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:


Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for third-party components of this work must be honored. For all other uses, contact the owner/author(s).

ESEC/FSE ’23, December 3–9, 2023, San Francisco, CA, USA © 2023 Copyright held by the owner/author(s).


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 [9, 10, 20, 21, 23, 37, 38, 50, 52, 54, 55, 62]. However, ML software can exhibit discrimination due to unfairness bugs in the models [2, 5]. These bugs can result in skewed decisions towards certain groups of people based on protected attributes such as race, age, or sex [29, 30].

To address this issue, the software engineering (SE) community has invested in developing testing and verification strategies to detect unfairness in software systems [1, 7, 29, 30, 61]. Additionally, the machine learning literature contains a wealth of research on defining different fairness criteria for ML models and mitigating bias [11, 19, 25, 36, 47, 49, 63, 64]. Various bias mitigation methods have been proposed to build fairer models. Some approaches mitigate data bias by adapting the training data [14, 15, 51]; some modify ML models during the training process to mitigate bias [18, 31, 40, 57, 59], and others aim to increase fairness by changing the outcome of predictions [1, 61, 65].

Despite these efforts, current bias mitigation techniques often come at the cost of decreased accuracy [5, 41]. Their effectiveness varies based on datasets, fairness metrics, or the choice of protected attributes [17, 24, 25, 36]. Hort et al. proposed Faira [41], 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 [17].

Recent works [33, 41, 59] 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 [33].

Recent advancements in AutoML technology [27, 28, 42] 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 [28], 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, 30, 61] and 16 buggy models collected from a recent study [5]. 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 [28]. The artifact is available here [32].

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 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 $D_{\text{train}}$ and a validation dataset $D_{\text{val}}$, 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 $D_{\text{train}}$. ML developers aims to search for a hyperparameter configuration $\lambda^*$ and complementary components $c^*$ for model $M$ to obtain optimal fairness-accuracy on $D_{\text{val}}$. The complementary components can be ML algorithms combined with a classifier i.e., pre-processing algorithms.

2.1.2 AutoML. Given the search spaces $\Lambda$ 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):

$$ M = \arg \min_{\lambda \in \Lambda, c \in C} \text{Cost} \left( M_{\lambda,c}, D_{\text{val}} \right) $$

(1)

$$ \left( \lambda^*, c^* \right) = \arg \min_{(\lambda,c)} \text{Loss} \left( M_{\lambda,c}, D_{\text{train}} \right) $$

(2)

2.1.3 Measures. We consider a problem, where each individual in the population has a true label in $y \in \{0,1\}$. We assume a protected attribute $z \in \{0,1\}$, 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.

$$ \text{Accuracy} = \frac{\# \text{True positive} + \# \text{True negative}}{\# \text{Total}} $$

Fairness Measure. We use four ways to define group fairness metrics, which are widely used in fairness literature [3, 4, 29]: The Disparate Impact (DI) is the proportion of the privileged group with the favorable label divided by the proportion of the privileged group with the favorable label [25, 63].

$$ \text{DI} = \frac{P(y = 1|z = 0)}{P(y = 1|z = 1)} $$

The Statistical Parity Difference (SPD) quantifies the disparity between the favorable label’s probability for the privileged group and the favorable label’s probability for the privileged group [11].

$$ \text{SPD} = P(y = 1|z = 0) - P(y = 1|z = 1) $$

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

$$ \text{TPR}_u = P(y = 1|y = 1, z = 0); \text{TPR}_p = P(y = 1|y = 1, z = 1) \quad \text{EOD} = \text{TPR}_u - \text{TPR}_p $$

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 [36].

$$ \text{FPR}_u = P(\hat{y} = 1|y = 0, z = 0); \text{FPR}_p = P(\hat{y} = 1|y = 0, z = 1) \quad \text{AOD} = \frac{1}{2} \cdot |\text{FPR}_u - \text{FPR}_p| + |\text{TPR}_u - \text{TPR}_p| $$

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 [15, 41]. 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 [29, 39]:

- Pre-processing approaches reduce bias by pre-processing the training data. For instance, Fair-SMOOTE [14] addresses data bias by removing biased labels and balancing the distribution of positive
and negative examples for each sensitive attribute. *Reweighing* [47] decreases bias by assigning different weights to different groups based on the degree of favoritism of a group. *Disparate Impact Remover* [25] is a pre-processing bias mitigation technique that aims to reduce bias by editing feature values.

In *post-processing* approaches reduce bias by modifying ML models during the training process; e.g., *Parfait-ML* [59] present a search-based solution to balance fairness and accuracy by tuning hyperparameter to approximate the twined Pareto curves. *MAAT* [18] 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.

In *pre-processing* approaches change the outcome of prediction to reduce bias. This technique favors privileged groups’ instances and favors those of unprivileged groups lying around the decision boundary. For example, *Equalized Odds* [36] reduces the value of EOD by modifying the output labels. *Fam-AI* [34] 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* [48] 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. [28] 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 [12]. 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.

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* [12] and *Manas* [53]. 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 [28], 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 [41], and are not applicable to a wide range of datasets, metrics, and models [24, 25, 36]. 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 Fairera [41] and are presented in Table 1.

*Fairera* is designed to assess the trade-off between fairness and accuracy of bias mitigation techniques. The methodology of Fairera 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. Fairera 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 inverted trade-off region. If a bias mitigation method simultaneously decreases both bias and accuracy, it would fall into the

---

![Figure 1: Baseline fairness-accuracy trade-off](image-url)
The results of the region classification of six bias mitigation techniques - Reweighting [47], Disparate Impact Remover [25], Parfait-ML [59], Equalized Odds [36], FaXAI [34], Reject Option Classification [46] - 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 Fairest 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: Mean proportions of mitigation cases that fall into each mitigation region

<table>
<thead>
<tr>
<th>Criteria</th>
<th>Lose</th>
<th>Bad</th>
<th>Inv</th>
<th>Good</th>
<th>Win</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI</td>
<td>7%</td>
<td>31%</td>
<td>5%</td>
<td>43%</td>
<td>14%</td>
</tr>
<tr>
<td>SPD</td>
<td>4%</td>
<td>36%</td>
<td>6%</td>
<td>40%</td>
<td>14%</td>
</tr>
<tr>
<td>EOD</td>
<td>23%</td>
<td>15%</td>
<td>14%</td>
<td>40%</td>
<td>8%</td>
</tr>
<tr>
<td>AOD</td>
<td>9%</td>
<td>30%</td>
<td>5%</td>
<td>40%</td>
<td>16%</td>
</tr>
<tr>
<td>Dataset</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Adult</td>
<td>18%</td>
<td>6%</td>
<td>14%</td>
<td>55%</td>
<td>7%</td>
</tr>
<tr>
<td>Bank</td>
<td>9%</td>
<td>44%</td>
<td>7%</td>
<td>23%</td>
<td>17%</td>
</tr>
<tr>
<td>German</td>
<td>6%</td>
<td>36%</td>
<td>2%</td>
<td>46%</td>
<td>10%</td>
</tr>
<tr>
<td>Titanic</td>
<td>11%</td>
<td>26%</td>
<td>3%</td>
<td>43%</td>
<td>17%</td>
</tr>
<tr>
<td>Mean</td>
<td>11%</td>
<td>28%</td>
<td>7%</td>
<td>41%</td>
<td>13%</td>
</tr>
</tbody>
</table>

Mean: mean proportion of all mitigation cases. Lose: lose-lose trade-off region, Lose: lose-lose trade-off region, Inv: inverted 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.

Hort et al. [41] 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) [22], which allows us to weigh the importance of different objectives and create a single scalar cost function.

\[ A = \sum_{i=1}^{n} c_i \beta_i \]  

where, \( \beta_i \) denotes the relative weight of importance of \( c_i \):

\[ \sum_{i=1}^{n} \beta_i = 1 \]  

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 problem:

\[ Cost(M_{\lambda, c}, D(z)) = \beta * f + (1 - \beta) * (1 - a) \]  

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 \( A^* \) 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, \(|A^*| < |A|\) 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.

\[ M = \arg \min_{\lambda \in \Lambda, c \in C^*} Cost(M_{\lambda, c}, D_{bad}(z)) \]  

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 [28], but updated and modified to tackle bias issues. To the best of our knowledge,
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 \( \beta \) to minimize the cost function. In this section, we propose an approach to automatically estimate the optimal value of \( \beta \) 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 \( \beta \), 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 [41], as shown in the Figure 1. In other words, the pseudo-model always achieves the accuracy on any binary classification problem as follows:

\[
\alpha_0 = \max(P(Y = 1), P(Y = 0))
\]

Given an input, the pseudo-model achieves an accuracy of \( \alpha_0 \) and a bias value of \( f_0 \) on that input. We define the cost function, \( \text{Cost} \), of the buggy ML model with accuracy \( \alpha \) and bias value \( f \) on the input. As AutoML tries different hyperparameter configurations to fix the model, the values of \( \alpha \) and \( f \) may change over time. The upper bound of the cost function is defined as Equations 8 and 9:

\[
\text{Cost}(M_{L^k}, D(z)) < \beta * f_0 + (1 - \beta) * (1 - \alpha_0) \quad (8)
\]

\[
\Rightarrow \beta * f + (1 - \beta) * (1 - a) < \beta * f_0 + (1 - \beta) * (1 - \alpha_0) \quad (9)
\]

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):

\[
\beta * f + (1 - \beta) * (1 - a) < (1 - \beta) * (1 - \alpha_0) \quad (10)
\]

5.1.2 Lower bound of \( \beta \). In this work, we desire to optimize the value of \( \beta \) 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 \( \beta \) will place more emphasis on reducing bias. However, simply setting \( \beta \) 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 [35, 56]. To overcome this challenge, we aim to find the lower bound of \( \beta \), which can be done based on the upper bound of the cost function. From Equation 10, we get:

\[
\beta < \frac{\alpha_0 - a}{\alpha_0 - a + f} \quad (11)
\]

However, if the value of \( \beta \) is smaller than \( \frac{\alpha_0 - a}{\alpha_0 - a + f} \), the optimization function \( \text{Cost} \) will always meet its upper bound condition. If the value of \( \beta \) always satisfies the upper bound condition of the cost function regardless of fairness and accuracy, 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 \( \beta \) as Equation 12:

\[
\beta \geq \frac{\alpha_0 - a}{\alpha_0 - a + f} \quad (12)
\]

The intuition being that our method aims to increase the chance for AutoML to achieve better fairness. However, by setting \( \beta < \frac{a - \alpha_0}{a - \alpha_0 + f_0} \) and \( a > \alpha_0 \) (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 \( \beta \geq \frac{a - \alpha_0}{a - \alpha_0 + f_0} \) and \( a > \alpha_0 \). 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 \( \beta \) estimation. The final step is estimating the value of \( \beta \) 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 beginning, we have: \( a = a_1 \) and \( f = f_1 \). In that time, the lower bound of \( \beta \) is \( L = \frac{a_1 - \alpha_0}{a_1 - \alpha_0 + f_1} \), so we have:

\[
\beta = L + k, k \in [0, 1 - L] \quad (13)
\]

We present a greedy algorithm for estimating the value of \( \beta \), 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 \( \beta \). Next, we run Fair-AutoML on the input under time constraint \( t \) with a value of \( \beta \) set to \( \frac{a_1 - \alpha_0}{a_1 - \alpha_0 + f_1} \) (Line 2-8). As the algorithm searches, whenever Fair-AutoML finds a candidate model that meets the condition \( \text{Cost} < \text{Cost}_0 \) (Lines 10-12), the value of \( \beta \) is slightly increased by \( a \) (Line 10-12). If after \( N \) tries, Fair-AutoML cannot find a model that satisfies the condition, the final value of \( \beta \) is set to \( \beta = \beta - a \) for the remaining search time to prevent overfitting from an excessively high value of \( \beta \) (Lines 13-15). The algorithm returns the best model found (Line 16).
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 [12, 27]; however, this is the first application of data characteristics to prune the search space for fairness-aware AutoML.

### 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 [12, 27]; 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 [16]. 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 age, sex, race, we use all of them in our search spaces.

**Algorithm 1 Greedy Weight Identifier**

1. **Input:** a dataset $D$ with protected attribute $z$, buggy model $M$ hyperparameterized by $\lambda$, the increment value $\alpha$, the searching time $t$ and the threshold $N$.
2. $\beta = \frac{a_1 - a_0}{a_1 + a_0}\frac{1}{t}$
3. $\text{Cost}(M_{\lambda,c}, D(z)) = \beta \star f + (1 - \beta) \star (1 - a)$
4. $\text{Cost}_0(M_{\lambda,c}, D(z)) = (1 - \beta) \star (1 - a_0)$
5. count = 0
6. checker = False
7. while $t$ do
8. $M_{\lambda,c,\alpha} = \arg \min_{\lambda} \text{Cost}(M_{\lambda,c}, D(z))$
9. count = count + 1
10. if $\text{Cost}(M_{\lambda,c}, D(z)) < \text{Cost}_0(M_{\lambda,c}, D(z))$ then
11. if checker = False then
12. $\beta = \beta + \alpha$
13. count = 0
14. if count $\geq N$ and checker = False then
15. $\beta = \beta - \alpha$
16. checker = True
17. return $M_{\lambda^*}$

**Algorithm 2 Database Building**

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

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.

**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 $k \ast n$ total pipelines. From these pipelines, we extract and store the 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 $k \ast n$ 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
We consider that the most similar dataset to the new dataset is Algorithm 3. So, we analyze the effectiveness of technique indicates its performance across a diverse range of datasets/metrics. The adaptability of a bias mitigation technique allows us to assess the capability of an AutoML system in fixing fairness bugs. To answer this question, we quantify the number of fairness bugs and discuss the experimental details. Then, we answer research questions regarding the efficiency and adaptability of Fair-AutoML.

### 6.1 Experiment

#### 6.1.1 Benchmarks.** We evaluated our method using real-world fairness bugs sourced from a recent empirical study [5], 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 [9, 60, 61]:

- The Adult Census (race) 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) 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) has 1000 observations with 21 features containing credit information to predict good or bad credit.
- The Titanic (sex) 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) [47], Disparate Impact Remover (DIR) [25].
- The in-processing includes Parfait-ML (PML) [59].
- The post-processing includes Equalized Odds (EO) [36], FAX.AI (FAX) [34], Reject Option Classification (ROC) [48].

Auto-Sklearn. We explore the efficiency of Auto-Sklearn (AS) [28] 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 $\beta \ast DI + (1 - \beta) \ast (1 - accuracy)$ as a cost function.
- **T2** uses $\beta \ast SPD + (1 - \beta) \ast (1 - accuracy)$ as a cost function.
- **T3** uses $\beta \ast EOD + (1 - \beta) \ast (1 - accuracy)$ as a cost function.
- **T4** uses $\beta \ast AOD + (1 - \beta) \ast (1 - accuracy)$ 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 [28], taking advantage of

---

**Algorithm 3 Input Matching**

1. **Input:** a input dataset $D$ with the protected attribute $z$, the number of data points $n$, 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[$d_i$] = $\lfloor f_i - f \rfloor + |p_i - p|$
5. similarDataset = min(dist, key=dist.get)
6. dist $= ()$
7. for $z_j$ 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

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

<table>
<thead>
<tr>
<th>Model</th>
<th>Metric</th>
<th>T1</th>
<th>T2</th>
<th>T4</th>
<th>AS</th>
<th>R</th>
<th>DLR</th>
<th>PML</th>
<th>EO</th>
<th>FAX</th>
<th>ROC</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF</td>
<td>Acc</td>
<td>0.010</td>
<td>0.011</td>
<td>0.004</td>
<td>0.007</td>
<td>0.005</td>
<td>0.008</td>
<td>0.002</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.206</td>
<td>0.192</td>
<td>0.003</td>
<td>0.006</td>
<td>0.003</td>
<td>0.004</td>
<td>0.001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Loss</td>
<td>0.214</td>
<td>0.234</td>
<td>0.004</td>
<td>0.005</td>
<td>0.004</td>
<td>0.006</td>
<td>0.003</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Acc</td>
<td>0.031</td>
<td>0.029</td>
<td>0.003</td>
<td>0.005</td>
<td>0.004</td>
<td>0.006</td>
<td>0.002</td>
<td>0.04</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Std</td>
<td>0.205</td>
<td>0.192</td>
<td>0.003</td>
<td>0.006</td>
<td>0.003</td>
<td>0.004</td>
<td>0.001</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Loss</td>
<td>0.214</td>
<td>0.234</td>
<td>0.004</td>
<td>0.005</td>
<td>0.004</td>
<td>0.006</td>
<td>0.003</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| XGB     | Acc    | 0.013  | 0.016  | 0.008  | 0.007  | 0.005 | 0.007 | 0.003 | 0.04 |     |     |
|         | Std    | 0.206  | 0.192  | 0.003  | 0.006  | 0.003 | 0.004 | 0.001 |     |     |     |
|         | Loss   | 0.214  | 0.234  | 0.004  | 0.005  | 0.004 | 0.006 | 0.003 |     |     |     |
|         | Acc    | 0.031  | 0.029  | 0.003  | 0.005  | 0.004 | 0.006 | 0.002 | 0.04 |     |     |
|         | Std    | 0.205  | 0.192  | 0.003  | 0.006  | 0.003 | 0.004 | 0.001 |     |     |     |
|         | Loss   | 0.214  | 0.234  | 0.004  | 0.005  | 0.004 | 0.006 | 0.003 |     |     |     |

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

<table>
<thead>
<tr>
<th>Method</th>
<th>Fairness Metric</th>
<th>Dataset</th>
<th>Adult Census</th>
<th>Bank Marketing</th>
<th>German Credit</th>
<th>Titanic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>True Positives</td>
<td>Low</td>
<td>Rad</td>
<td>Inv</td>
<td>Good Churn</td>
<td>Low</td>
</tr>
<tr>
<td>Ti</td>
<td>0% 0% 0% 0%</td>
<td>6% 6% 6%</td>
<td>6% 6% 6% 6%</td>
<td>6% 6% 6% 6%</td>
<td>0% 0% 0% 0%</td>
<td>6% 6% 6%</td>
</tr>
<tr>
<td>Ti</td>
<td>0% 0% 0% 0%</td>
<td>6% 6% 6%</td>
<td>6% 6% 6% 6%</td>
<td>6% 6% 6% 6%</td>
<td>0% 0% 0% 0%</td>
<td>6% 6% 6%</td>
</tr>
<tr>
<td>Ti</td>
<td>0% 0% 0% 0%</td>
<td>6% 6% 6%</td>
<td>6% 6% 6% 6%</td>
<td>6% 6% 6% 6%</td>
<td>0% 0% 0% 0%</td>
<td>6% 6% 6%</td>
</tr>
<tr>
<td>Ti</td>
<td>0% 0% 0% 0%</td>
<td>6% 6% 6%</td>
<td>6% 6% 6% 6%</td>
<td>6% 6% 6% 6%</td>
<td>0% 0% 0% 0%</td>
<td>6% 6% 6%</td>
</tr>
</tbody>
</table>

The proportions in this table are based on the data presented in Table 2: proportion for fairness metric = # buggy cases of a metric falling into a region / # buggy cases of that metric, proportion for dataset = # buggy cases of a dataset falling into a region / # buggy cases of that dataset.
Methodology Configuration. We selected an increment value of $\alpha$ for $\beta$ of 0.05 to balance the time between $\beta$ search and model fixing processes. The user can opt for a more accurate value of $\beta$ 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 [8]. 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 [41]. The runtime for each run of Fair-AutoML and Auto-Sklearn is approximately one hour [27, 28]. 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 [3, 5, 13]. 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.

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 illustrate 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 tool 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 [58] 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 [59], 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>
<thead>
<tr>
<th># bugs fixed</th>
<th>FA</th>
<th>AS</th>
<th>R</th>
<th>DIR</th>
<th>PML</th>
<th>EO</th>
<th>FAX</th>
<th>ROC</th>
</tr>
</thead>
<tbody>
<tr>
<td># best models</td>
<td>10</td>
<td>4</td>
<td>9</td>
<td>8</td>
<td>10</td>
<td>8</td>
<td>9</td>
<td>3</td>
</tr>
</tbody>
</table>

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.
Table 5: Accuracy and fairness achieved by Fair-AutoML and Parfait-ML on Parfait-ML’s benchmark

<table>
<thead>
<tr>
<th>Data</th>
<th>Decision Tree</th>
<th>Logistic Regression</th>
<th>Random Forest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Acc</td>
<td>EOD</td>
<td>PML</td>
</tr>
<tr>
<td>Adult</td>
<td>0.847</td>
<td>0.086</td>
<td>0.817</td>
</tr>
<tr>
<td>Compa</td>
<td>0.969</td>
<td>0.000</td>
<td>0.970</td>
</tr>
</tbody>
</table>

Figure 3: Accuracy and fairness achieved by Fair-AutoML (green circle) and Parfait-ML (orange circle) with Decision Tree (left) and logistic regression (right) on Adult dataset (Parfait-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 highlight 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.

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 implement Fair-AutoML on top Auto-Sklearn to focus classic ML algorithms for the following reasons: (1) DL models are not suitable for tabular data, which is small in size and commonly used in fairness research. (2) State-of-the-art bias mitigation techniques also primarily use classic ML algorithms [5, 6, 14, 15, 18, 59]. As an extension of the Auto-Sklearn framework, Fair-AutoML seamlessly extends its applicability to encompass datasets and ML models that are supported by Auto-Sklearn, or those that can potentially be integrated into the Auto-Sklearn framework. Should one desire to explore alternative model types not directly supported by Auto-Sklearn, they have the option to refer to the Auto-Sklearn documentation [26] for guidance on implementation.
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 input dataset types and targeted models, making it suitable for any type of ML problem. This approach exclusively requires targeted model performance data, with a particular emphasis on fairness and accuracy within this study. Furthermore, the search space pruning approach is refined through the utilization of a pre-constructed database and a matching mechanism that capitalizes on linear computable data attributes, prevalent across diverse datasets stored in repositories such as OpenML or Kaggle. Past studies, exemplified by Manas [53] and Auto-Sklearn [27, 28], have convincingly demonstrated how data characteristics can aptly ascertain dataset similarity.

Besides the stated advantages, our approach needs to overcome several limitations to ensure applicability across diverse machine learning problems. First, the greedy weight identifier algorithm’s performance might suffer with substantial datasets and complex models due to computational costs (Algorithm 1). Second, search space pruning quantitatively estimate the similarity of datasets based on data characteristics. This allows us to prune the search space by using the most similar dataset to the input dataset. However, this method cannot identify when a dataset similar enough with the input dataset. Thus, if we do not have a dataset similar enough to the input dataset, our method 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. Addressing these concerns is crucial for maximizing the versatility and effectiveness of our approach.

As research progresses on bias mitigation in machine learning through the innovative Fair-AutoML approach, exciting opportunities for future exploration arise. 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 [14, 15, 18, 59]. 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 [59], we have used AIF 360 toolkit. For evaluating Parfait-ML, we have used its original implementation. We create a baseline using the original FairaML 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 [5]. Then, these meticulously chosen models undergo evaluation using four extensively studied datasets in the fairness literature [9, 60, 61]. We conducted experiments under identical setups and subsequently validated our findings [5]. 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 [59] 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 [32].

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

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


