.015	0.009	-0.227	-0.023	-0.152	XGB	Acc	-0.079	-0.101	-0.099	-0.110	-0.019	-0.129	-0.006	0.008	-0.157	0.009	-0.159
		DI	0.743	1.619	1.673	**0.149**	0.086	0.597	**0.214**	0.643	bad	bad	2.557		DI	1.364	1.701	1.470	1.663	lose	0.671	0.203	0.539	1.811	Inv	2.172
		SPD	0.101	0.552	0.597	**0.113**	0.063	bad	**0.007**	0.115	bad	0.312	0.785		SPD	0.280	0.542	0.406	0.491	lose	bad	0.065	0.051	0.567	Inv	0.642
		EOD	bad	0.467	0.557	**0.021**	lose	lose	**0.009**	0.171	bad	0.275	0.623		EOD	bad	0.400	0.285	0.389	lose	lose	lose	0.058	0.473	Inv	0.423
		AOD	0.140	0.562	0.632	**0.179**	0.101	bad	**0.067**	0.214	bad	0.420	bad		AOD	0.300	0.532	0.356	0.524	lose	bad	0.174	0.181	0.585	**0.062**	bad

Each cell shows the accuracy/bias difference between the original and repaired models. For accuracy, accuracy difference = new accuracy - old accuracy. For bias (DI, SPD, EOD, AOD), bias difference = old bias - new bias. Thus, a positive value indicates an improvement in bias/accuracy in the repaired model compared to the original and vice versa. For bias, if a method falls into either the good region (regular numbers) or the win-win region (**bold numbers**), the bias difference value will be provided. If it falls into any other region, the region type will be indicated. The values highlighted in blue denote the most effective bug fixing method. The data from this table is divided and analyzed in depth in Tables 3, 4, 6.

Table 3: Proportion of *Fair-AutoML*, *Auto-Sklearn*, and mitigation techniques that fall into each mitigation region

Method	Fairness Metric																				Dataset																
	DI					SPD					EOD					AOD					Adult Census					Bank Marketing					German Credit					Titanic				
	Lose	Bad	Inv	Good	Win	Lose	Bad	Inv	Good	Win	Lose	Bad	Inv	Good	Win	Lose	Bad	Inv	Good	Win	Lose	Bad	Inv	Good	Win	Lose	Bad	Inv	Good	Win	Lose	Bad	Inv	Good	Win	Lose	Bad	Inv	Good	Win
T1	0%	6%	0%	75%	19%	6%	12%	0%	63%	19%	18%	25%	0%	38%	19%	6%	0%	0%	75%	19%	0%	12%	0%	63%	25%	25%	0%	0%	75%	0%	6%	13%	0%	31%	50%	0%	19%	0%	81%	0%
T2	0%	0%	0%	81%	19%	0%	0%	0%	81%	19%	12%	13%	13%	56%	6%	0%	6%	6%	75%	13%	0%	0%	12%	75%	13%	12%	13%	6%	50%	19%	0%	6%	0%	69%	25%	0%	6%	0%	94%	0%
T3	6%	0%	0%	63%	31%	6%	0%	6%	63%	25%	6%	6%	6%	56%	26%	6%	6%	6%	56%	26%	0%	0%	0%	75%	25%	0%	12%	19%	13%	56%	19%	6%	6%	50%	19%	0%	6%	0%	94%	0%
T4	0%	0%	0%	75%	25%	0%	0%	6%	75%	19%	6%	6%	6%	63%	19%	6%	0%	0%	69%	25%	0%	0%	0%	100%	0%	5%	0%	13%	19%	63%	6%	0%	0%	94%	0%	0%	6%	0%	94%	0%
Avg	2%	2%	0%	73%	23%	3%	3%	3%	70%	21%	10%	13%	6%	53%	18%	5%	3%	3%	69%	20%	0%	3%	3%	78%	16%	11%	6%	9%	39%	35%	8%	6%	2%	61%	23%	0%	9%	0%	91%	0%
AS	38%	6%	6%	38%	12%	44%	6%	13%	31%	6%	56%	0%	6%	25%	13%	50%	0%	0%	31%	19%	13%	0%	19%	44%	24%	88%	0%	0%	12%	0%	38%	12%	0%	50%	0%	50%	0%	6%	25%	19%
R	0%	13%	0%	62%	25%	0%	31%	0%	44%	25%	50%	0%	19%	25%	6%	18%	38%	13%	19%	12%	13%	0%	6%	62%	19%	19%	6%	19%	25%	31%	13%	25%	6%	38%	18%	25%	50%	0%	25%	0%
DIR	13%	25%	6%	44%	12%	6%	50%	6%	25%	13%	50%	0%	13%	25%	12%	25%	6%	13%	44%	12%	31%	6%	19%	38%	6%	31%	38%	13%	18%	0%	13%	25%	6%	38%	18%	19%	13%	0%	43%	25%
PML	25%	38%	6%	12%	19%	19%	25%	6%	31%	19%	13%	13%	6%	49%	19%	6%	13%	6%	56%	19%	44%	6%	25%	25%	0%	0%	50%	0%	50%	0%	0%	25%	0%	50%	25%	19%	6%	0%	25%	50%
EO	0%	44%	0%	56%	0%	0%	56%	0%	44%	0%	0%	38%	0%	62%	0%	0%	31%	0%	69%	0%	0%	0%	0%	100%	0%	0%	100%	0%	0%	0%	0%	25%	0%	75%	0%	0%	44%	0%	56%	0%
FAX	0%	25%	19%	25%	31%	0%	6%	25%	44%	25%	0%	0%	38%	50%	12%	6%	6%	0%	38%	50%	0%	6%	31%	44%	19%	0%	0%	0%	31%	69%	6%	19%	0%	75%	0%	0%	13%	19%	38%	30%
ROC	6%	38%	0%	56%	0%	0%	50%	0%	50%	0%	25%	38%	12%	25%	0%	0%	88%	0%	12%	0%	19%	19%	0%	62%	0%	6%	68%	13%	13%	0%	6%	94%	0%	0%	0%	0%	31%	0%	69%	0%
Avg	7%	31%	5%	43%	14%	4%	36%	6%	40%	14%	23%	15%	14%	40%	8%	9%	30%	5%	40%	16%	18%	6%	14%	55%	7%	9%	44%	7%	23%	17%	6%	36%	2%	46%	10%	11%	26%	3%	43%	17%

The proportions in this table are determined based on the data presented in Table 2: proportion for fairness metric = # buggy cases of a metric fall into a region / # buggy cases of that metric, proportion for dataset = # buggy cases fall of a dataset into a region / # buggy cases of that dataset.

its automatic optimization of the best ML model for a given dataset. We tailored *Auto-Sklearn* to better fit our method in two ways: (1) its search space was restricted to the type of the faulty classifier - for example, if the faulty classifier is Random Forest, *Auto-Sklearn* will only optimize the hyperparameters and identify complementary components for that specific classifier. (2) The faulty model was set as the default model for *Auto-Sklearn*. These modifications are features of *Auto-Sklearn* that we utilized.

Methodology Configuration. We selected an increment value of α for β of 0.05 to balance the time between β search and model fixing processes. The user can opt for a more accurate value of β by decreasing the increment value and using a longer search time. To conduct search space pruning, we ran *Fair-AutoML* 10 times (n) with a 1-hour search time (t) to gather the best ML pipelines [9]. From each run, we collected the top 10 pipelines (k), resulting in 100 models per input. This pre-built search space includes a set of hyperparameters and the top 3 most frequently used complementary components (m). We have explored other parameter settings, but these have proven to provide optimal results.

Evaluation Configuration. We evaluate each tool on each buggy scenario 10 times using a random re-split of the data based on a 7:3 train-test split ratio [42]. The runtime for each run of *Fair-AutoML* and *Auto-Sklearn* is approximately one hour [28, 29]. The mean performance of each method is calculated as the average of the 10 runs, which is a commonly used practice in the fairness literature [4, 6, 14]. Our evaluation targets fixing 16 buggy models for 4 fairness metrics, resulting in a total of 64 buggy cases.

6.2 Effectiveness (RQ1)

We evaluate the effectiveness of *Fair-AutoML* by comparing it with *Auto-Sklearn* and existing bias mitigation techniques based on *Fairea* baseline. The comparisons are based on the following rules:

- **Rule 1**: A model is considered successfully repaired when its post-mitigation mean accuracy and fairness falls into win-win/good trade-off regions.
- **Rule 2**: A model that falls in the win-win region is always better than one falling into any other region.
- **Rule 3**: If two models are in the same trade-off region, the one with lower bias is preferred.

Our comparison rules for bug-fixing performance were established based on Fairea and our evaluations. Firstly, we define a successful bug fix as a fixed model that falls within the win-win or good trade-off regions, as these regions demonstrate improved fairness-accuracy trade-offs compared to the baseline in Fairea. Secondly, when comparing successfully fixed models in different trade-off regions (win-win versus good), we consider the win-win models to be superior as they offer improved fairness and accuracy. Lastly, for models that fall within the same trade-off region, the one with lower bias is deemed to be better, as our goal is to fix unfair models. Our evaluations then consider two aspects of the bug-fixing performance: the number of successful bug fixes and the number of times a bias mitigation method outperforms others.

6.2.1 Is Fair-AutoML effective in fixing fairness bugs? The results presented in Table 4 show that *Fair-AutoML* was effective in resolving 60 out of 64 (94%) fairness bugs, while *Auto-Sklearn* only fixed 28 out of 64 (44%) and bias mitigation techniques resolved up to 44 out of 64 (69%). This indicates that *Auto-Sklearn* alone was not effective in reducing bias, however, our methods were successful in enhancing AutoML to repair fairness bugs. Moreover, *Fair-AutoML* was able to repair more cases than other bias mitigation techniques, which often resulted in lower accuracy for lower bias. This highlights the effectiveness of our approaches in guiding AutoML towards repairing models for better trade-off between fairness and accuracy compared to the Fairea baseline.

Table 4: Fair-AutoML (FA) vs bias mitigation methods in fixing fairness bugs

	FA	AS	R	DIR	PML	EO	FAX	ROC
# bugs fixed	60	28	35	30	36	37	44	23
# best models	19	4	9	0	10	9	9	3

The results in this table are derived from the data presented in Table 2. The row *# bugs fixed* indicates the number of cases where the technique falls into either the win-win or good trade-off region. The row *# best models* represents the number of instances where a bias mitigation technique outperforms all other methods.

6.2.2 Does Fair-AutoML outperform bias reduction techniques? *Fair-AutoML* demonstrated superior performance in fixing fairness bugs compared to other bias mitigation techniques. The results presented in Table 4 indicate that 63 out of 64 buggy cases were fixed by *Fair-AutoML*, *Auto-Sklearn*, or bias mitigation techniques. Among the repaired buggy cases, *Fair-AutoML* outperformed other techniques 19 times (30%). On the other hand, *Auto-Sklearn* outperformed *Fair-AutoML* and bias mitigation techniques only 4 times (6%), and bias mitigation techniques outperformed other techniques 10 times at most (16%). This highlights that *Fair-AutoML* is often more effective in improving fairness and accuracy simultaneously or reducing more bias than other bias mitigation techniques.

6.3 Adaptability (RQ2)

To assess the adaptability of *Fair-AutoML*, we measure the proportions of each evaluated tools that fall into each fairness-accuracy trade-off region in different categories: fairness metric and dataset (Table 3). To further evaluate the adaptability of *Fair-AutoML*, instead of using our prepared models and datasets, we used the benchmark [59] of Parfait-ML to evaluate *Fair-AutoML*. Particularly, we evaluate *Fair-AutoML* and Parfait-ML on three different ML models (Decision Tree, Logistic Regression, Random Forest) on two datasets (Adult Census and COMPAS) (Table 5 and Figure 3).

6.3.1 Is Fair-AutoML more adaptable than existing bias mitigation techniques and Auto-Sklearn? Table 3 shows *Fair-AutoML* demonstrates exceptional repair capabilities across various datasets and fairness metrics, with a high rate of success in fixing buggy models. For example, in the Adult Census, Bank Marketing, German Credit, and Titanic datasets, *Fair-AutoML* (T4) repaired 100%, 82%, 94%, and 94% of the models, respectively. Similarly, in the DI, SPD, EOD, and AOD fairness metrics, *Fair-AutoML* (T4) achieved repair rates of 100%, 94%, 82%, and 94%. On the other hand, bias mitigation methods often show inconsistent results. For instance, Equalized Odds repaired all buggy cases in *Adult Census* but none in *Bank Marketing*. In fact, our methods effectively guides AutoML in hyperparameter tuning to reduce bias, leading to superior repair performance across different datasets and metrics.

6.3.2 Is Fair-AutoML effective in fixing fairness bugs on other bias mitigation methods benchmark? Based on evaluation of Parfait-ML [60], we only use accuracy and EOD as evaluation metrics for this evaluation. To make a fair comparison with Parfait-ML, we utilize the version of *Fair-AutoML* that incorporates EOD and accuracy as its cost function (T3). The results are displayed in Table

Table 5: Accuracy and fairness achieved by Fair-AutoML and Pafait-ML on Pafait-ML's benchmark

Data	Decision Tree				Logistic Regression				Random Forest			
	T3		PML		T3		PML		T3		PML	
	Acc	EOD	Acc	EOD	Acc	EOD	Acc	EOD	Acc	EOD	Acc	EOD
Adult	0.847	0.036	0.817	0.002	0.818	0.038	0.803	0.023	0.851	0.032	0.843	0.039
Compas	0.969	0.000	0.970	0.000	0.970	0.000	0.968	0.000	0.970	0.000	0.970	0.000

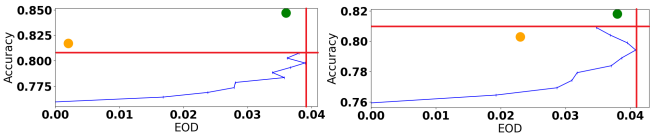

Figure 3: Accuracy and fairness achieved by Fair-AutoML (green circle) and Pafait-ML (orange circle) with Decision Tree (left) and logistic regression (right) on Adult dataset (Pafait-ML's benchmark). The blue line shows the Fairea baseline and red lines define the trade-off regions.

5, showcasing the accuracy and bias (EOD) achieved by both *Fair-AutoML* (T3) and Parfait-ML in Parfait-ML's benchmark. The table showcases the actual results of the repaired models, rather than the difference in accuracy/fairness between the original and repaired models. Upon inspection, the results for the COMPAS dataset for both *Fair-AutoML* and Parfait-ML are similar. However, for the Adult dataset, some differences arise. For instance, with the Random Forest classifier, *Fair-AutoML* performs better than Parfait-ML in both accuracy and EOD. With the Logistic Regression classifier, *Fair-AutoML* achieved a higher accuracy but higher bias compared to Parfait-ML. Nevertheless, *Fair-AutoML* falls into the win-win trade-off region, while Parfait-ML only falls into good trade-off region (Figure 3). With the Decision Tree classifier, both *Fair-AutoML* and Parfait-ML fall into the win-win trade-off region (Figure 3); however, Parfait-ML performed better since it has lower bias. These results highlights the generalization capability of *Fair-AutoML* to repair various datasets and ML models.

6.4 Ablation Study (RQ3)

We create an ablation study to observe the efficiency of the dynamic optimization function and the search space pruning separately. The ablation study compares the performance of the following tools:

- *Auto-Sklearn* (*AS*) represents AutoML.
- *Fair-AutoML* version 1 (*FAv1*) represents AutoML + dynamic optimization function.
- *Fair-AutoML* version 2 (*FAv2*) represents AutoML + dynamic optimization function + search space pruning.

To evaluate the efficiency of the dynamic optimization function, we compare the performance of *FAv1* with *Auto-Sklearn*. We compare *FAv1* with *FAv2* to observe the efficiency of the search space pruning approach. The complete result is shown in Table 6. Notice that we use *Fair-AutoML* to optimize different fairness metrics; thus, we only consider the metric that each tool tries to optimize. For instance, the results of Random Forest on Adult dataset in the Table 6 shows that achieved scores of 0.096 for DI, 0.014 for SPD, 0.024 for EOD, and 0.035 for AOD. This result means that T1 achieves 0.096 for DI, T2 achieves 0.014 for SPD, T3 achieves 0.024 for EOD, T4 achieves 0.035 for AOD. The evaluation only considers cases where the tools successfully repair the bug. The same rules described in RQ1 is applied in this evaluation.

Table 6: Trade-off assessment results of *Auto-Sklearn*, *FAv1*, and *FAv2*

	Metric	Model	AS	FAv1	FAv2	Model	AS	FAv1	FAv2
Adult Census	DI	RF	0.058	0.119	**0.096**	LRG	0.04	0.094	0.19
	SPD		0.016	0.053	**0.024**		0.005	0.041	0.075
	EOD		0.008	**0.015**	**0.014**		lose	0.043	0.044
	AOD		0.021	**0.025**	0.035		0.039	0.078	0.089
	DI	XGB	0.004	0.136	0.183	GBC	inv	bad	0.124
	SPD		0.003	0.046	0.074		inv	bad	0.055
	EOD		lose	0.015	0.037		inv	0.02	0.013
	AOD		0.017	0.049	0.053		**0.018**	inv	0.05
Bank Marketing	DI	RF	0.000	0.663	0.103	XGB2	lose	inv	0.158
	SPD		lose	0.026	bad		lose	inv	0.051
	EOD		lose	inv	lose		lose	lose	**0.003**
	AOD		0.016	**0.004**	0.032		lose	0.003	**0.027**
	DI	XGB1	lose	inv	0.098	GBC	lose	0.014	0.01
	SPD		lose	0.018	**0.062**		lose	lose	0.028
	EOD		lose	inv	**0.011**		lose	lose	inv
	AOD		lose	lose	**0.046**		lose	0.003	**0.025**
German Credit	DI	RF	lose	bad	bad	SVC	0.035	0.127	0.109
	SPD		lose	bad	0.052		0.021	0.078	0.092
	EOD		0.033	bad	0.045		0.068	0.112	0.101
	AOD		lose	lose	lose		lose	0.032	0.027
	DI	XGB	bad	bad	**0.07**	KNN	0.038	inv	**0.112**
	SPD		bad	lose	0.065		0.027	**0.012**	**0.075**
	EOD		0.036	lose	0.073		0.066	**0.050**	**0.085**
	AOD		lose	lose	0.037		lose	inv	0.034
Titanic	DI	RF	lose	1.04	1.549	GBC	**0.092**	0.447	0.501
	SPD		lose	0.525	0.571		inv	0.273	0.275
	EOD		lose	0.386	0.445		**0.058**	0.184	bad
	AOD		0.062	0.577	0.601		**0.058**	0.429	0.445
	DI	LRG	0.086	1.062	0.743	XGB	lose	1.205	1.364
	SPD		0.063	0.594	0.552		lose	0.314	0.542
	EOD		lose	0.556	0.557		lose	0.287	0.285
	AOD		0.101	0.651	**0.179**		lose	0.441	0.524

The data in Table 6 is created in the same ways as Table 2. For each method in the good trade-off region and win-win region (**bold number**), a trade-off measurement value is given; for other regions the region type is displayed. The values in blue, orange, and black indicate the top 1, top 2, top 3 bug fixing tools, respectively.

6.4.1 Are dynamic optimization function and search space pruning effective in fixing fairness bugs? From Table 6, our results show that the dynamic optimization function approach in *Fair-AutoML* helps fix buggy models more efficiently. Comparing the performance in fixing fairness bugs, *FAv1* outperforms *Auto-Sklearn* 39 times, while *Auto-Sklearn* outperforms *FAv1* only 7 times. The search space pruning approach in *Fair-AutoML* also contributes to more efficient bug fixing, as *FAv2* outperforms both *FAv1* and *Auto-Sklearn* 46 and 55 times respectively, while *FAv1* and *Auto-Sklearn* only outperform *FAv2* 14 and 4 times respectively.

7 DISCUSSION

In this work, we bring particular attention to the fairness-accuracy tradeoff while mitigating bias in ML models. Many works in the area only optimize fairness metrics by sacrificing accuracy, and do not consider the tradeoff rigorously. However, as shown by recent work [42], trivial mutation methods can also achieve fairness if accuracy is compromised in different magnitudes. Therefore, a rigorous evaluation method is necessary to demonstrate that the tradeoff is beneficial. Another limitation of existing tools is not generalizing over different ML classifiers (e.g., LRG, GBC, RF, XGB), multiple fairness metrics, and dataset characteristics. To that end, we leveraged the recent progress of AutoML in the context and achieved better tradeoff than SOTA methods. We believe that our approach

is versatile and can be applied to various ML problems. Particularly, the dynamic optimization function approach remains versatile across various datasets and models. Furthermore, the search space pruning approach is refined through pre-constructed database and a matching mechanism, that capitalizes on diverse datasets stored in repositories such as *OpenML* or *Kaggle*.

We implemented *Fair-AutoML* on top of *Auto-Sklearn* to ensure its wide applicability on ML algorithms. State-of-the-art bias mitigation techniques also primarily use classic ML algorithms [6, 7, 15, 16, 19, 60] that are supported by *Auto-Sklearn*. These models are more suitable than the DL models since the fairness critical tasks in prior works commonly use tabular datasets. Should one desire to explore alternative model types not directly supported by *Auto-Sklearn*, they can adopt the general ML model adoption of *Auto-Sklearn* [27].

Our approach also outlines several opportunities towards leveraging AutoML and search-based software engineering to ensure fairness in new ML models that are becoming available. First, the greedy weight identifier algorithm's performance might suffer for complex models due to computational costs (Algorithm 1). Second, search space pruning quantitatively estimates the similarity of datasets based on data characteristics. Thus, if we do not have a dataset similar enough to the input dataset, AutoML may not perform well. To address this, we plan to regularly update our database with new datasets. Lastly, constructing suitable search spaces, particularly for resource-intensive methods like deep learning, could entail significant computational expenses. Further works are needed to maximize the versatility and effectiveness of our approach over novel fairness-critical tasks. One key direction is to combine *Fair-AutoML* with other bias mitigation techniques, such as integrating Fair-AutoML's model with pre-processing bias mitigation methods to enhance overall pipeline fairness. Additionally, integrating *Fair-AutoML* with ensemble learning could improve both performance and fairness by capturing a broader range of biases and patterns. These directions could significantly amplify the impact of this work, making Fair-AutoML a potent tool for promoting fairness and equity in machine learning across various domains.

8 THREATS TO VALIDITY

Construct Validity. The choice of evaluation metrics and existing mitigation techniques may pose a threat to our results. We mitigate this threat by employing a diverse range of metrics and mitigation methods. First, we have used accuracy and four most recent and widely-used fairness metrics to evaluate *Fair-AutoML* and the state-of-the-art. These metrics have been commonly applied in the software engineering community [15, 16, 19, 60]. Second, we demonstrate the superiority of *Fair-AutoML* over state-of-the-art methods in different categories: pre-processing, in-processing, and post-processing, which are most advanced techniques from the SE and ML communities. For evaluating fairness and applying these mitigation algorithms except Parfait-ML [60], we have used AIF 360 toolkit. For evaluating Parfait-ML, we have used its original implementation. We create a baseline using the original Fairea implementation, enabling us to conduct a comprehensive comparison between our approach and existed mitigation methods. In the future, we intend to explore supplementary performance metrics and extend our analysis to incorporate additional mitigation techniques for a more comprehensive evaluation.

External Validity. To ensure an equitable comparison with cutting-edge bias mitigation techniques, we leverage a diverse array of real-world models, datasets, and evaluation scenarios. Particularly, we utilize a practical benchmark comprising 16 real-world models thoughtfully curated by prior research [6]. Then, these meticulously chosen models undergo evaluation using four extensively studied datasets in the fairness literature [10, 61, 62]. We conducted experiments under identical setups and subsequently validated our findings [6]. In addition to assessing *Fair-AutoML* against alternative methods within our established settings and benchmarks, we subject *Fair-AutoML* to evaluation using the Parfait-ML [60] benchmark, a leading-edge bias mitigation framework.

Internal Validity. Implementing *Fair-AutoML* on top of *Auto-Sklearn* may introduce a threat to its actual bias mitigation performance. In other words, the favorable outcomes achieved by *Fair-AutoML* could be attributed to its integration with *Auto-Sklearn*. To address this threat, we evaluated *Auto-Sklearn* on various benchmarks, comparing its performance with (*Fair-AutoML*) and without (*Auto-Sklearn*) our proposed approaches, to gauge the effectiveness of *Fair-AutoML*.

9 CONCLUSION

We present *Fair-AutoML*, an innovative system that enhances existing AutoML frameworks to resolve fairness bugs. The core concept of *Fair-AutoML* is to optimize the hyperparameters of faulty models to resolve fairness issues. This system offers two novel technical contributions: a dynamic optimization function and a search space pruning approach. The dynamic optimization function dynamically generates an optimization function based on the input, enabling AutoML to simultaneously optimize both fairness and accuracy. The search space pruning approach reduces the size of the search space based on the input, resulting in faster and more efficient bug repair. Our experiments show that *Fair-AutoML* outperforms *Auto-Sklearn* and conventional bias mitigation techniques, with a higher rate of bug repair and a better fairness-accuracy trade-off. In the future, we plan to expand the capabilities of *Fair-AutoML* to include deep learning problems, beyond the scope of the current study.

10 DATA AVAILABILITY

To increase transparency and encourage reproducibility, we have made our artifact publicly available. All the source code and evaluation data with detailed descriptions can be found here [33].

ACKNOWLEDGMENTS

This work was supported in part by National Science Foundation under Grant CCF-15-18897, CNS-15-13263, CNS-21-20448, CCF-19-34884, and CCF-22-23812. Additionally, our sincere gratitude extends to the reviewers for their invaluable insights, which significantly contributed to enhancing the quality of the paper.

REFERENCES

[1] 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.

[2] Shibbir Ahmed, Sayem Mohammad Imtiaz, Samantha Syeda Khairunnesa, Breno Dantas Cruz, and Hridesh Rajan. 2023. Design by Contract for Deep Learning APIs. In *ESEC/FSE'2023: The 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (San Francisco, California).

[3] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016. Machine bias risk assessments in criminal sentencing. *ProPublica, May* 23 (2016).

[4] Rachel KE Bellamy, Kuntal Dey, Michael Hind, Samuel C Hoffman, Stephanie Houde, Kalapriya Kannan, Pranay Lohia, Jacquelyn Martino, Sameep Mehta, Aleksandra Mojsilovic, et al. 2018. AI Fairness 360: An extensible toolkit for detecting, understanding, and mitigating unwanted algorithmic bias. *arXiv preprint arXiv:1810.01943* (2018).

[5] Reuben Binns. 2018. Fairness in machine learning: Lessons from political philosophy. In *Conference on Fairness, Accountability and Transparency*. PMLR, 149–159.

[6] Sumon Biswas and Hridesh Rajan. 2020. Do the Machine Learning Models on a Crowd Sourced Platform Exhibit Bias? An Empirical Study on Model Fairness. In *ESEC/FSE'2020: The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Sacramento, California, United States).

[7] Sumon Biswas and Hridesh Rajan. 2021. Fair preprocessing: towards understanding compositional fairness of data transformers in machine learning pipeline. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 981–993.

[8] Sumon Biswas and Hridesh Rajan. 2023. Fairify: Fairness Verification of Neural Networks. In *ICSE'2023: The 45th International Conference on Software Engineering* (Melbourne, Australia).

[9] Sumon Biswas, Mohammad Wardat, and Hridesh Rajan. 2022. The Art and Practice of Data Science Pipelines: A Comprehensive Study of Data Science Pipelines In Theory, In-The-Small, and In-The-Large. In *ICSE'2022: The 44th International Conference on Software Engineering* (Pittsburgh, PA, USA).

[10] Miranda Bogen and Aaron Rieke. 2018. Help wanted: An examination of hiring algorithms, equity, and bias. (2018).

[11] Ajay Byanjankar, Markku Heikkilä, and Jozsef Mezei. 2015. Predicting credit risk in peer-to-peer lending: A neural network approach. In *2015 IEEE symposium series on computational intelligence*. IEEE, 719–725.

[12] Toon Calders and Sicco Verwer. 2010. Three naive Bayes approaches for discrimination-free classification. *Data mining and knowledge discovery* 21, 2 (2010), 277–292.

[13] José P Cambronero, Jürgen Cito, and Martin C Rinard. 2020. Ams: Generating automl search spaces from weak specifications. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 763–774.

[14] L Elisa Celis, Lingxiao Huang, Vijay Keswani, and Nisheeth K Vishnoi. 2019. Classification with fairness constraints: A meta-algorithm with provable guarantees. In *Proceedings of the conference on fairness, accountability, and transparency*. 319–328.

[15] Joymallya Chakraborty, Suvodeep Majumder, and Tim Menzies. 2021. Bias in machine learning software: Why? how? what to do?. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 429–440.

[16] Joymallya Chakraborty, Suvodeep Majumder, Zhe Yu, and Tim Menzies. 2020. Fairway: a way to build fair ML software. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 654–665.

[17] Joymallya Chakraborty, Tianpei Xia, Fahmid M Fahid, and Tim Menzies. 2019. Software engineering for fairness: A case study with hyperparameter optimization. *arXiv preprint arXiv:1905.05786* (2019).

[18] Zhenpeng Chen, Jie M Zhang, Federica Sarro, and Mark Harman. 2022. A comprehensive empirical study of bias mitigation methods for software fairness. *arXiv preprint arXiv:2207.03277* (2022).

[19] Zhenpeng Chen, Jie M Zhang, Federica Sarro, and Mark Harman. 2022. MAAT: a novel ensemble approach to addressing fairness and performance bugs for machine learning software. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1122–1134.

[20] Alexandra Chouldechova. 2017. Fair prediction with disparate impact: A study of bias in recidivism prediction instruments. *Big data* 5, 2 (2017), 153–163.

[21] Alexandra Chouldechova, Diana Benavides-Prado, Oleksandr Fialko, and Rhema Vaithianathan. 2018. A case study of algorithm-assisted decision making in child maltreatment hotline screening decisions. In *Conference on Fairness, Accountability and Transparency*. PMLR, 134–148.

[22] Jeffrey De Fauw, Joseph R Ledsam, Bernardino Romera-Paredes, Stanislav Nikolov, Nenad Tomasev, Sam Blackwell, Harry Askham, Xavier Glorot, Brendan O'Donoghue, Daniel Visentin, et al. 2018. Clinically applicable deep learning for diagnosis and referral in retinal disease. *Nature medicine* 24, 9 (2018), 1342–1350.

[23] Matthias Ehrgott. 2005. *Multicriteria optimization*. Vol. 491. Springer Science & Business Media.

[24] Virginia Eubanks. 2018. *Automating inequality: How high-tech tools profile, police, and punish the poor*. St. Martin's Press.

[25] Michael Feffer, Martin Hirzel, Samuel C Hoffman, Kiran Kate, Parikshit Ram, and Avraham Shinnar. 2022. An Empirical Study of Modular Bias Mitigators and Ensembles. *arXiv preprint arXiv:2202.00751* (2022).

[26] Michael Feldman, Sorelle A Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. 2015. Certifying and removing disparate impact. In *proceedings of the 21th ACM SIGKDD international conference on knowledge discovery and data mining*. 259–268.

[27] Matthias Feurer. 2022. Auto-Sklearn Documentation. https://automl.github.io/auto-sklearn/master/

[28] Matthias Feurer, Katharina Eggensperger, Stefan Falkner, Marius Lindauer, and Frank Hutter. 2020. Auto-sklearn 2.0: The next generation. *arXiv preprint arXiv:2007.04074* (2020).

[29] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. 2015. Efficient and robust automated machine learning. In *Advances in neural information processing systems*. 2962–2970.

[30] Sorelle A Friedler, Carlos Scheidegger, Suresh Venkatasubramanian, Sonam Choudhary, Evan P Hamilton, and Derek Roth. 2019. A comparative study of fairness-enhancing interventions in machine learning. In *Proceedings of the conference on fairness, accountability, and transparency*. 329–338.

[31] 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.

[32] 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.

[33] Giang Nguyen, Sumon Biwas, and Hridesh Rajan. 2023. *Replication Package of the ESEC/FSE 2023 Paper Entitled "Fix Fairness, Don't Ruin Accuracy: Performance Aware Fairness Repair using AutoML*. https://doi.org/10.5281/zenodo.8280911

[34] Usman Gohar, Sumon Biswas, and Hridesh Rajan. 2023. Towards Understanding Fairness and its Composition in Ensemble Machine Learning. In *ICSE'2023: The 45th International Conference on Software Engineering* (Melbourne, Australia).

[35] Przemyslaw A Grabowicz, Nicholas Perello, and Aarshee Mishra. 2022. Marrying fairness and explainability in supervised learning. In *2022 ACM Conference on Fairness, Accountability, and Transparency*. 1905–1916.

[36] Jussi Hakanen and Joshua D Knowles. 2017. On using decision maker preferences with ParEGO. In *International Conference on Evolutionary Multi-Criterion Optimization*. Springer, 282–297.

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

[38] Xinran He, Junfeng Pan, Ou Jin, Tianbing Xu, Bo Liu, Tao Xu, Yanxin Shi, Antoine Atallah, Ralf Herbrich, Stuart Bowers, et al. 2014. Practical lessons from predicting clicks on ads at facebook. In *Proceedings of the eighth international workshop on data mining for online advertising*. 1–9.

[39] Mitchell Hoffman, Lisa B Kahn, and Danielle Li. 2018. Discretion in hiring. *The Quarterly Journal of Economics* 133, 2 (2018), 765–800.

[40] Max Hort, Zhenpeng Chen, Jie M Zhang, Federica Sarro, and Mark Harman. 2022. Bia mitigation for machine learning classifiers: A comprehensive survey. *arXiv preprint arXiv:2207.07068* (2022).

[41] Max Hort and Federica Sarro. 2021. Did you do your homework? Raising awareness on software fairness and discrimination. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 1322–1326.

[42] Max Hort, Jie M Zhang, Federica Sarro, and Mark Harman. 2021. Fairea: A model behaviour mutation approach to benchmarking bias mitigation methods. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 994–1006.

[43] Haifeng Jin, Qingquan Song, and Xia Hu. 2019. Auto-Keras: An Efficient Neural Architecture Search System. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*. ACM, 1946–1956.

[44] Kaggle. 2017. Adult Census Dataset. https://www.kaggle.com/datasets/uciml/adult-census-income

[45] Kaggle. 2017. Bank Marketing Dataset. https://www.kaggle.com/c/bank-marketing-uci

[46] Kaggle. 2017. German Credit Dataset. https://www.kaggle.com/datasets/uciml/german-credit

[47] Kaggle. 2017. Titanic ML Dataset. https://www.kaggle.com/c/titanic/data

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

[49] Faisal Kamiran, Asim Karim, and Xiangliang Zhang. 2012. Decision theory for discrimination-aware classification. In *2012 IEEE 12th International Conference on Data Mining*. IEEE, 924–929.

[50] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. 2012. Fairness-aware classifier with prejudice remover regularizer. In *Joint European conference on machine learning and knowledge discovery in databases*. Springer, 35–50.

[51] Konstantina Kourou, Themis P Exarchos, Konstantinos P Exarchos, Michalis V Karamouzis, and Dimitrios I Fotiadis. 2015. Machine learning applications in cancer prognosis and prediction. *Computational and structural biotechnology journal* 13 (2015), 8–17.

[52] Yanhui Li, Linghan Meng, Lin Chen, Li Yu, Di Wu, Yuming Zhou, and Baowen Xu. 2022. Training data debugging for the fairness of machine learning software. In *Proceedings of the 44th International Conference on Software Engineering*. 2215–2227.

[53] Milad Malekipirbazari and Vural Aksakalli. 2015. Risk assessment in social lending via random forests. *Expert Systems with Applications* 42, 10 (2015), 4621–4631.

[54] Giang Nguyen, Johir Islam, Rangeet Pan, and Hridesh Rajan. 2022. Manas: Mining Software Repositories to Assist AutoML. In *ICSE'22: The 44th International Conference on Software Engineering* (Pittsburgh, PA, USA).

[55] Luca Oneto, Michele Donini, Andreas Maurer, and Massimiliano Pontil. 2019. Learning fair and transferable representations. *arXiv preprint arXiv:1906.10673* (2019).

[56] Claudia Perlich, Brian Dalessandro, Troy Raeder, Ori Stitelman, and Foster Provost. 2014. Machine learning for targeted display advertising: Transfer learning in action. *Machine learning* 95, 1 (2014), 103–127.

[57] Ralph E Steuer and Eng-Ung Choo. 1983. An interactive weighted Tchebycheff procedure for multiple objective programming. *Mathematical programming* 26, 3 (1983), 326–344.

[58] Guanhong Tao, Weisong Sun, Tingxu Han, Chunrong Fang, and Xiangyu Zhang. 2022. RULER: discriminative and iterative adversarial training for deep neural network fairness. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1173–1184.

[59] Saeid Tizpaz-Niari, Ashish Kumar, Gang Tan, and Ashutosh Trivedi. 2022. https://github.com/Tizpaz/Parfait-ML

[60] Saeid Tizpaz-Niari, Ashish Kumar, Gang Tan, and Ashutosh Trivedi. 2022. Fairness-aware Configuration of Machine Learning Libraries. *arXiv preprint arXiv:2202.06196* (2022).

[61] 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.

[62] 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.

[63] Rhema Vaithianathan, Tim Maloney, Emily Putnam-Hornstein, and Nan Jiang. 2013. Children in the public benefit system at risk of maltreatment: Identification via predictive modeling. *American journal of preventive medicine* 45, 3 (2013), 354–359.

[64] Muhammad Bilal Zafar, Isabel Valera, Manuel Gomez Rogriguez, and Krishna P Gummadi. 2017. Fairness constraints: Mechanisms for fair classification. In *Artificial Intelligence and Statistics*. PMLR, 962–970.

[65] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. 2018. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*. 335–340.

[66] 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.