POZNAN UNIVERSITY OF TECHNOLOGY

FACULTY OF COMPUTING AND TELECOMMUNICATIONS
Institute of Computing Science

Bachelor's thesis

DETOXAI: IMPROVING FAIRNESS IN NEURAL NETWORKS VIA CONCEPT UNLEARNING

Ignacy Stępka, 148179
Łukasz Sztukiewicz, 151959
Michał Wiliński, 151938

Supervisor
prof. dr hab. inż. Jerzy Stefanowski

POZNAŃ 2025

ACKNOWLEDGMENTS

We would like to express our sincere gratitude to Professor Jerzy Stefanowski for his continuous support as the supervisor of this thesis. His expert guidance and insightful feedback have been invaluable, substantially enriching the quality of this work. We are also deeply thankful to Dr. Mateusz Lango and the Group of Horribly Optimistic Statisticians (GHOST) scientific club for introducing us to the fascinating field of machine learning. Their efforts have created a collaborative environment in which we could nurture our passion and expand our knowledge. Furthermore, we would like to extend our appreciation to our families for their unwavering support throughout this academic journey. In addition, Łukasz would like to express his genuine appreciation to his parents and his sister Weronika for their incredible support and motivation.

ABSTRACT

Machine learning (ML) systems have achieved remarkable success across numerous domains due to their predictive capabilities. However, their widespread adoption gives rise to many ethical concerns, particularly in light of recent regulations that posit fairness as a legal requirement. While foundational research on fairness has produced effective toolkits for tabular data, these approaches have become prohibitively expensive when adapted to deep learning systems dealing with high-dimensional data such as images. The absence of specialized debiasing methods and proper software frameworks leaves a critical gap in addressing fairness issues for these systems. To address these challenges, we introduce DetoxAI, a Python-based software framework for post-hoc debiasing of neural networks in image classification tasks. Designed with deep learning in mind, DetoxAI integrates state-of-the-art interventions, evaluation metrics, and visualization tools into a unified, production-ready ecosystem. Our approach applies post-training adaptation, allowing users to mitigate bias while maintaining model performance. Focusing on high-level semantic representations, DetoxAI addresses the unique challenges posed by vision data, where protected attributes such as race or gender are not explicitly encoded. This toolkit provides a modular interface for bias mitigation, making it accessible and adaptable for real-world applications. Through experimental studies, we quantitatively demonstrate that DetoxAI reliably improves upon baseline vanilla models on a fairness-performance trade-off. Moreover, with attribution maps, we qualitatively show that DetoxAI's methods can shift the model's focus away from protected attributes.

STRESZCZENIE

Współczesne systemy uczenia maszynowego (ML) osiągają spektakularne sukcesy w wielu dziedzinach, jednak rosnące wymagania związane z ich zastosowaniami oraz automatyzacją decyzji, które mogą dotyczyć ludzi, skłaniają do uwzględniania aspektów etycznych podczas projektowania tych systemów. W szczególności dotyczy to zapewniania ich sprawiedliwości (z ang. fairness), postulowanej w regulacjach prawnych. Podczas gdy badania podstawowe nad sprawiedliwością doprowadziły do powstania skutecznych narzędzi dla danych tabelarycznych, adaptacja tych podejść do systemów uczenia głębokiego, operujących na danych o wysokiej wymiarowości, a w szczególności danych obrazowych, staje się nieproporcjonalnie kosztowna. Brak dedykowanych metod redukcji uprzedzeń oraz odpowiednich narzędzi programistycznych tworzy istotną lukę w rozwiązaniu problemów związanych ze sprawiedliwością. W celu sprostania tym wyzwaniom stworzyliśmy DetoxAI, oprogramowanie oparte o język Python, służące do redukcji uprzedzeń w sieciach neuronowych stosowanych w zadaniach klasyfikacji obrazów. DetoxAI zostało zaprojektowane z myślą o uczeniu głębokim i integruje techniki interwencji, miary oceny oraz narzędzia wizualizacyjne w jedną platformę. Nasze podejście wykorzystuje adaptację modelu predykcyjnego po zakończeniu procesu trenowania, umożliwiając użytkownikom redukcję uprzedzeń przy jednoczesnym zachowaniu wysokiej użyteczności modelu. Skupiając się na wysokopoziomowych reprezentacjach semantycznych, DetoxAI odpowiada na unikalne wyzwania związane z danymi obrazowymi, gdzie chronione atrybuty, takie jak grupa etniczna czy płeć, nie są jawne. DetoxAI oferuje modularny interfejs do redukcji uprzedzeń, co czyni go narzędziem gotowym do praktycznego zastosowania. Nasze eksperymenty wykazują, że DetoxAI systematycznie poprawia wyniki w porównaniu z modelami bazowymi w zakresie przetargu sprawiedliwość-trafność predykcji. Ponadto, za pomocą map atrybucji obserwujemy, że metody zawarte w DetoxAI potrafią skierować uwagę modelu w taki sposób, aby unikał wykorzystywania chronionych pojęć podczas procesu wnioskowania.

Contents

Chapter 1

Introduction

1.1 Motivation

Machine learning (ML) systems have revolutionized how we address complex real-world problems. From image classification [1, 2, 3] and face recognition [4] to coherent text generation [5, 6] and image captioning [7] these systems have demonstrated state-of-the-art performance by leveraging vast amounts of data. However, as ML systems become widespread, concerns about safety, transparency, and ethical implications of human-machine collaboration have become urgent [8].

One of the most pressing concerns is *bias*, a systematic prejudice embedded in the data that ML systems learn from and often use during the reasoning process [9]. Even when datasets appear balanced, models can achieve disparate performance across protected groups, such as those defined by gender or race [10]. The implications of such biases can be far-reaching and often lead to harm to individuals and marginalized communities. To address these concerns, we must strive for *fairness* in our models, defined as the absence of any favoritism toward individuals or groups based on their inherent characteristics.

Consider the COMPAS algorithm, used in the U.S. judicial system to predict recidivism (the likelihood that a defendant will commit another crime). It was found to assign higher risk scores to black defendants compared to white defendants with similar criminal profiles [11]. However, developers defended their algorithm by arguing that COMPAS achieved *equal predictive parity* (a measure in this case defined as the same false positive rates). In the later analysis, it was shown that both parties were somewhat correct [12], and this argument highlighted a broader issue: fairness is not a one-size-fits-all concept. Instead, it is a multifaceted issue with often conflicting definitions. This underscores the importance of enabling decision-makers and practitioners to tailor their systems to address specific fairness concerns, considering potential biases and mitigating risks most relevant to the context of their application [13].

Bias in ML does not necessarily have to come from malicious intent. It can arise as a result of a variety of design factors, including flawed problem modeling [14], systemic inequities [11], or inherent biases in the data generation process [15, 16]. Addressing systemic sources of bias, such as social inequities, often lies beyond the reach of model developers; thus, practitioners have turned to algorithmic fairness to address bias using the available data and models. These interventions typically fall into three stages of the ML pipeline: pre-processing (addressing bias in the gathered data), in-processing (modifying training to incorporate fairness objectives), and post-processing (adjusting model outputs after training) [17, 18, 19]. Although these methods aim to balance model performance and fairness, they are still constrained by the biases inherent in the data generation process [20, 13].

Most algorithmic fairness research has focused on tabular data, where protected attributes such as gender or race are explicitly labeled and can be directly manipulated. However, as ML has evolved, there has been a paradigm shift toward deep learning, particularly for high-dimensional data modalities like images and text [21]. In computer vision (CV) and natural language processing (NLP), pre-trained models such as ResNet [1], EfficientNet [3], GPT [5], and LLAMA [6] dominate the field. These models, often with billions of parameters, are computationally expensive to train, making pre- and in-processing fairness techniques less feasible. As a result, attention has increasingly shifted to post-processing techniques that apply fairness-aware adjustments without the need for costly retraining.

In our work, we focus on computer vision, a problem with high-dimensional data modality that is natural to humans but difficult to process by statistical models. In image data, features are pixels that lack explicit semantic meaning. For example, attributes such as gender or race are not directly represented as single features in an image but must be inferred holistically from complex, high-dimensional representations. This complexity introduces additional challenges in the assessment and mitigation of bias in vision models. Unlike in tabular data, where one can discard a protected feature to ensure fairness (albeit with limitations), it is not straightforward to remove specific features or concepts from an image. Instead, fairness interventions in vision models must focus on high-level semantic concepts, making the task inherently more complex.

Our work addresses these challenges by exploring fairness-improving techniques for vision models through post-training adaptation. Specifically, we implement a suite of 9 methods that work under a variety of bias mitigation paradigms, including

- Concept unlearning: a removal of a certain concept from neural network's internal representations (*P-ClArC*, *A-ClArC*, *RR-ClArC*, *LEACE*).
- Fairness-focused fine-tuning: performing a few fine-tuning steps with a modified loss function (*ZhangAL*, *SavaniAFT*).
- Per-layer optimization: a small perturbation of model parameters to achieve certain fairness goals (*SavaniRP*, *SavaniLWO*).
- Classification threshold optimization: an adjustment of the classification threshold to maximize fairness while sacrificing as little predictive performance as possible (*NT*).

To verify the practical usefulness of the implemented debiasing techniques, we perform a suite of experimental studies. Specifically, we quantitatively analyze the performance-fairness trade-off in different image classification scenarios and qualitatively examine the attribution maps, which highlight the parts of the image that were relevant to a given prediction.

In general, by focusing on high-level representations of protected attributes, our objective in this work is to mitigate bias while maintaining predictive performance. This approach aligns with the practical constraints of modern ML workflows, offering a useful technical solution for practitioners and researchers navigating and analyzing the trade-offs between fairness and performance.

1.2 Aim and scope of work

This thesis introduces DetoxAI, a software toolkit designed to enhance the fairness of neural networks in the domain of computer vision through the process of debiasing. The objective was to create a unified software ecosystem that implements and evaluates bias mitigation algorithms and is distributed as an open-source package. Specifically, our goal was to:

1. Develop a robust and production-ready toolkit that enables ML practitioners and researchers to systematically mitigate biases across different neural network architectures in image classification problems. This includes implementations of state-of-the-art model debiasing algorithms, fairness metrics, and visualization tools for model explainability and bias analysis.

2. Conduct a systematic analysis of existing model debiasing methods, comparing the efficacy of in-processing methods. That is, examine trade-offs between the model's predictive performance and fairness improvements.

3. Investigate the effects of debiasing on model decision-making processes using attribution maps popular in Explainable AI to provide insight into how different techniques modify internal representations and the decision function. This analysis aims to bridge the gap between algorithmic fairness and practical model behavior, offering a deeper understanding of debiasing mechanisms.

Through these objectives, DetoxAI aims to accelerate the adoption of fairness-aware machine learning practices by providing a comprehensive, well-documented, and empirically validated toolkit for building more equitable AI systems.

The scope of this work encompasses the following primary directions:

1. **Software Package**
   - Implementation of DetoxAI as a production-ready library integrating state-of-the-art post-hoc bias detection and mitigation techniques.
   - Development of standardized interfaces for bias mitigation and model evaluation.
   - Creation of visualization tools for analyzing model predictions.
   - Integration with a modern PyTorch-based machine learning stack.
2. **Empirical Validation**
   - Systematic evaluation across selected benchmark scenarios based on CelebA dataset [22], in the task of analyzing fairness in facial attribute prediction.
   - Comparative analysis of selected debiasing approaches.
   - Qualitative analysis of the effectiveness of implemented debiasing techniques.

1.3 Team members contributions

The work presented in this thesis was a collaborative effort, with team members' duties and contributions often overlapping. Many tasks were completed during pair programming sessions, adhering to Extreme Programming (XP) practices. This approach was particularly valuable during the ideation phase, where we collectively conducted literature reviews and analyzed methods to identify gaps in the field, implement proof-of-concept solutions, and test them rapidly. However, the main responsibilities of each team member are outlined below:

Ignacy Stępka Implementation of the following debiasing methods in the DetoxAI package: *LEACE*, *ZhangAL*, *SavaniRP*, *SavaniLWO*, *SavaniAFT*, *P-ClArC*, *RR-ClArC*, *A-ClArC*. Design and implementation of the visualization module, DetoxAI's package interface, model evaluation tools, on-premise dataset download utility, and integration with pre-trained torchvision model backbones (e.g., ResNet, VGG). Conducting experiments and evaluations of the experimental

environment and DetoxAI integration. Literature review on ML fairness, fairness metrics, methods, and examples of fairness issues in real ML systems.

Łukasz Sztukiewicz Designed the backbone structure of the DetoxAI package and implemented core components, including data handling (loading, pre-processing, variant creation, rebalancing), model training and evaluation, performance and fairness metrics, and utility functions for experiment reproducibility. Designed and developed the experimental environment, including the experiment pipeline, Command Line Interface, configuration management (Hydra), agents, and experiment tracking services. Set up the cloud environment, Continuous Integration pipeline, packaging, and distribution for DetoxAI. Designed and ran proof-of-concept experiments and executed final experiment suites. Conducted a literature review on related software and bias mitigation techniques.

Michał Wiliński Contributed to DetoxAI package by implementing *NT* optimizer for model debiasing and Concept Activation Vectors (CAVs) system with multiple variants, including signal-based CAVs. Designed and developed the activation collection system for neural network layers. Created a test suite for method validation and evaluation of fairness and performance metrics. Aided in debugging and implementing the proper testing methodology in DetoxAI package. Conducted a literature review on linear explainability methods and CAV approaches.

1.4 Thesis structure

The structure of the work is as follows. Chapter 2 briefly presents the necessary concepts and reviews the related literature. This is an introduction to deep learning and image classification, a review of the literature on fair machine learning, covering definitions and metrics, a brief overview of existing bias mitigation techniques, and a review of related fairness-oriented software. Chapter 3 describes our proposed solution, a software package called DetoxAI, including the library architecture, implemented debiasing techniques, visualization tools, and technical details. Chapter 4 describes the software and research environment created for framework validation and empirical experimentation. Chapter 5 presents experimental results grouped into five research questions that empirically validate the practical utility of DetoxAI. Finally, Chapter 6 provides conclusions, discusses limitations, and outlines directions for future work.

Automated grammar checkers usage acknowledgment We acknowledge the use of Large Language Models (LLMs) and grammar checkers to refine our writing. However, the entire text, including its content and ideas, is our original work.

Chapter 2

Background

This chapter presents the background necessary to derive and understand the proposed framework. It is organized as follows. First, we introduce the main deep learning architectures for image classification (Section 2.1). Then, we briefly discuss explainability techniques in deep learning (Section 2.2). Next, we introduce the concept of fairness in machine learning, definitions, metrics, and the issue of trade-offs between performance and fairness (Section 2.3). We then cover the methods that aim to ensure fairness through post-hoc debiasing (Section 2.4). Finally, we cover the existing fairness-oriented Python frameworks and libraries, and point out their limitations (Section 2.5).

Here we briefly define the terminology that we later use in the work:

Term	**Definition**
protected (sensitive) attribute	An attribute that partitions a population into groups that have parity in terms of benefit received such as race, gender and religion. Protected attributes are not universal, but are application specific.
group fairness	Is achieved when the groups defined by protected attributes receive similar treatments or outcomes.
bias	A systematic error. In the context of fairness, we are concerned with unwanted bias that places privileged groups at advantage and unprivileged groups at disadvantage.
fairness metric	A quantification of bias in model's predictions.
bias mitigation (debiasing) algorithm	A procedure for reducing bias in models.
performance (predictive performance)	A broadly-defined measure of model performance and usability in a given task. Depending on the context it can be equivalent to e.g., Accuracy or GMean.

Table 2.1: Glossary. In this work we use the terms in parentheses interchangeably.

2.1 Deep learning for image classification

Image classification is the process of assigning a single label to an image based on its content [23]. Early approaches relied on manually designed features such as edges, textures, and color histograms [24]. However, deep learning marked a paradigm shift, enabling the automatic extraction of hierarchical and meaningful representations directly from raw pixel data. This breakthrough was made possible by LeCun et al. [25]'s Convolutional Neural Networks (CNNs), which drew inspiration from the organization of the animal visual cortex [26]. Unlike traditional neural networks, CNNs exploit the spatial structure of images. The fundamental components of CNNs include *convolutional layers* that apply learnable filters across input data, performing element-wise matrix multiplication to extract spatial features, *pooling layers* that provide dimensionality reduction, and *fully connected layers* that combine these features for final classification, usually with a *softmax layer* that converts logit scores into class probabilities. The goal of this architecture is to progressively learn increasingly complex features, ranging from elementary edges and textures in the initial layers to complex objects and textures in deeper layers [27].

Building upon LeCun et al. [25]'s early work in CNNs for pattern recognition, the field of deep learning experienced a transformative moment when Krizhevsky et al. [28]'s AlexNet achieved a top-5 error rate of 15.3% on the ImageNet dataset [29], significantly outperforming the previous state-of-the-art method by 10.9 percentage points, which was based on Fisher Vectors [30].

After that, a variety of diverse architectures were proposed, improving certain aspects of existing models. For example, the VGG architecture [2] introduced the concept of deeper networks with smaller convolutional filters (e.g., 3×3), showing that increasing depth could enhance performance. The ResNet family of models [1] addressed the problem of vanishing gradients by introducing residual connections, enabling the training of extremely deep networks with hundreds of layers. EfficientNet [3] introduced a scaling method that balances depth, width, and resolution, achieving superior performance with fewer parameters and computations. Recently, Vision Transformers (ViTs) [31] adapted transformer architectures from natural language processing to image classification. By dividing images into patches and processing them as sequences, ViTs have challenged the dominance of CNNs, offering new perspectives on representation learning. While Vision Transformers have gained traction, our work focuses on CNN-based architectures due to their maturity and compatibility with existing debiasing techniques that usually do not support attention mechanisms. We also explore advancements in the deep learning architectures and training introduced after 2020 to keep our work relevant and up-to-date.

For example, instead of the original EfficientNet architecture, we adopt its refined version, EfficientNet V2 [32], which improves training efficiency and achieves better performance on modern hardware. Considering the growing importance of edge processing and mobile devices, we also use MobileNet [33], characterized by depth-wise separable convolutions that reduce computational complexity while maintaining accuracy. Furthermore, we adopt ConvNext [34], a reimagined ResNet architecture that integrates transformer-inspired design principles to deliver state-of-the-art performance while maintaining architectural simplicity.

2.2 Explainability

A key challenge in deploying AI solutions is the "black box" nature of many algorithms. Understanding why an AI system makes a particular decision is crucial for building trust and enabling adoption, especially in critical domains [35, 8]. Explainable AI (XAI) aims to address this by developing techniques to explain the model's predictions. This section introduces attribution-based and concept-based methods for model explanation.

2.2.1 Attribution-based methods

Attribution-based explainability methods aim to understand model decisions by identifying which input features are most influential in driving a particular prediction. These techniques generate saliency maps or feature importance scores, highlighting the parts of the input that the model attends to when making a decision. Several approaches exist within this category. SHAP (SHapley Additive exPlanations) [36] leverages game theory to assign each feature an importance value for a particular prediction, considering interactions with other features. In image analysis, SHAP can be applied to superpixels, which are groups of pixels perceptually similar in color and texture, to provide explanations at a higher, more interpretable level than individual pixels. Another line of work focuses on more holistic approaches tailored for neural networks, which exploit the gradient information flow. A well-known example of such a method is Grad-CAM (Gradient-weighted Class Activation Mapping) [37], which uses gradient information flowing into the final convolutional layer to produce a coarse localization map highlighting important regions in the input image. Simpler gradient-based methods, like Vanilla Gradients [38], directly use the gradient of the output with respect to the input as a saliency map. SmoothGrad [39] addresses the noisy gradients often produced by Vanilla Gradients by averaging gradients over multiple noisy versions of the input, resulting in more visually coherent saliency maps. All of the above-mentioned methods utilize the backpropagation of gradients, but this approach has two major drawbacks. First, it does not explicitly provide information about the influence that a specific pixel had on the prediction but rather focuses on sensitivity analysis of network outputs with respect to its inputs. Second, it is a lossy approximation of the forward pass through the network.

In our work, however, we focus on Layer-wise Relevance Propagation (LRP) [40]. Unlike gradient-based methods, which often are heuristics, LRP offers a rigorous framework for decomposing predictions. It adheres to a relevance conservation principle, ensuring that the entire prediction score is attributed back to the input features through a layer-wise propagation process. Given a neuron's activation x_j and its relevance score R_j, LRP defines messages $R_{i\leftarrow j}$ that distribute this relevance to neurons i in the previous layer according to:

$$R_{i\leftarrow j} = \frac{z_{ij}}{z_j} R_j \tag{2.1}$$

where $z_{ij} = x_i w_{ij}$ represents the weighted activation flowing from neuron i to j, and $z_j = \sum_i z_{ij}$ is the total input received by neuron j. The relevance of a neuron i is then computed by pooling incoming messages:

$$R_i = \sum_j R_{i\leftarrow j} \tag{2.2}$$

To handle potential numerical instabilities when z_j approaches zero, a stabilized variant introduces a small positive term ϵ:

$$R_{i\leftarrow j} = \frac{z_{ij}}{z_j + \epsilon \cdot \text{sign}(z_j)} R_j \tag{2.3}$$

An alternative formulation separately treats positive and negative contributions:

$$R_{i\leftarrow j} = R_j \left(\alpha \frac{z_{ij}^+}{z_j^+} + \beta \frac{z_{ij}^-}{z_j^-} \right) \tag{2.4}$$

where $\alpha+\beta = 1$ and the superscripts indicate positive and negative parts, respectively. This formulation allows for explicit control over how positive and negative evidence is propagated through the network. The Deep Taylor Decomposition provides theoretical foundations for LRP by interpreting the relevance propagation rules as a first-order Taylor expansion of the relevance function

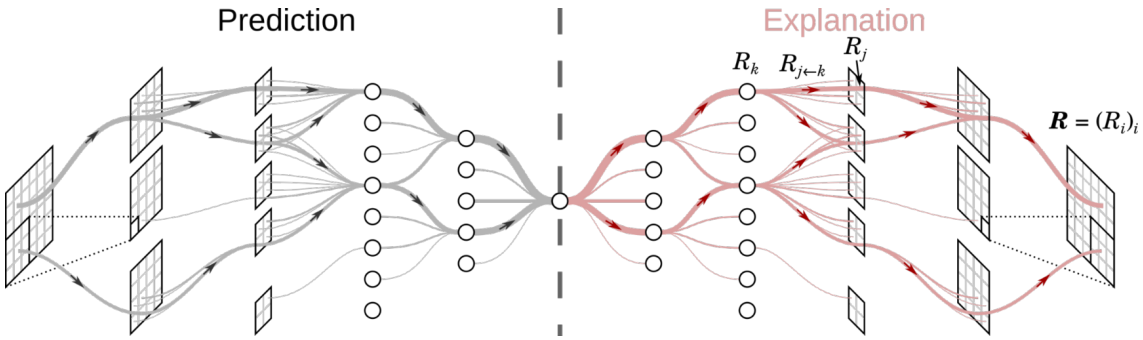


FIGURE 2.1: Layer-wise Relevance Propagation process. The relevance scores are propagated backwards through the network layers, starting from the output prediction score. The final result is a heatmap highlighting the input features most relevant to the network's decision. Image source: `https://www.hhi.fraunhofer.de/en/departments/ai/technologies-and-solutions/layer-wise-relevance-propagation.html`

around specific root points. For a given neuron with relevance function $R_j(x)$, the method identifies a root point $\tilde{x}$ where $R_j(\tilde{x}) = 0$, and approximates the relevance as:

$$R_j(x) \approx \sum_i \frac{\partial R_j}{\partial x_i}(\tilde{x})(x_i - \tilde{x}_i) \tag{2.5}$$

2.2.2 Concept-based methods

While attribution methods focus on input features, concept-based methods offer a complementary approach by explaining neural network decisions in terms of high-level concepts. These methods operate by examining the internal representations learned by the network. The core idea is to treat the network's internal activations as vectors in a high-dimensional space, where specific directions in this space correspond to meaningful concepts. By understanding how these concepts are activated and utilized, we can gain insights into the network's reasoning process at a more abstract level.

Vector space formulation

Given a neural network with inputs $x \in \mathbb{R}^n$ and a feed-forward layer l containing m neurons, we can define the layer's activation function as:

$$f_l : \mathbb{R}^n \to \mathbb{R}^m \tag{2.6}$$

This mapping transforms input data into the network's internal representation space. The activation values at layer l can be represented as a vector:

$$\mathbf{a}_l = f_l(x) \in \mathbb{R}^m \tag{2.7}$$

Concept Activation Vectors

To interpret this high-dimensional space in terms of human-understandable concepts, we utilized Concept Activation Vectors (CAVs) [41]. A CAV represents a direction (Figure 2.2) in the activation space that corresponds to a particular semantic concept (e.g., a concept can correspond to "curly hair", "blue eyes" etc.). For a concept C, we define its CAV $\mathbf{v}_C^l \in \mathbb{R}^m$ at layer l through supervised learning.

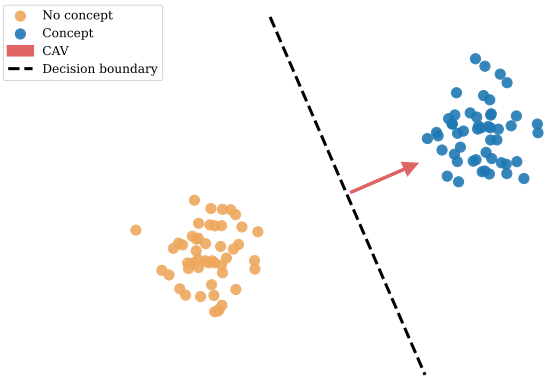


FIGURE 2.2: **CAV** is a unit vector showing the direction from activations without a given concept (**no concept**) to the activations containing it (**concept**).

Given a set of positive examples P_C representing the concept, a set of negative examples N, without the concept, and the activations of these examples: $\{f_l(x) : x \in P_C\}$ and $\{f_l(x) : x \in N\}$, we can compute the CAV through various methods.

Ridge Regression In Ridge Regression, we obtain the CAV by training a linear classifier in a supervised manner, having the activations as an input and concepts as labels.

$$\mathbf{v}_C^l = \arg\min_w \sum_{x \in P_C \cup N} (w^T f_l(x) - y)^2 + \alpha \|w\|^2 \tag{2.8}$$

where α is a regularization parameter, y is a binary label indicating concept presence, and w are the weights of a linear model.

Signal CAV In contrast to classifier-based CAVs that focus on separability, the signal CAV, introduced by Dreyer et al. [42] directly estimates concept direction from the data's covariance structure. This method posits that the concept direction is aligned with the direction of maximal covariance between layer activations and concept labels. The signal CAV is calculated as:

$$\mathbf{v}_C^l = \frac{\text{Cov}(f_l(x), y)}{\text{Var}(y)} \tag{2.9}$$

where $\text{Cov}(f_l(x), y)$ represents the covariance between the activations at layer l and the concept labels y, and $\text{Var}(y)$ is the variance of the concept labels.

Finally, the resulting CAV is normalized to unit length:

$$\mathbf{v}_C^l = \frac{\mathbf{v}_C^l}{\|\mathbf{v}_C^l\|_2} \tag{2.10}$$

2.3 Fairness

Fairness in ML is not just an ethical concern but also a legal obligation. Legal protections, such as those discussed by Wachter et al. [43], emphasize the need for algorithms to comply with non-discrimination regulations. Efforts to address the challenges of fairness have included the development of checklists for organizations to verify the ethical compliance of their technologies before deployment [44]. Similarly, trials to establish AI ethics guidelines, such as those discussed by Jobin et al. [45], aim to integrate fairness into the responsible development and deployment of

AI systems. However, achieving fairness is difficult, as relying on biased ground truth data often makes it impossible to construct classifiers that are truly fair [46].

Unfairness in ML systems is not only theoretical but has been observed in real-world applications. One prominent example is the COMPAS system, which we described in Section 1.1, which exhibited racial biases in its deployment within the U.S. judiciary system. Beyond COMPAS, numerous studies highlight biases in different domains and data modalities. For instance, Bolukbasi et al. [16] demonstrated stereotypical gender biases in word embeddings trained on the Google News corpus. Their findings revealed associations such as *"men"* being similar to terms like *"maestro"*, *"protege"*, or *"captain"*, while *"women"* were more closely associated with *"homemaker"* or *"receptionist"*. Similarly, Otterbacher et al. [47] showed that crowd-sourced image descriptions used in downstream ML tasks often exhibit cultural and regional biases, with annotators from the U.S. assigning more gender, race, and shape-related labels than annotators from India. They also found a systematic bias in labeling white males, as crowd-sourcers were omitting to assign labels to them, essentially making a white male a default category, a pattern not observed for other gender-race combinations. Furthermore, Barlas et al. [15] revealed that image tagging algorithms perform better when the depicted scenes align with stereotypical expectations, such as *"a woman in a kitchen"* versus *"a man in a kitchen"*.

Individual and Group Perspectives Fairness in machine learning can be categorized into two main paradigms: *individual fairness* and *group fairness* [19]. Each presents unique benefits and challenges. Individual fairness, as introduced by Dwork et al. [48], requires that similar individuals receive similar results. While intuitive, this notion faces practical challenges, such as defining and measuring individual similarity. If similarity can be reliably measured, it could theoretically be used to construct fair classifiers [13]. However, establishing an appropriate similarity metric often involves subjective judgments, which makes this notion complex to operationalize. Due to these issues, *group fairness* became a more extensively studied paradigm that focuses on ensuring that protected groups (e.g., based on race or gender) are treated equally well. Numerous definitions and metrics for group fairness exist in the literature [19]. Yet, despite its widespread use, group fairness is not without limitations. It is susceptible to aggregation biases, such as Simpson's paradox[1], and relies on the potentially controversial assignment of individuals to groups, often requiring sensitive data. Moreover, it struggles to address intersectional fairness issues, as highlighted by Buolamwini and Gebru [50], where specific attribute combinations, such as race and gender, can lead to significant disparities in outcomes (e.g., different rates of attaining positive outcomes).

The conflict between individual and group fairness is a fundamental philosophical and practical dilemma. For example, in graduate admissions, unequal base rates between groups may necessitate boosting scores for underrepresented groups to achieve an equitable outcome, effectively disadvantaging individuals from overrepresented groups. Although these two notions of fairness may seem orthogonal in practice [13], their apparent conflict is arguably the result of technical formalization and the inability to capture the unbiased ground truth [51].

Considering the above discussion, in our work we focus on group fairness.

2.3.1 Group fairness metrics

Group fairness aims to ensure equitable treatment and outcomes across different groups defined by sensitive attributes such as race, gender, or age. The literature presents various definitions of group

[1] Simpson's paradox is a phenomenon in probability and statistics in which a trend appears in several groups of data but disappears or reverses when the groups are combined [49].

fairness, as equality and equity can be interpreted in multiple ways [19]. While we recognize that these definitions and associated measures have different properties and capture various aspects of fairness [52], our focus in this work is limited to three specific measures. The intricate relationships between them are outside the scope of this study. Furthermore, as a design choice in DetoxAI, we allow users to select their preferred fairness metric, a practice advocated by researchers [13].

Thus, we focus on three selected definitions of fairness: *Equality of Odds (EO)*, *Demographic Parity (DP)*, and *Accuracy Parity (AP)*. To assess how well a model satisfies these definitions, it is necessary to quantify disparities between groups based on sensitive attributes. This is generally done by comparing outcomes between groups, either through differences (GAP) or ratios. In this work, we adopt the GAP approach due to its intuitive nature and common usage.

Before establishing group fairness definitions and metrics, let us briefly introduce prerequisite notions of measuring the correctness of classifier predictions. In Figure 2.3, we visualize the group-wise confusion matrix (groups are defined via some binary protected attribute A taking a and b values). This matrix aggregates the predictions made by a classifier on the input data and assigns them to four categories for each group separately. To simplify the definitions, we assume that A is a binary protected attribute taking on a and b values and we temporarily focus on the confusion matrix for the group a.

The *true positive rate* is defined as follows:

$$TPR = \frac{TP}{TP + FP} \tag{2.11}$$

it accounts for the proportion of the positive predictions that were correctly classified as positive.

The *false positive rate* is defined as follows:

$$FPR = \frac{FP}{FP + TN} \tag{2.12}$$

which captures the proportion of the falsely positive predictions as a proportion of all examples truly belonging to the negative class.

Having defined the prerequisite TPR and FPR, we can now proceed to define the notions of fairness we use in this work.

Definition 1 (Equality of Odds) *A predictive model satisfies* ***Equality of Odds*** *if the true positive rate (TPR) and false positive rate (FPR) are equal across all groups defined by a sensitive attribute.*

$$TPR_{A=a} = TPR_{A=b} \quad \wedge \quad FPR_{A=a} = FPR_{A=b}$$

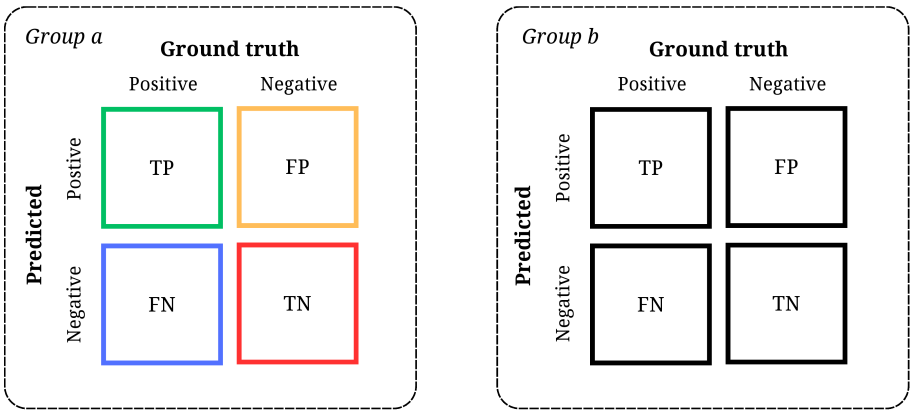


FIGURE 2.3: Group-wise confusion matrix.

Equality of Odds ensures that the model's performance is independent of the group membership. We measure Equality of Odds via the following metric:

$$EO\_GAP = \max\left(|TPR_{A=a} - TPR_{A=b}|, |FPR_{A=a} - FPR_{A=b}|\right) \tag{2.13}$$

Definition 2 (Demographic Parity) *A predictive model satisfies **Demographic Parity** if the predicted positive rate is equal across all groups defined by a sensitive attribute.*

$$P(\hat{Y} = 1 \mid A = a) = P(\hat{Y} = 1 \mid A = b)$$

This ensures that the likelihood of a positive prediction is independent of the membership of the group, regardless of the true labels. We measure Demographic Parity via the following metric:

$$DP\_GAP = |P(\hat{Y} = 1 \mid A = a) - P(\hat{Y} = 1 \mid A = b)| \tag{2.14}$$

Definition 3 (Accuracy Parity) *A predictive model satisfies **Accuracy Parity** if the overall accuracy is equal across all groups defined by a sensitive attribute.*

$$Accuracy_{A=a} = Accuracy_{A=b}$$

Accuracy is defined as the proportion of correctly predicted labels (see Equation (2.16)). Accuracy Parity ensures consistent overall performance across groups. We measure Accuracy Parity via the following metric:

$$AP\_GAP = |Accuracy_{A=a} - Accuracy_{A=b}| \tag{2.15}$$

2.3.2 Peformance-fairness trade-off

Various metrics focus on highlighting different aspects of predictive performance of a classifier in a given task. We adapt a number of them in this work. In the following, we briefly discuss a few common predictive performance metrics and briefly comment on their significance.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{2.16}$$

Accuracy measures the overall correctness of the classifier by considering the count of true positive and true negative predictions. It is perhaps the most widely used metric in machine learning, yet is often not sufficient to comprehensively assess classifier's predictive performance. Research shows that it is not well suited to classification tasks with class imbalance [53], especially when the imbalance is significant. As such, the assessment of model performance should consider a holistic, multifaceted look at classification outcomes.

$$Precision = \frac{TP}{TP + FP} = TPR \tag{2.17}$$

Precision measures the proportion of correctly predicted positive instances among all predicted positive instances. It highlights the reliability of positive predictions.

$$Recall = \frac{TP}{TP + FN} \tag{2.18}$$

Recall, also known as sensitivity, measures the proportion of correctly predicted positive instances among all actual positive instances. It emphasizes the ability of the model to identify positive cases.

$$Specificity = \frac{TN}{TN + FP} \tag{2.19}$$

Specificity measures the proportion of correctly predicted negative instances among all actual negative instances. It highlights the model's ability to identify negative cases.

$$F1 = \frac{2 \cdot Precision \cdot Recall}{Precision + Recall} \tag{2.20}$$

The F1 score balances precision and recall, providing a harmonic mean.

$$GMean = \sqrt{Recall \cdot Specificity} \tag{2.21}$$

The geometric mean ($GMean$) combines recall and specificity into a single metric, treating positive and negative classes equally. It is particularly useful when dealing with imbalanced datasets.

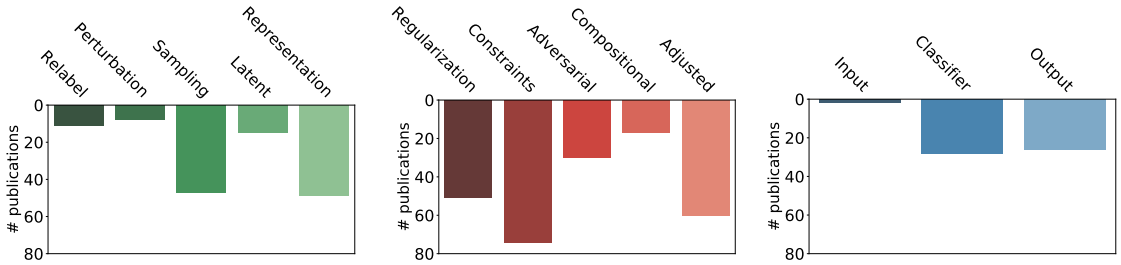
$$BA = \frac{Recall \cdot Specificity}{2} \tag{2.22}$$

Balanced Accuracy (BA) similarly to $GMean$ combines recall and specificity, but it halves their multiplication instead of taking the square root. It also is a common metric suited for imbalanced data.

In scenarios where base rates differ between groups, achieving fairness often results in a performance-fairness trade-off. For example, Demographic parity has been shown to trade-off with *Accuracy* [20, 54]. This trade-off highlights the inherent tension between optimizing for predictive performance and adhering to fairness constraints.

2.4 Debiasing techniques

Bias mitigation in machine learning models, particularly in the context of classification, has traditionally been addressed through a variety of techniques that can be broadly categorized into pre-processing, in-processing, and post-processing methods, as outlined by Friedler et al. [55]. These categories are based on the stage of the machine learning pipeline in which the intervention is applied.



(A) Pre-processing (B) In-processing (C) Post-processing

FIGURE 2.4: Popularity of the three main approaches to algorithmic bias mitigation in machine learning: pre-processing (data modification before training), in-processing (algorithmic adjustments during training), and post-processing (output corrections after training). Figure adapted from [56].

2.4.1 Pre-processing

Pre-processing covers methods that focus on modifying the training data before it is used to train a model. The goal is to transform the data in such a way that any inherent biases are removed or reduced. Common pre-processing techniques include:

- **Relabeling and Perturbation**: This involves modifying the labels or features of the training data to reduce bias. *Relabeling* alters the ground truth labels, as exemplified by "massaging" methods where, for instance, Kamiran and Calders [57] strategically relabeled instances near the decision boundary to minimize negative impacts on accuracy. *Perturbation* techniques adjust the features; for example, Feldman et al. [58] introduced methods to perturb features to make the distributions of different groups more similar, while preserving the rank order within groups, thus aiming to create a fairer dataset by modifying feature and label values.

- **Sampling**: This approach alters the distribution of samples in the training data to achieve fairness. Techniques include *oversampling*, such as SMOTE by Chawla et al. [59], which generates synthetic samples for minority groups to balance the dataset and mitigate bias arising from imbalanced representation.

- **Latent Variables**: This technique augments the training data with additional features, or latent variables, that are ideally unbiased, thereby enriching the dataset with fairness-promoting information (e.g., Calders and Verwer [60]).

- **Representation Learning**: The goal here is to learn a transformation of the training data into a new feature space that reduces bias while retaining relevant information. This often involves techniques like optimization, as demonstrated by Calmon et al. [61], adversarial learning, such as in the work by Edwards and Storkey [62], and variational autoencoders, exemplified by Louizos et al. [63], which aim to create a less biased data representation for subsequent model training.

2.4.2 In-processing

In-processing methods rely on modifying the learning algorithm itself to mitigate bias during the model training phase. These techniques often involve:

- **Regularization**: Adding a penalty term to the loss function that discourages the model from learning biased representations. The penalty term is designed to measure and reduce disparities in model performance across different groups. For instance, Kamishima et al. [64] introduced a prejudice remover regularizer to penalize discrimination.

- **Constraints**: Incorporating fairness constraints directly into the optimization problem during training (e.g., Zafar et al. [65]). These constraints can be based on various fairness metrics and aim to ensure that the model satisfies certain fairness criteria.

- **Adversarial Learning**: Training a classifier and an adversary simultaneously, where the adversary attempts to exploit fairness issues, and the classifier aims to minimize both classification error and the adversary's ability to predict sensitive attributes (e.g., Zhang et al. [66]).

- **Compositional Approaches**: Combining multiple classification models, each potentially trained on different subsets of the data or with different fairness objectives, to achieve a more balanced and fair outcome. An example of such compositional approach is Calders and Verwer [60]'s approach, which consists of three naive Bayes approaches for discrimination-free classification.

- **Adjusted Learning**: Modifying the learning procedure or creating novel algorithms specifically designed to address bias. This can involve techniques like adjusting decision thresholds or incorporating fairness considerations into the splitting criteria of decision trees. For instance, Aghaei et al. [67] learned optimal and fair decision trees for non-discriminative decision-making, showcasing an adjusted learning approach tailored for fairness.

2.4.3 Post-processing

Post-processing methods are applied after a model has been trained. They can be classified into the following paradigms.

- **Input Correction**: Modifying the features of the test-time input data to reduce bias, similar to perturbation techniques in pre-processing. For example, Adler et al. [68] proposed auditing black-box models for indirect influence, which can be seen as a form of input correction in a post-processing context when used to understand and adjust inputs.
- **Classifier Correction**: Directly altering the parameters or probabilistic predictions of a trained classifier to enhance fairness. This can involve techniques like adjusting decision thresholds or recalibrating the classifier output (e.g., [69, 70]).
- **Output Correction**: Modifying the predicted labels to improve fairness. This can involve techniques like relabeling, where predictions are adjusted based on proximity to the decision boundary or other criteria. An example of such approach is Kamiran and Calders [71]'s theory for discrimination-aware classification, which can be used in post-processing to adjust outputs for fairness.

2.5 Related Software

The need for fair machine learning solutions has sparked the development of various software toolkits designed to help practitioners build responsible AI systems. While commercial solutions exist, such as Accenture's `Responsible AI Solutions` and Meta's `Fairness Flow`, the open-source community has been particularly active in developing tools for bias detection and mitigation.

Our analysis of existing fairness software includes 11 toolkits, ranging from basic bias detection tools to more comprehensive solutions (see Table 2.2). Popular options like `What-If Tool` and `TF Fairness Indicators` focus solely on bias detection, while more advanced toolkits like `Fairlearn` and `AIF360` (both have more than 2,000 GitHub stars) incorporate detection and mitigation capabilities.

However, most current solutions face significant limitations. They predominantly rely on statistical metrics for bias detection, offering a narrow quantitative view of fairness. Few provide qualitative tools for the visualization of biases in models. Moreover, many lack bias mitigation techniques and those that implement them typically focus on tabular data, leaving other important data types, such as images and text, unaddressed. The key challenge lies in their technical architecture - the APIs usually expect entire datasets to be loaded into memory, which works well for moderate-sized tabular datasets but becomes impractical for larger datasets commonly used in deep learning. This architectural limitation creates a significant gap for modern machine learning practitioners. Most existing tools are built around `scikit-learn`'s API and expect `Pandas`'s dataframes as input, rather than supporting the batch data loading approaches used by deep learning frameworks like `PyTorch` and `TensorFlow`. As a result, practitioners working with deep learning models are often restricted to techniques that modify only model inputs or outputs rather

than addressing bias within the model's architecture or learning process itself. While this surface-level intervention can improve fairness metrics, it presents a critical challenge by leaving model biases intact. The model continues to leverage protected attributes that could emerge as features in its internal representations, even if these influences are not observed in the final outputs. This approach creates a disconnect between superficial fairness improvements and deeper algorithmic biases of deep learning models.

Toolkit	Capabilities	Latest Version	GitHub Stars*
`FairTest`	Detection	2020	56
`FAT Forensics`	Detection	2022	75
`Themis-ML`	Detection & Mitigation	2018	124
`REVISE`	Detection	2021	187
`Audit AI`	Detection	2019	313
`TF Fairness Indicators`	Detection	Active (v0.46.0)	345
`FairML`	Detection	2017	362
`Aequitas`	Detection	2023	703
`What-If Tool`	Detection	2023	930
`Fairlearn`	Detection & Mitigation	Active (v0.12.0)	over 2100
`AIF360`	Detection & Mitigation	Active (v0.5.0)	over 2300

Table 2.2: Current landscape of fairness toolkits. *As of January 15th 2025.

In the following, we provide a brief overview of the most sophisticated toolkits related to DetoxAI.

2.5.1 AIF360

`AIF360` stands out as one of the most widely adopted fairness toolkits in both academic and industrial settings. Released under the Apache v2.0 license, it represents a significant milestone in transitioning fairness research into practical tools for machine learning practitioners.

This toolkit implements techniques from eight published papers in the fairness research community. It offers a robust set of features, including over 71 statistical metrics for bias detection and a suite of bias mitigation methods spanning four pre-processing, seven in-processing, and four post-processing techniques. Additionally, it includes eight preloaded tabular datasets for experimentation and benchmarking.

Despite its comprehensive feature set, `AIF360` has notable constraints. As `AIF360` is recommended as an end-to-end solution, practitioners must redesign their pipelines to integrate with `AIF360`, which is often impractical. The toolkit enforces a structured workflow through its `StructuredDataset` class, which requires explicit definitions of features, labels, protected attributes, and their identifiers. This design ensures clarity and consistency but limits flexibility. Moreover, its bias mitigation techniques are tailored exclusively for structured data. For instance, applying `AIF360` to image data requires pre-processing steps to integrate with its internal `StructuredDataset` class. The preparation includes resizing, normalizing pixel values, and feature extraction. These manual adaptations undermine its usability for non-tabular modalities.

While `AIF360` has advanced the field of algorithmic fairness software, its design reflects a broader trend among fairness toolkits - primary support only for structured data. For practitioners working with complex, multimodal datasets, these limitations highlight the need for next-generation tools that seamlessly integrate with modern machine learning frameworks and address biases holistically.

2.5.2 Fairlearn

`Fairlearn` represents a significant milestone in the evolution of fairness tools. Originally launched by Microsoft Research in 2018, it has transformed into an open-source project under the MIT license. This transition from corporate research projects to community-driven initiatives highlights the growing democratization of fairness tools in machine learning.

The toolkit implements a few bias mitigation strategies across the taxonomy of methods described in Section 2.4. For pre-processing, `Fairlearn` offers correlation removal through linear transformation. This technique effectively reduces direct associations between features and sensitive attributes in structured data. However, its effectiveness diminishes when dealing with complex data types like images, where there is no direct relationship between pixels (features) and latent sensitive attributes. `Fairlearn`'s in-processing capabilities lie in two distinct approaches. The adversarial module implements *ZhangAL* adversarial debiasing technique. Complementing this, the reductions module takes a more mathematical approach, transforming fairness constraints into Lagrange multipliers. For post-processing, `Fairlearn` implements a threshold optimizer, based on work by Hardt et al. [69]. This tool fine-tunes classification thresholds for different groups to optimize fairness metrics. Under the hood, `Fairlearn`, heavily relies on `sklearn` library, which is quite unsuitable for deep learning. It lacks GPU support, and requires the inputs to be `NumPy` arrays instead of dataloaders, which is prohibitive for almost any image-based dataset. Moreover, all but one methods focus on adjusting model outputs that don't address deeper biases within the classifier itself.

Chapter 3

DetoxAI

In this chapter, we introduce our proposed solution DetoxAI, which is a package containing debiasing techniques and visualization tools useful for various post-hoc debiasing scenarios. This chapter is organized as follows. First, in Section 3.1 we introduce and describe nine implemented debiasing algorithms providing formal and intuitive definitions. Then, in Section 3.2 we present the visualization tools that are part of the package and that we also utilize later in experiments. Next, in Section 3.3 we outline the structure of the library and the organization of the toolkit, and in Section 3.4 we outline the dependencies of DetoxAI. Finally, in Section 3.5 we provide information about the online distribution of DetoxAI as a fully open-source software.

3.1 Methods

Our proposed solution comprises nine model debiasing techniques, all of which operate in a post-processing fashion. Figure 3.1 depicts the implementation structure of the methods, divided into specific modules. In the following subsections, we briefly describe each method, providing some background information, an intuitive description, as well as the algorithm's pseudocode.

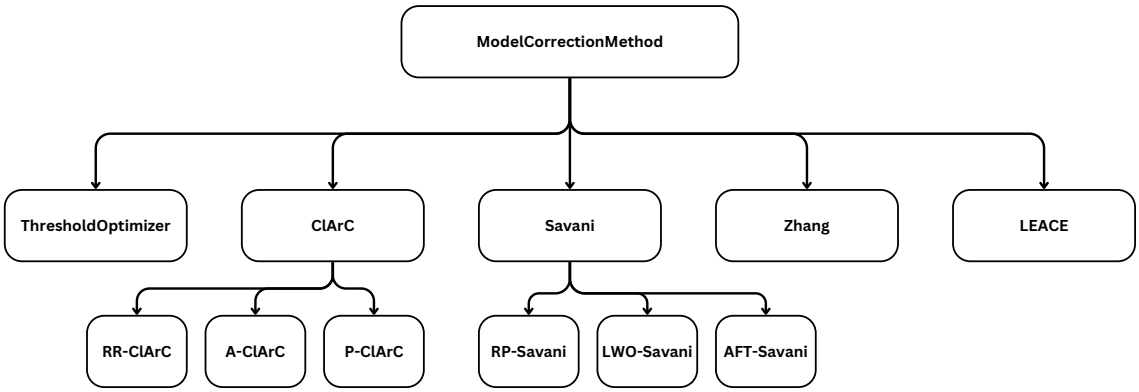


FIGURE 3.1: A diagram depicting the structural composition of implemented methods.

3.1.1 Savani's intra-processing

Savani et al. [70] proposed a novel framework for debiasing neural networks, which they termed *intra-processing*, as methods that intervened in model weights post-hoc were nonexistent at the time. Savani introduced an idea to leverage both in-processing and post-processing techniques by creating a method that bridges them. This approach fits within our taxonomy (Section 2.4) under post-processing model correction methods, though at the time of its introduction, it contradicted existing taxonomies. The proposed methodology employs two key concepts aimed at maximizing a constrained objective function that incorporates both predictive performance and fairness.

The first concept is debiasing as an intervention in the model, where the model parameters θ are perturbed or optimized to produce modified parameters θ'. The second concept involves threshold optimization, which applies a post-processing function $h : [0, 1] \rightarrow \{0, 1\}$ to map predicted probabilities to output classes using a selected threshold.

The problem was framed as a constrained optimization task for the model f, with the goal of minimizing bias, measured by a chosen fairness metric μ (e.g., *EO_GAP*), to fall below a specified threshold ϵ, while simultaneously maximizing predictive performance, as captured by a selected metric ρ (e.g., Balanced Accuracy). Specifically, for a given set of predictions $\hat{\mathcal{Y}}$ and ground truth outcomes $\mathcal{Y}$, paired with a corresponding set of protected attribute annotations $\mathcal{A}$, this optimization is formalized as $\phi_{\mu,\rho,\epsilon}$.

$$\phi_{\mu,\rho,\epsilon}(\mathcal{Y}, \hat{\mathcal{Y}}, \mathcal{A}) = \begin{cases} \rho(\mathcal{Y}, \hat{\mathcal{Y}}) & \text{if} \quad \mu(\mathcal{Y}, \hat{\mathcal{Y}}, \mathcal{A}) < \epsilon \\ 0 & \text{otherwise} \end{cases} \tag{3.1}$$

The authors proposed three variations of this general framework that utilize random perturbations, zeroth-order optimization, and first-order optimization schemes. Here, we briefly describe them formally and give an intuitive explanation of their inner workings.

Random Perturbation (*SavaniRP*) The first variation is an algorithm (pseudocode in Algorithm 1), which is a straightforward method to optimize $\phi_{\mu,\rho,\epsilon}$, which resembles a random local search. In essence, for every iteration, all model weights θ are perturbed by multiplying them with random Gaussian variables $Z \sim \mathcal{N}(1, 0.1)$, introducing noise into the weights. The updated model weights θ' are then used to generate new predictions $\hat{\mathcal{Y}}$. An optimal threshold τ is determined to map $\hat{\mathcal{Y}}$ to the final outputs $\ddot{\mathcal{Y}}$ using h_τ, a post-processing function designed to maximize $\phi_{\mu,\rho,\epsilon}$.

$$h_\tau : \hat{\mathcal{Y}} \rightarrow \ddot{\mathcal{Y}} = \begin{cases} 1 & \text{if} \quad \hat{\mathcal{Y}} > \tau \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

The random perturbation process is repeated for k iterations, and the best combination of θ and τ that maximizes $\phi_{\mu,\rho,\epsilon}$ across all iterations is returned.

Layer-Wise Optimization (*SavaniLWO*) The second variant introduced by the authors to be more effective than random perturbations is designed to be a zeroth-order optimization method, as it tries to selectively optimize model weights layer by layer to achieve better $\phi_{\mu,\rho,\epsilon}$. Specifically, it assumes that the model f is a composite function that can be decomposed into its constituent functions $f : f^L \circ f^{L-1} \circ ... \circ f^1$. This assumption is easily satisfied in neural networks, as each layer and component can be treated as a separate function in the chain. After decomposition, the algorithm uses model-based zeroth-order optimization techniques (e.g., gradient-boosted regression trees [72]) where it optimizes the weights in a given layer (and freezes all others) with respect to the objective $\phi_{\mu,\rho,\epsilon}$. However, this is impractical, as zeroth-order optimization in each layer separately

Algorithm 1 *SavaniRP*

Input:
Model f with weights θ
Validation dataset $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{A}$
Objective $\phi_{\mu,\rho,\epsilon}$ ▷ Set bias limit ϵ, bias metric μ and performance metric ρ
Number of iterations k
Initialize:
$\theta^* \leftarrow \theta$, $\phi^* \leftarrow -\infty$, $\tau^* \leftarrow 0$
for $i = 1$ to k **do**
  Sample perturbations $Z \sim^N \mathcal{N}(1, 0.1)$ ▷ Where N is the number of parameters in θ
  Update weights $\theta' \leftarrow \theta \cdot Z$ ▷ Element-wise multiplication
  Obtain predictions using new model $\ddot{\mathcal{Y}} \leftarrow f_{\theta'}(\mathcal{X})$
  for $\tau \in [0, 1]$ **do** ▷ Grid search
    Apply new mapping $\ddot{\mathcal{Y}} \leftarrow h_\tau(\hat{\mathcal{Y}})$
    Compute objective function value $\phi \leftarrow \phi_{\mu,\rho,\epsilon}(\mathcal{Y}, \ddot{\mathcal{Y}}, \mathcal{A})$
    if $\phi > \phi^*$ **then** ▷ If ϕ improved, save best configuration
      Update $\phi^* \leftarrow \phi$, $\theta^* \leftarrow \theta'$, $\tau^* \leftarrow \tau$
    end if
  end for
end for
return θ^*, τ^* ▷ Return θ and τ maximizing ϕ

is a costly procedure. Thus, the authors recommend optimizing only a few layers in the back of the model, as they typically have fewer parameters and operate on already extracted latent features. In Algorithm 2 we present the pseudo-code for this method. For simplicity, we assume that all L layers are to be optimized, but trivially the algorithm could be tweaked to optimize only a selected set of layers.

Algorithm 2 *SavaniLWO*

Input:
Model $f = f^L \circ \cdots \circ f^1$ with weights $\theta = \{\theta^L, \ldots, \theta^1\}$
Validation dataset $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{A}$
Objective $\phi_{\mu,\rho,\epsilon}$ ▷ Set bias limit ϵ, bias metric μ, and performance metric ρ
Black-box model-based optimizer $\mathcal{B}$
Initialize:
$\theta^* \leftarrow \emptyset$, $\phi^* \leftarrow -\infty$, $\tau^* \leftarrow 0$
for $l = 1$ to L **do** ▷ Iterate over all layers
  Optimize θ^l using $\mathcal{B}$: $\theta^{l\prime} \leftarrow \mathcal{B}(\theta^l, \phi_{\mu,\rho,\epsilon})$
  Update weights: $\theta' \leftarrow \{\theta^1, \ldots, \theta^{l\prime}, \ldots, \theta^L\}$
  Obtain predictions using the updated model: $\ddot{\mathcal{Y}} \leftarrow f_{\theta'}(\mathcal{X})$
  for $\tau \in [0, 1]$ **do** ▷ Grid search over threshold values
    Apply new mapping: $\ddot{\mathcal{Y}} \leftarrow h_\tau(\hat{\mathcal{Y}})$
    Compute the objective function value: $\phi \leftarrow \phi_{\mu,\rho,\epsilon}(\mathcal{Y}, \ddot{\mathcal{Y}}, \mathcal{A})$
    if $\phi > \phi^*$ **then** ▷ If ϕ improves, save best configuration
      Update $\phi^* \leftarrow \phi$, $\theta^* \leftarrow \theta'$, $\tau^* \leftarrow \tau$
    end if
  end for
end for
return θ^*, τ^* ▷ Return θ and τ maximizing ϕ

Adversarial Fine Tuning (*SavaniAFT*) The third and final variant proposed by Savani leverages first-order optimization through adversarial learning, making it distinct from the previous two methods that relied on zeroth-order optimization. The key innovation is the introduction of a critic

model g that learns to predict the bias in a minibatch of data. Note that it is not possible to optimize bias directly with a first-order method as the objective is non-differentiable. By using bootstrapping through random sampling with replacement to create minibatches, the critic can effectively serve as a differentiable proxy for the typically non-differentiable fairness metrics. The algorithm alternates between two phases:

Phase 1: training the critic model g using predictions from the original model f,

Phase 2: fine-tuning f using a custom differentiable objective function that approximates $\phi_{\mu,\rho,\epsilon}$ while incorporating the bias predictions $\hat{\mu}$ from g.

This custom function is defined as $\max\{1, \lambda(|\hat{\mu}| - \epsilon + \delta) + 1\} \cdot \mathcal{L}(y, \hat{y})$, where λ and δ are hyperparameters controlling, respectively, the strictness of the bias constraint and the desired error margin, and $\mathcal{L}$ is a loss function such as binary cross-entropy. This custom loss function effectively scales the task-specific loss based on the predicted bias, maintaining the original loss when the absolute bias is below $\epsilon - \delta$ and increasing it proportionally otherwise. The pseudocode for this method is presented in Algorithm 3.

Algorithm 3 *SavaniAFT*

Input:
Model f with weights θ
Validation dataset $\mathcal{X}$, $\mathcal{Y}$, $\mathcal{A}$
Critic model g with weights $\theta^{(g)}$
Hyperparameters λ, ϵ, δ
Number of outer iterations n
Number of critic training iterations m
Number of model training iterations m'
Loss function $\mathcal{L}$ ▷ For example binary cross-entropy

for $i = 1$ to n **do**
  for $j = 1$ to m **do** ▷ Train critic model
    Sample minibatch $(\mathcal{X}_j, \mathcal{Y}_j, \mathcal{A}_j)$ with replacement
    Compute true bias $\hat{\mu} \leftarrow \mu(\mathcal{Y}_j, f(\mathcal{X}_j), \mathcal{A}_j)$
    Update $\theta^{(g)}$ using gradient $\nabla_{\theta^{(g)}}(\hat{\mu} - g(f(\mathcal{X}_j)))^2$
  end for
  for $j = 1$ to m' **do** ▷ Fine-tune original model
    Sample minibatch $(\mathcal{X}_j, \mathcal{Y}_j)$ with replacement
    Update θ using gradient $\nabla_\theta[\max\{1, \lambda(|g(f(\mathcal{X}_j))| - \epsilon + \delta) + 1\} \cdot \mathcal{L}(\mathcal{Y}_j, f(\mathcal{X}_j))]$
  end for
  Select optimal threshold $\tau \in [0, 1]$ maximizing $\phi_{\mu,\rho,\epsilon}$
    ▷ In the same way as in *SavaniRP*and *SavaniLWO*
end for
return θ, τ ▷ Return θ and τ

Implementation details In the text above, we presented three algorithms along with their pseudocodes. To meet the specific requirements of this approach, we adapted certain components to ensure compatibility with DetoxAI.

First, since Savani's methods require threshold optimization, we implemented a custom intervention, a *hook* in the model which realizes the thresholding mechanism. This approach allows DetoxAI to return the debiased model with the hook applied without requiring the model owner to modify their existing pipeline. The hook is placed within the computational graph after the model's last layer, transforming the input (either logits or probabilities) into final predictions by applying the desired threshold.

To support DetoxAI's goal of debiasing models across various scenarios and model architectures, we had to introduce an additional mechanism facilitating the use of any backbone. Typically,

model outputs are either logits or probabilities, depending on the design, and are used directly in tasks based on the usage scenario. For instance, in binary classification, the output layer design can vary. A developer may opt for a single-output layer with logits passed through a sigmoid activation to produce probabilities or a two-output layer with softmax activation. Savani's original approach assumes single-output layers with sigmoid activation, enabling direct application of h_τ for thresholding. However, this assumption does not hold for architectures with a softmax-activated output, where the outputs are coupled and sum to 1.

To address this limitation and make Savani's approach compatible with different architectures, we implemented a custom forward hook for soft thresholding. This enhancement supports both single-output (sigmoid) and two-output (softmax) configurations. The solution involves a differentiable approximation of the thresholding function. For models with logit outputs, we first apply softmax (or sigmoid) to obtain probabilities, then perform soft thresholding using a sigmoid function on the scaled difference between the positive class probability and the threshold τ:

Two-output variant, assuming index 0 is the negative and index 1 is the positive class:

$$\hat{y}_1 = \begin{cases} \sigma((\text{softmax}(z)_1 - \tau) \cdot \alpha) & \text{if logits} \\ \sigma((\hat{y} - \tau) \cdot \alpha) & \text{if probabilities} \end{cases} \tag{3.3}$$

$$\hat{y}_0 = 1 - \hat{y}_1 \tag{3.4}$$

Single-output variant:

$$\hat{y} = \begin{cases} \sigma(\sigma(z) - \tau) \cdot \alpha) & \text{if logits} \\ \sigma((\hat{y} - \tau) \cdot \alpha) & \text{if probabilities} \end{cases} \tag{3.5}$$

Here, z represents the output logits, $\hat{y}$ is the direct probability output, σ is the sigmoid function, and α (usually set to a large value) controls the sharpness of the threshold approximation (temperature). For two-dimensional outputs, the negative class probability is computed as the complement of the positive one.

This soft thresholding approach is critical as it preserves gradient flow during training while approximating the behavior of h_τ. It is also essential for explanation techniques employed in our experiments, which require differentiable models.

3.1.2 Zhang's adversarial learning

Zhang et al. [66] proposed an adversarial debiasing framework that is closely related to *SavaniAFT* discussed in the previous section. While originally designed as an in-processing method, we follow Savani's adaptation and implement it as an intra-processing approach, allowing for both model fine-tuning and output adjustments.

The key innovation in *ZhangAL*'s approach lies in its formulation of the debiasing problem as a minimax optimization game between two neural networks: a predictor trying to accurately model the target variable $\mathcal{Y}$ given input $\mathcal{X}$, and an adversary attempting to predict the protected attribute $\mathcal{A}$ from the predictor's output. Unlike *SavaniAFT*, which uses a custom differentiable objective incorporating bias predictions, *ZhangAL* method employs a more direct adversarial objective where the adversary predicts whether the model prediction was based on an input containing a protected attribute. The overall adversary optimization objective can be formulated as follows:

$$\min_{\theta} \max_{\theta(g)} \mathcal{L}(\hat{\mathcal{Y}}, \mathcal{Y}) - \alpha \mathcal{L}_{adv}(g_{\theta(g)}(\hat{\mathcal{Y}}), \mathcal{A}) \tag{3.6}$$

for a predictor f with parameters θ and an adversary g with parameters $\theta^{(g)}$. where $\mathcal{L}$ is the predictor's loss (e.g., binary cross-entropy), $\mathcal{L}_{adv}$ is the adversary's loss, and α is a hyperparameter controlling the trade-off between prediction accuracy and fairness. The negative sign before α indicates that the predictor tries to maximize the adversary's loss while the adversary tries to minimize it.

However, in an attempt to counter a possible collapse in optimization, where the predictor is helping the adversary, *ZhangAL*'s parameter update rule also encompasses a projection term. It is designed to ensure that the predictor will never help the adversary by updating its parameters in a way that helps them. Therefore, the final parameter update rule in the optimization of the predictor is performed using the f' s and g' s gradients as well as their projection and is formulated as follows:

$$\nabla_\theta \mathcal{L}(\hat{\mathcal{Y}}, \mathcal{Y}) - \text{proj}_{\nabla_\theta \mathcal{L}_{adv}} \nabla_\theta \mathcal{L} - \alpha \nabla_\theta \mathcal{L}_{adv}(g_{\theta^{(g)}}(\hat{\mathcal{Y}}), \mathcal{A}) \tag{3.7}$$

where the projection operation is performed according to:

$$\text{proj}_{\nabla_\theta \mathcal{L}_{adv}} \nabla_\theta \mathcal{L} = \nabla_\theta \mathcal{L} \times \nabla_\theta \mathcal{L}_{adv} - \nabla_\theta \mathcal{L}_{adv} \tag{3.8}$$

To be compatible with different fairness definitions, Zhang's framework can be modified by adjusting what information is provided to the adversary.

- For demographic parity, the adversary receives only the predictor's output $\hat{\mathcal{Y}}$
- For equality of odds, the adversary receives both $\hat{\mathcal{Y}}$ and the true label $\mathcal{Y}$

The training procedure alternates between updating the adversary and the predictor, as shown in Algorithm 4. During each iteration, the adversary is first trained to predict Z, followed by an update to the predictor that attempts to minimize its task loss and maximize the adversary's prediction error.

Algorithm 4 *ZhangAL*

Input:
Training data $(\mathcal{X}, \mathcal{Y}, \mathcal{A})$
Model f with θ weights and adversary g with $\theta^{(g)}$
Trade-off parameter α
Number of training epochs T
Number of adversary steps per epoch K
Number of fine-tuning steps per epoch J
Loss functions $\mathcal{L}$ and $\mathcal{L}_{adv}$

for $t = 1$ to T **do**
  Get predictions $\hat{\mathcal{Y}} \leftarrow f_\theta(\mathcal{X})$
  for $k = 1$ to K **do** ▷ Train adversary
    Update $\theta^{(g)}$ to minimize $\mathcal{L}_{adv}(g(\hat{\mathcal{Y}}), \mathcal{A})$
  end for
  for $j = 1$ to J **do** ▷ Fine-tune the original model
    Update θ according to Equation (3.7) update rule
  end for
end for
return f_θ

A practical consideration in implementing Zhang's method is the architecture of the adversary network. While the predictor's architecture is typically determined by the task requirements, the adversary can be designed specifically for the debiasing objective. For binary protected attributes,

a simple feed-forward neural network with a single hidden layer often suffices, as the adversary's task is fundamentally about detecting potentially subtle statistical patterns in the predictor's output that correlate with the protected attribute.

3.1.3 Class artifact compensation

Anders et al. [73] introduced Class Artifact Compensation (*ClArC*), a family of methods designed to remove the influence of specific concepts—represented via Concept Activation Vectors (CAVs)—in deep neural networks. Unlike adversarial debiasing approaches that modify training objectives, *ClArC* operates directly on the model's internal representations through geometric interventions in activation space.

The core of *ClArC* lies in its use of CAVs to identify artifact directions in neural network activations. These vectors represent directions in activation space correlated with specific protected attributes. By projecting activations away from these directions or modifying gradient updates, *ClArC* variants aim to produce models invariant to the targeted artifacts.

Projective *ClArC*

The projective variant (*P-ClArC*) applies an orthogonal transformation to layer activations, aligning them with artifact-free patterns. As shown in Algorithm 5, this is achieved through a projection operation followed by activation adjustment relative to mean non-artifactual patterns:

$$\Pi = \frac{\mathbf{a}_l \cdot \mathbf{h}_l}{\|\mathbf{h}_l\|^2} \mathbf{h}_l \tag{3.9}$$

where $\mathbf{h}_l$ is the CAV for layer l and $\mathbf{a}_l$ represents layer activations. The modified activations $\mathbf{a}'_l$ are computed as:

$$\mathbf{a}'_l = \mathbf{a}_l - (\Pi - \bar{\mathbf{a}}_{l,na}) \tag{3.10}$$

with $\bar{\mathbf{a}}_{l,na}$ denoting mean non-artifactual activations. This transformation is applied during inference via forward hooks without modifying model weights.

Algorithm 5 Projective *ClArC* (*P-ClArC*)

Input:
Model f with weights θ
Input data $\mathcal{X}$
Concept Activation Vector $\mathbf{h}_l$ for layer l
Mean non-artifactual activation $\bar{\mathbf{a}}_{l,na}$
Initialize:
Register forward hook on layer l
for each input **x do**
    Compute original activations: $\mathbf{a}_l \leftarrow f_l(\mathbf{x})$
    Project onto CAV: $\Pi \leftarrow \frac{\mathbf{a}_l \cdot \mathbf{h}_l}{\|\mathbf{h}_l\|^2} \mathbf{h}_l$
    Modify activations: $\mathbf{a}'_l \leftarrow \mathbf{a}_l - (\Pi - \bar{\mathbf{a}}_{l,na})$
    Complete forward pass with modified activations
end for
return Modified predictions

Augmentative *ClArC*

The augmentative variant (*A-ClArC*) extends *P-ClArC* by introducing the projection mechanism during fine-tuning. As described in Algorithm 6, this approach modifies activations while updating

model weights through standard backpropagation. The key difference in formula from *P-ClArC* is in its use of mean artifactual activation $\bar{\mathbf{a}}_{l,a}$ for adjustment:

$$\mathbf{a}'_l = \mathbf{a}_l - (\Pi - \bar{\mathbf{a}}_{l,a}) \tag{3.11}$$

This enables the model to learn representations that avoid encoding the targeted artifact while maintaining task performance.

Algorithm 6 Augmentative *ClArC* (*A-ClArC*)

Input:
Model f with weights θ
Training dataset $\mathcal{X}$, $\mathcal{Y}$
Concept Activation Vector $\mathbf{h}_l$ for layer l
Mean artifactual activation $\bar{\mathbf{a}}_{l,a}$
Initialize:
Register forward hook on layer l
for each batch $(\mathbf{x}, y)$ in training data **do**
    Compute original activations: $\mathbf{a}_l \leftarrow f_l(\mathbf{x})$
    Project onto CAV: $\Pi \leftarrow \frac{\mathbf{a}_l \cdot \mathbf{h}_l}{\|\mathbf{h}_l\|^2} \mathbf{h}_l$
    Modify activations: $\mathbf{a}'_l \leftarrow \mathbf{a}_l - (\Pi - \bar{\mathbf{a}}_{l,a})$
    Forward pass with modified activations
    Update weights using standard optimization
end for
return Updated model weights θ

Right Reasons *ClArC*

The Right Reasons variant (*RR-ClArC*) [42] operates in gradient space rather than activation space. As presented in Algorithm 7, it penalizes gradient components aligned with artifact directions during fine-tuning. The regularization loss $\mathcal{L}_{RR}$ comes in three forms:

- L2 penalty: $(\mathbf{g}^\top \mathbf{h}_l)^2$
- L1 penalty: $|\mathbf{g}^\top \mathbf{h}_l|$
- Cosine similarity: $|\text{cosine\_similarity}(\mathbf{g}, \mathbf{h}_l)|$

where $\mathbf{g}$ represents gradients at layer l. The total loss combines task loss with artifact alignment penalty:

$$\mathcal{L} = \mathcal{L}_{CE} + \lambda_{RR} \mathcal{L}_{RR} \tag{3.12}$$

Implementation Details All *ClArC* variants employ PyTorch's hook mechanism for activation modification. For convolutional layers with activations $\mathbf{a}^{\text{conv}}(\mathbf{x}) \in \mathbb{R}^{m \times w \times h}$, we apply spatial max-pooling to obtain per-filter activations:

$$\mathbf{a}_i(\mathbf{x}) = \max_{v,q} \mathbf{a}^{\text{conv}}_{i,v,q}(\mathbf{x}) \tag{3.13}$$

Algorithm 7 Right Reasons *ClArC* (*RR-ClArC*)

Input:
Model f with weights θ
Training dataset $\mathcal{X}$, $\mathcal{Y}$
Concept Activation Vector $\mathbf{h}_l$ for layer l
Regularization strength λ_{RR}
RR loss type (L2, L1, or Cosine)
Initialize:
Register forward hook on layer l
for each batch $(\mathbf{x}, y)$ in training data **do**
  Forward pass: $\hat{\mathcal{Y}} \leftarrow f(\mathbf{x})$
  Compute cross-entropy loss: $\mathcal{L}_{CE} \leftarrow \text{CE}(\hat{\mathcal{Y}}, y)$
  Compute masked criterion: $\mathcal{M} \leftarrow \text{mask}(\hat{\mathcal{Y}}, y)$
  Calculate gradients: $\mathbf{g} \leftarrow \nabla_{\mathbf{a}_l} \mathcal{M}$
  Compute RR loss based on type:
  if L2 loss **then**
    $\mathcal{L}_{RR} \leftarrow (\mathbf{g}^\top \mathbf{h}_l)^2$
  else if L1 loss **then**
    $\mathcal{L}_{RR} \leftarrow |\mathbf{g}^\top \mathbf{h}_l|$
  else
    $\mathcal{L}_{RR} \leftarrow |\text{cosine\_similarity}(\mathbf{g}, \mathbf{h}_l)|$
  end if
  Total loss: $\mathcal{L} \leftarrow \mathcal{L}_{CE} + \lambda_{RR}\mathcal{L}_{RR}$
  Update weights using combined loss
end for
return Updated model weights θ

3.1.4 Least-squares concept erasure

LEACE [74] introduces a closed-form transformation approach that optimally removes concept information from neural network representations while minimizing changes to the activation space. The method operates by computing an affine transformation that prevents linear classifiers from detecting target concepts, while maintaining the network's fundamental processing capabilities. By leveraging whitening transformations and orthogonal projections, *LEACE* provides an optimal solution with respect to all positive semi-definite norms simultaneously. Procedure is shown in Algorithm 8.

Implementation Details *LEACE* is implemented using PyTorch's forward hook mechanism to modify layer activations during inference. For convolutional layers with activations $\mathbf{a}^{\text{conv}}(\mathbf{x}) \in \mathbb{R}^{m \times w \times h}$, the transformation is applied to the flattened representation:

$$\mathbf{a}_{\text{flat}}(\mathbf{x}) = \text{reshape}(\mathbf{a}^{\text{conv}}(\mathbf{x}), (m, w \cdot h)) \tag{3.14}$$

The implementation maintains computational efficiency through:

- Batched processing of activations during transformation
- Caching of computed statistics and transformation matrices
- In-place modification of activations during the forward pass
- Efficient handling of high-dimensional convolutional features

The method supports both regular PyTorch modules and Lightning modules, with flexible configuration options for layer selection and concept specification. The transformation parameters

are computed once during initialization and stored for repeated use during inference, ensuring computational efficiency during deployment.

Algorithm 8 Least-squares Concept Erasure (*LEACE*)

Input:
Model f with weights θ
Input data $\mathcal{X}$
Concept labels $\mathcal{A}$
Target layer l
Initialize:
Register forward hook on layer l
Compute activation statistics:
  $\mathbb{E}[\mathbf{X}] \leftarrow \text{mean}(\mathcal{X})$
  $\mathbf{\Sigma_{XX}} \leftarrow \text{cov}(\mathcal{X})$
  $\mathbf{\Sigma_{XZ}} \leftarrow \text{cross\_cov}(\mathcal{X}, \mathcal{A})$
Compute whitening transform: $\mathbf{W} \leftarrow (\mathbf{\Sigma}_{\mathbf{XX}}^{1/2})^{+}$
Compute projection: $\mathbf{P_{WXZ}} \leftarrow (\mathbf{W\Sigma_{XZ}})(\mathbf{W\Sigma_{XZ}})^{+}$
for each input **x do**
  Compute original activations: $\mathbf{a}_l \leftarrow f_l(\mathbf{x})$
  Center activations: $\mathbf{a}_c \leftarrow \mathbf{a}_l - \mathbb{E}[\mathbf{X}]$
  Transform: $\mathbf{a}'_l \leftarrow \mathbf{a}_l - \mathbf{W}^{+}\mathbf{P_{WXZ}}\mathbf{W}\mathbf{a}_c$
  Complete forward pass with modified activations
end for
return Modified predictions

3.1.5 Threshold optimization

Naive Threshold Optimization (*NT*) technique [69], shown in Algorithm 9, introduces a post-processing approach that achieves fairness by optimizing the classification threshold. The method operates by modifying the model's decision boundaries without altering its internal representations or requiring retraining. By searching over threshold values that minimize specified fairness gap, *NT* provides a computationally efficient solution for improving fairness binary classification tasks where only black-box access to the model is available.

Algorithm 9 Naive Threshold Optimization (NT) with EO_GAP

Input:
  Model f
  Validation data: features $\mathcal{X}$, labels $\mathcal{Y}$, protected attributes $\mathcal{A}$
  Fairness metric: *EO_GAP*
  Threshold range: $[t_{\min}, t_{\max}]$
Output:
  Best threshold t^*
Initialize: Best fairness gap $m^* \leftarrow \infty$, Best threshold $t^* \leftarrow t_{\max}$
Compute probabilistic predictions: $\hat{\mathcal{Y}} \leftarrow f(\mathcal{X})$
for each threshold t in range $[t_{\min}, t_{\max}]$ **do**
  Compute binary predictions: $\hat{\mathcal{Y}} \leftarrow (\hat{\mathcal{Y}} > t)$
  Compute fairness metric: $g \leftarrow \mathit{EO\_GAP}(\hat{\mathcal{Y}}, \mathcal{Y}, \mathcal{A})$
  if $g < m^*$ **then**
    Update best threshold: $t^* \leftarrow t$
    Update best fairness gap: $m^* \leftarrow g$
  end if
end for
return Best threshold t^*

Implementation Details The method supports three primary fairness metrics for optimization: *EO_GAP*, *DP_GAP*, *AP_GAP*. The implementation is optimized through vectorized operations for efficient threshold and archiving of intermediate computations for fairness metrics. An additional feature designed to make *NT* more flexible is the support for custom metric definitions via lambda functions.

The implementation of thresholding of the model outputs is the same as in Section 3.1.1.

3.2 Visualizations

In machine learning, a crucial part of any model development effort is the validation phase. To streamline the process of evaluating models before and after the corrections introduced by DetoxAI, we have natively included visualization techniques as part of the package. Specifically, apart from providing tools for quantitative analysis with fairness and performance metrics, we provide a set of visualizing techniques to qualitatively assess the relevance attributed to parts of images by the model in its decision-making process. This section briefly introduces these visualization tools, which are later utilized in experiments to evaluate several research questions.

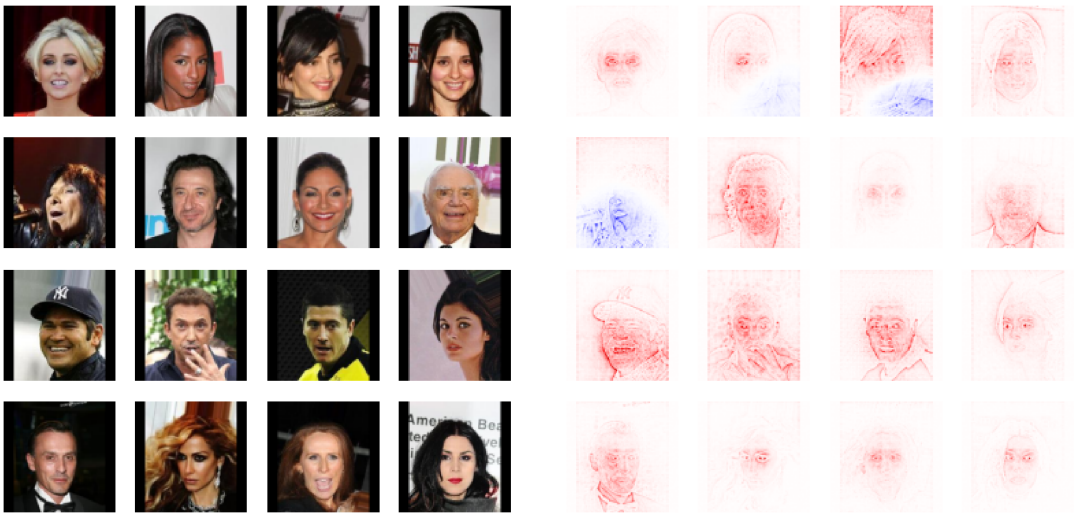


FIGURE 3.2: Image and heatmap-based pixel-wise relevance visualizations.

3.2.1 Instance-level visualization

The first type of visualization is per-sample plot. These plots visualize input images along with prediction relevance in the form of a heatmap with the same dimensionality as input space. In simpler words, we provide visualization where each pixel from the input image gets a relevance score, how much it contributes to the final prediction. To generate these relevance maps, we utilize the LRP method introduced by Bach et al. [40] and described in Section 2.2. As a result, DetoxAI provides visualizations such as the one shown in Figure 3.2. In these maps, pixels highlighted in stronger red indicate a more positive relevance to the predicted class, while pixels in deeper blue signify a more negative contribution to the predicted class.

These visualizations are particularly useful for manually evaluating whether the model's relevance aligns with the appropriate parts of the image. This type of debugging is a common practice and has recently gained attention in the literature [42], as it can highlight specific visual fea-

tures considered during predictions that should not have been used (e.g., a face recognition model focusing on trees in the background of an image).

3.2.2 Aggregate-level visualization

The next type of visualizations natively supported by DetoxAI are aggregate-level visualizations. These show the average image for each combination of the protected attribute and the target class, along with the average pixel-wise contribution. An example of such a visualization is shown in Figure 3.3. The relevancy maps and images are averaged across a batch, split into rows by gender (female, male) and columns by the attribute "smiling" (yes, no).

In this example, it can be observed that the prediction's relevancy in the upper row is concentrated on the eyes of the person, while in the bottom row it is more evenly distributed across the image. This suggests that, for the model for which we ran this visualization when the person in the image is a woman, the eyes play a significant role in the prediction, whereas this pattern is not observed for men. There could be several reasons for this behavior, which are explored further in the experiments and analysis in Section 5.3.2.

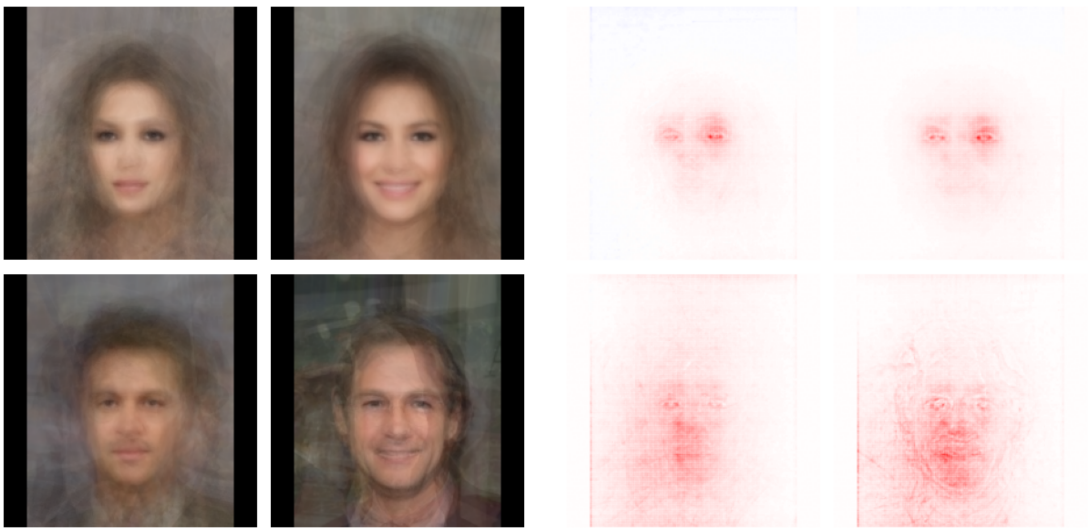


FIGURE 3.3: Aggregate-level visualizations of relevancy maps and average images. Rows correspond to gender (female, male), while columns represent the attribute "smiling" (yes, no).

3.2.3 Performance-fairness trade-off visualization

The final type of visualization is quantitative, designed to illustrate the performance-fairness trade-off between the methods applied by DetoxAI and the original (*Vanilla*) model. An example is shown in Figure 3.4. The rows correspond to performance metrics (in this case *GMean* and *F1*), while the columns represent fairness metrics (in this case *EO_GAP*, *DP_GAP*, and *AP_GAP*).

This visualization enables practitioners to assess the performance-fairness trade-off and determine which version of the model is most suitable for their specific application scenario.

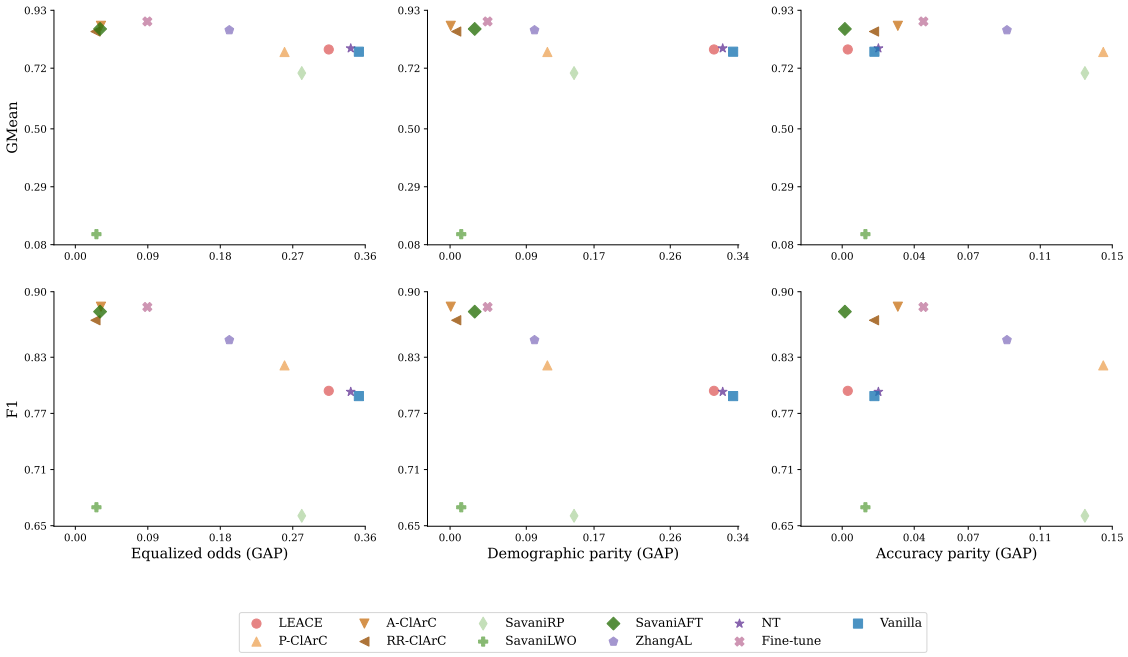


FIGURE 3.4: Performance-fairness trade-off visualization comparing DetoxAI and the Vanilla model. Rows represent performance metrics, while columns show fairness metrics.

3.3 Library structure

The source code of the DetoxAI library resides in the `src` folder. This structure ensures the library remains isolated from Python invocations in the project root and keeps distributed library code well-organized and distinct from other project components. The `src` folder contains seven primary modules (Figure 3.5): `cavs`, `core`, `datasets`, `methods`, `metrics`, `utils` and `viz`. Each module is designed to serve a specific purpose, ensuring modularity and ease of use.

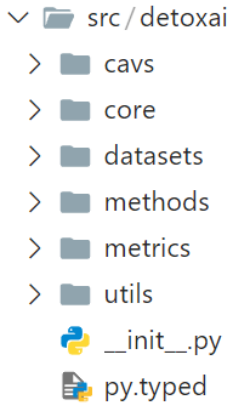


FIGURE 3.5: DetoxAI package source folder

3.3.1 Core module

The `core` module provides the primary Application Programming Interface (API) available to users. As the name suggests, it encapsulates the core logic of the library, offering a unified interface for all functionalities. Additionally, it includes functions for model evaluation, tools for selecting the most suitable bias mitigation method based on specific objectives, and PyTorch Lightning wrappers for user-defined models. This module also defines default configurations for hyperparameters

associated with each bias mitigation technique, enabling users to adopt recommended settings effortlessly.

3.3.2 CAVs module

The `cavs` module handles the computation of Concept Activation Vectors (CAVs). To ensure flexibility, it supports six different methods for CAV computation, including Support Vector Machines (SVM), Ridge Regression, Lasso, Logistic Regression, and a signal-CAV approach inspired by Dreyer et al. [42] Furthermore, this module provides custom functionality for extracting activations from neural networks, ensuring compatibility with various architectures and use cases.

3.3.3 Datasets module

The `datasets` module manages external data handling. DetoxAI includes several pre-integrated datasets, such as CelebA and FairFace for facial attributes (e.g., gender, race, age) and general datasets like CIFAR-10, CIFAR-100, and Caltech101. This module facilitates downloading and caching datasets system-wide in a user-specified folder or the default home directory. This caching mechanism prevents redundant downloads across different applications or repositories.

Each dataset is structured uniformly, comprising the following components: `comment.txt` contains source details, creator information, acknowledgments, and citation guidelines. `handler.py` includes dataset-specific instructions for unpacking and pre-processing. `links.yaml` lists source URLs for dataset archives. Users can extend the dataset catalog by providing their own `handler.py` files filled with dataset pre-processing code.

The module also supports dataset rebalancing based on user-defined constraints. For instance, users can enforce correlations such as ensuring images of males always include hats, while at least 50% of female images feature earrings. The rebalanced datasets, referred to as *variants*, are saved by storing the sampled image IDs in the cache, enabling system-wide reuse under user-defined names. The detailed procedure is described in Section 5.1.

3.3.4 Methods module

The `methods` module implements all bias mitigation techniques described in Section 3.1. Each technique is a subclass of `ModelCorrectionMethod`, ensuring a unified interface across methods. This design facilitates seamless integration with the rest of the library.

3.3.5 Metrics module

The `metrics` module provides implementations for evaluating fairness and bias. To minimize dependencies, the metrics are implemented in-house rather than relying on external libraries. The module includes versions of the metrics in `PyTorch` for computational flexibility. Supported fairness metrics include *Equal Opportunity*, *Equalized Odds*, *Demographic Parity*, *Accuracy Parity*. To enhance stability, a small epsilon is added to all operations that include division to avoid division by zero. Additionally, the module includes distance metrics such as cosine similarity, maximum value distance, and Euclidean distance.

3.3.6 Utils module

The `utils` module contains various helper classes and utilities that can be used independently of the bias mitigation pipeline. Key components include:

- `DetoxaiDataloader`: Extends `PyTorch`'s `DataLoader` with features such as retrieving specific batches.
- `DetoxaiDataset`: Supports custom collate functions and adheres to the dataset structure guidelines outlined in the `datasets` module.
- Image transformations: Includes auxiliary transformations like square padding, which are not available in standard libraries like `torchvision`.

3.3.7 Visualization module

The visualization module implements the visualizations described in Section 3.2. All visualizations inherit from a `Visualizer` interface and implement visualizations based on the `matplotlib` library.

3.4 Dependencies

We selected `Python` as the primary programming language for DetoxAI because of its popularity in the machine learning community and its extensive ecosystem of tools and libraries. Below, we list the Python packages that DetoxAI relies on:

3.4.1 Core dependencies

In designing DetoxAI, one of our key objectives was to minimize dependencies to ensure that our library is lightweight and maintainable. Below, we detail the key dependencies and the rationale behind their inclusion; a full list of dependencies and their required specific versions can be found in Appendix B.

PyTorch is the core deep learning framework for DetoxAI, selected for its dominance in the `Python` ecosystem and active community behind it. Its flexibility makes it suitable for both research and production environments. To enhance `PyTorch`'s capabilities for computer vision tasks, `torchvision` package was incorporated, providing pre-trained model backbones and their weights, as well as tools for image data handling. Additionally, `PyTorch Lightning` is utilized to streamline training and evaluation processes, abstracting repetitive boilerplate code. This design decision allowed us to focus on implementing bias mitigation techniques rather than managing training infrastructure.

NumPy and Pandas are integral to efficient data handling and pre-processing in DetoxAI. `NumPy` enables high-performance operations on multidimensional arrays and matrices, leveraging low-level optimizations for computational speed. `Pandas` complements this by offering powerful tools for manipulating structured data, including reading, transforming, and organizing attributes and labels from various file formats. These libraries are particularly leveraged in the datasets module to prepare data for model training and evaluation pipelines.

Pillow (PIL) is a modern successor of the Python Imaging Library (PIL). Thanks to this package, we can efficiently load and preprocess image data rather than store it in RAM as tensors during training and debiasing pipelines.

matplotlib is a visualization library that provides flexible tools for creating plots.

3.4.2 Additional dependencies

To meet specific requirements, DetoxAI incorporates several specialized libraries within its modules. `scikit-learn` facilitates the training of auxiliary models, such as those used to compute Concept Activation Vectors (CAVs), a core feature of the `cavs` module. Built on `scikit-learn`, `scikit-optimize` supports model-based optimization used by *SavaniLWO* debiasing method. `LeaceEraser` library implements the *LEACE* debiasing method. `Zennit` provides Layer-wise Relevance Propagation (LRP) implementations for heatmap visualizations.

3.5 Distribution

Python Package Index (PyPI) is the standard repository for sharing and distributing Python packages, making it easy to find and install community-developed software. We decided to publish DetoxAI as a PyPI package[1] to allow quick and seamless installation for developers. The package can be installed via a standard PyPi interface:

```
python -m pip install detoxai
```

Together with the PyPI package, we make the implementation of DetoxAI open-source on GitHub[2] with the CC-BY-4.0 license (see Figure 3.6).

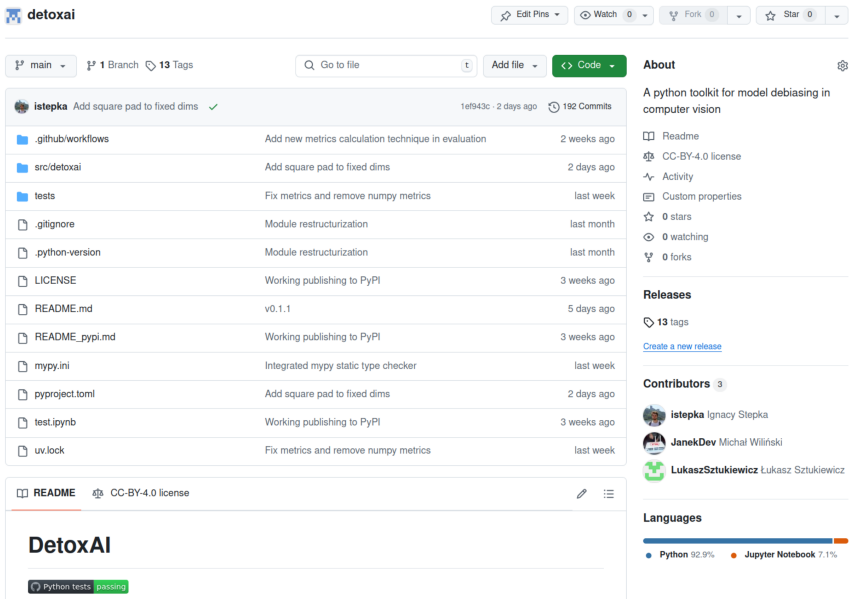


FIGURE 3.6: The main page of DetoxAI's GitHub repository.

We provide a brief introduction to the main interface of the package in Appendix A.

[1] `https://pypi.org/project/detoxai`
[2] `https://github.com/DetoxAI/detoxai`

Chapter 4

Technical Framework

4.1 Codebase

4.1.1 Code repositories

To ensure modularity and streamlined collaboration, we maintained three separate GitHub repositories during the development of DetoxAI. The first repository was dedicated to experiments, containing code for running experiments, configuring methods, and setting up datasets and specific experimental scenarios. This repository acted as a sandbox for testing hypotheses and experimenting with hyperparameters. The second repository focused on package development, validation, and visualization. It included tools and scripts necessary for internal testing and refinement that were not intended for the final open-source release. Finally, the open-source repository hosted the core DetoxAI library. This repository adhered to strict versioning practices using Semantic Versioning[1] and employed a continuous integration and deployment (CI/CD) pipeline for publishing releases to PyPI. To integrate the repositories, we organized them in a hierarchy and linked them with Git submodules. This approach ensured that each repository could function independently while remaining interconnected.

4.1.2 Package management

Cross-platform compatibility was a critical consideration during the development process, as the team worked on different operating systems: Linux (Ignacy), Windows (Łukasz), and macOS (Michał). To achieve uniformity in dependency management across these platforms, we adopted the `uv` package manager[2]. This tool provided a robust mechanism for resolving and locking package versions through the `uv.lock` file. By using this system, we ensured that all dependencies were consistent across Linux, Windows, and macOS environments, eliminating potential platform-specific discrepancies.

4.1.3 Ensuring code quality

Maintaining high code quality was a priority to avoid technical debt and facilitate long-term maintenance of DetoxAI. While good architectural design played a significant role, we also leveraged modern tools to enforce best practices. A key challenge was ensuring stylistic consistency among developers, as each team member had their own preferred coding style. To address this, we adopted a unified coding style that adheres to the recommended Python's PEP 8[3].

[1] `https://semver.org`
[2] `https://docs.astral.sh/uv`
[3] `https://peps.python.org/pep-0008`

To automate style enforcement and detect potential issues, we used `ruff`[4], a fast and comprehensive tool that served as both a linter and a formatter. Additionally, we integrated `pre-commit`[5] hooks into our workflow, ensuring that all code changes were automatically formatted and linted by `ruff` before being committed to the repository.

4.1.4 Version control

Version control was integral to our development process, enabling collaborative and parallel contributions. We used Git, a widely adopted distributed version control system, to manage changes to the codebase. By hosting a centralized Git repository on GitHub, we facilitated a visual pull request workflow that streamlined code reviews and merging. This workflow minimized conflicts when multiple developers worked on the same files, ensuring a smooth and efficient development cycle. The use of Git also provided a reliable mechanism for tracking changes, maintaining a history of revisions, and managing branches for new features or experimental developments.

4.2 Experimental framework

4.2.1 Reproducibility and experiment tracking

To ensure that our experiments were reproducible on any machine, we adopted `ClearML`[6], an open-source experiment tracking tool. `ClearML` provides centralized management of experiments, including features for reproducing and comparing experiments within the web UI. Moreover, it also serves as a centralized server storing all artifacts generated during the experiments. Our efforts to achieve full reproducibility and invariance to host machines enabled us to run experiments on multiple platforms in parallel. This was facilitated by `ClearML` agents, a lightweight service that proactively monitors specific experiment queues. When an experiment is queued, an agent pulls the code to run it and replicates the exact environment, including all dependencies, as defined by the master machine that initiated the experiment.

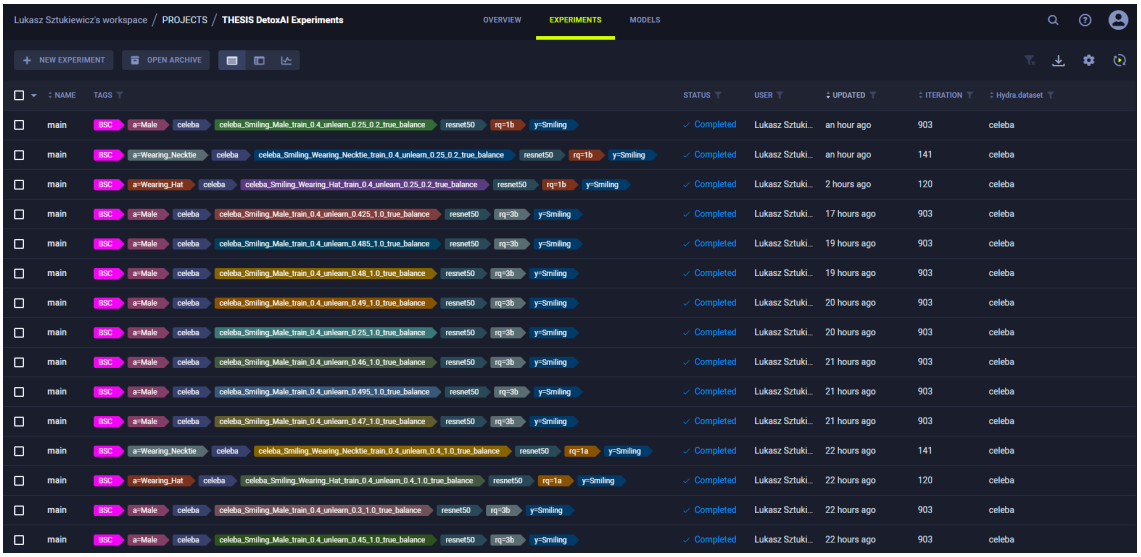


FIGURE 4.1: A screenshot from `ClearML` central experiment repository.

[4] `https://github.com/astral-sh/ruff`
[5] `https://pre-commit.com`
[6] `https://github.com/clearml/clearml`

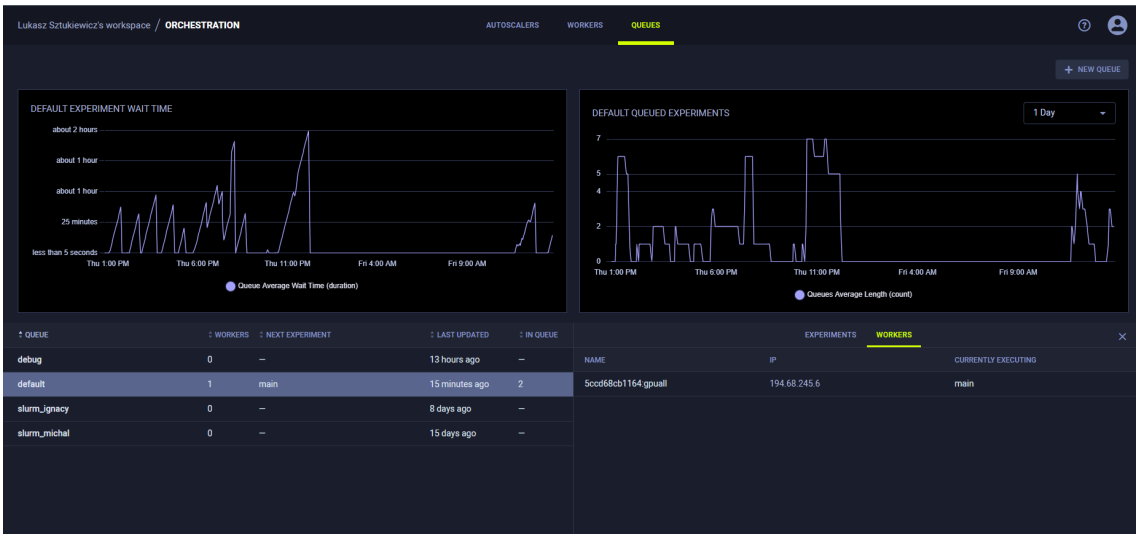


FIGURE 4.2: A screenshot from queued experiments in `ClearML`.

4.2.2 Logging and analysis

Each experiment's outputs, including plots and dataframes, were registered using `TensorboardX`, a version of `TensorBoard` compatible with `PyTorch`. These artifacts were subsequently logged to `ClearML`, enabling seamless analysis on both local machines and the centralized `ClearML` repository. This workflow streamlined the process of reviewing the outcomes of the experiments and ensured that all results were securely stored and accessible.

4.2.3 Hyperparameter management

To follow best practices in experiment management, we extracted all hyperparameters into separate configuration files. As the complexity of our experiments grew, we adopted `Hydra`, a production-ready configuration management tool maintained by Meta[7]. `Hydra` enables the dynamic creation of hierarchical configurations through composition and allows overrides via configuration files or the command line. It also supports running multiple similar jobs with different configurations.

Integrating `Hydra` with `ClearML` required some engineering work; however, the effort was well justified. The combination of `Hydra` and `ClearML` provided robust experiment and configuration management, allowing us to reproduce any experiment within seconds, even from mobile devices.

4.2.4 Custom command-line interface

To further streamline our workflow, we developed a custom command-line interface (CLI) that encapsulated the functionalities of `ClearML` and `Hydra`. This unified interface allowed for efficient creation, duplication, execution and deletion of experiments. As the codebase expanded and basic commands became more complex, this tool significantly improved productivity. The CLI was implemented in `Python` to ensure compatibility and ease of integration with our existing infrastructure.

4.2.5 Scaling resources for large-scale experiments

Initially, we conducted experiments on personal computers, which sufficed for toy datasets. However, as we transitioned to large-scale datasets, the need for greater GPU power became apparent.

[7] `https://github.com/facebookresearch/hydra`

To address this, we rented GPUs from three vendors: Lambda Cloud[8], RunPod[9], and TensorDock[10]. These platforms offered cost-effective GPUs, such as the RTX 2000 Ada and RTX 4000 Ada, with substantial VRAM and CUDA capabilities.

Subsequently, thanks to support from our supervisor, we gained access to a university platform equipped with A100 GPUs featuring 40GB of VRAM. This infrastructure provided the computational power necessary to efficiently run benchmarks and large-scale experiments that we describe in Section 5.3.

4.3 Validation

Ensuring code reliability and correctness was a crucial aspect of the DetoxAI development process. We implemented a testing strategy that covered multiple layers of validation, from unit tests to integration tests.

Unit testing

Our testing framework primarily utilized Python's built-in `pytest` module. The test suite included validation of core functionalities, particularly focusing on metrics calculations and data handling. The tests covered various scenarios, including edge cases and perfect fairness conditions.

Layer-wise relevance propagation as a validation

We utilized LRP relevance maps to sanity-check the validity of biases that are learned during the training process. For each experiment, we verify that the model learned specific bias through visual inspection of relevance maps. This assured us of the correct introduction of a specific bias into the network's inference process.

Continuous integration

We implemented automated testing through GitHub Actions, ensuring that all tests were run automatically on each push and pull request. The CI pipeline uses `uv` package manager for dependency management and runs the entire test suite to prevent accidental introduction of breaking changes.

[8] `https://lambdalabs.com`
[9] `https://runpod.io`
[10] `https://tensordock.com`

Chapter 5

Experiments

In this chapter, we evaluate the effectiveness of our proposed package, DetoxAI, in mitigating biases. We begin by presenting the dataset used in the experiments in Section 5.1, along with its variants and the enforcement of protected attribute-target correlation. Next, in Section 5.3, we empirically analyze the results of applying the DetoxAI package from various perspectives, addressing five distinct research questions.

5.1 Data

5.1.1 Dataset

The Large-scale CelebFaces Attributes (CelebA) dataset [75] is a well-known benchmark dataset in the field of computer vision. This dataset was originally designed for tasks such as face detection, face landmark localization, and face attribute recognition. For our purposes, we have simply used it for a conventional binary image classification task.

The CelebA dataset consists of over 200,000 celebrity images, each annotated with 40 binary attributes. These attributes range from physical characteristics, such as hair color and facial features, to the presence of accessories, such as glasses or hats. This wealth of annotated attributes makes the dataset highly flexible for a variety of experiments. In our study, we focused on attributes relevant to fairness. Among the available attributes, *"Gender"* stood out as the one most directly associated with discrimination risks in real-world applications. Therefore, it served as the protected attribute in most of our experiments. However, fairness can be defined also in terms of other attributes. For example, we explored scenarios where attributes such as *"Wearing Hat"* were treated as protected. This choice reflects concerns about potential indirect bias, as head coverings can be culturally or religiously significant, making them a potential proxy for sensitive demographic characteristics.

To ensure the robustness of the experiments, we divided the dataset into three distinct subsets: a training set, a test set, and a debiasing set. The training set was used to fine-tune pre-trained backbones obtained from the `torchvision` package. The test set was used to evaluate the performance of the model, and the dedicated debiasing set was only used by the DetoxAI methods to compute Concept Activation Vectors (CAVs) or for fine-tuning; the DetoxAI methods did not have access to the training or test sets. The percentage splits for these subsets were kept constant across experiments, as shown in Table 5.1. We took care to avoid any data leakage between subsets to ensure the validity of our results.

A key step in data preparation was the selection of target and protected attributes. Since our task was binary image classification, the target attribute had to be a single label with two possible

values. The CelebA dataset does not come with predefined classes, but provides several attributes, so we had the flexibility to define our target class from these. For this study, we chose the attribute *"Smiling"* as the target. This attribute indicates whether the person in the image is smiling (class 1) or not (class 0). The base model, *Vanilla*, was fine-tuned on this dataset to classify whether the person in the image is smiling.

Train subset	Test subset	Debiasing subset
30%	50%	20%

TABLE 5.1: Dataset splits

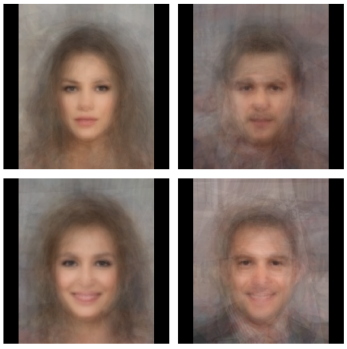


FIGURE 5.1: Averaged images for a variant where the target attribute is *"Smiling"* and the protected attribute is *"Male"*.

FIGURE 5.2: Averaged images for a variant where the target attribute is *"Smiling"* and the protected attribute is *"Wearing Hat"*.

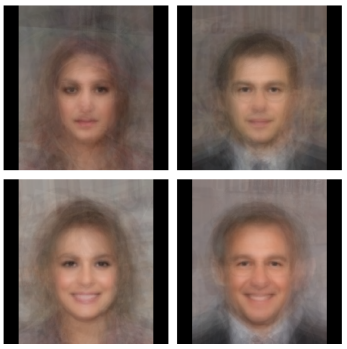


FIGURE 5.3: Averaged images for a variant where the target attribute is *"Wearing Necktie"* and the protected attribute is *"Smiling"*.

5.1.2 Correlation between attributes

The CelebA dataset naturally exhibits correlations between attributes, which we quantify using Yule's correlation coefficient (ϕ)[1]. For instance, *"Heavy Makeup"* attribute is strongly correlated to *"Wearing Lipstick"* ($\phi = 0.8$), as lipstick is usually a component of makeup. Similarly, *"Smiling"* shows moderate correlations with *"High Cheekbones"* ($\phi = 0.65$) and *"Mouth Slightly Open"* ($\phi = 0.57$), reflecting natural associations between facial expressions and features. However, some correlations appear more questionable, revealing inherent biases in the data labeling process. For instance, *"Attractive"* and *"Gender"* exhibit a negative correlation ($\phi = -0.4$), indicating that women are generally rated as more attractive than men. Furthermore, datasets reflect imbalances that occur in the real world; for instance, attributes like *"Blond Hair"* and *"Wavy Hair"* negatively correlate with *"Gender"*, so are more commonly associated with women.

To better control these biases in our experiments, we artificially enforced specific correlations between attributes by rebalancing the dataset. This rebalancing was performed separately for training, testing, and debiasing sets. For instance, to establish a correlation $\phi = 0.6$ between *"Gender"* and *"Smiling"* in the training set, we undersampled specific attribute combinations, resulting in the distribution shown in Table 5.2. Yule's correlation coefficient was computed using the formula in Equation 5.1:

$$\phi = \sqrt{\frac{\chi^2}{n}} = \frac{n_{11}n_{00} - n_{10}n_{01}}{\sqrt{(n_{11} + n_{10})(n_{11} + n_{01})(n_{00} + n_{10})(n_{00} + n_{01})}} \tag{5.1}$$

Here, n_{ij} represents the percentage of samples in a combination of the i-th attribute with the j-th label. Intuitively, in a perfectly balanced dataset, each of the $2^2 = 4$ label combinations would

[1]Note that all attributes in CelebA are binary, thus Yule's correlation coefficient is a suitable measure in this context and, in fact, equivalent to Pearson's r.

comprise 25% of the dataset, corresponding to $\phi = 0$. By adjusting the distribution of the label combinations, we enforced the desired correlation.

	Smiling	**Not smiling**
Female	0.4	0.1
Male	0.1	0.4

TABLE 5.2: Attribute combination percentages enforcing $\phi = 0.6$ correlation between *"Gender"* and *"Smiling"* in the training set.

	Smiling	**Not smiling**
Female	0.25	0.25
Male	0.25	0.25

TABLE 5.3: Attribute combination percentages enforcing $\phi = 0$ correlation between *"Gender"* and *"Smiling"* in the test set.

5.2 Hyperparamers

In this section, we outline the key hyperparameter settings employed in our experiments. For a comprehensive list of hyperparameters refer to Appendix E.

To ensure numerical precision during training, we explicitly set the floating-point matrix multiplication precision in `torch` to *high*. This choice ensured accurate computations throughout the training process. Furthermore, to guarantee reproducibility, all experiments were initialized with a fixed random seed of 42.

In terms of model architecture, we utilized pre-trained backbones available in `torchvision`, leveraging pre-trained weights from `IMAGENET1K_V1`. For most experiments we employed ResNet-50 as the base model, a widely used architecture comprising 25.6 million parameters. All methods in `detoxai` requiring model intervention operated on the final feed-forward layer of each backbone. The Concept Activation Vector (CAV) type used was consistently the signal CAV. Activations were extracted once per experiment and cached to minimize computational overhead during the debiasing process. For optimization, we adopted the Adam optimizer due to its adaptability and reliable performance in deep learning tasks. The learning rate was set to 3×10^{-4}. To further stabilize training, we fixed the batch size at 128 and accelerated data loading by configuring `torch` parameters: `num_workers` was set to 6, and the `prefetch_factor` to 3.

5.3 Experimental studies

In the following experiments, the objective is to answer several research questions to validate the usefulness and effectiveness of DetoxAI. First, in Section 5.3.1, we verify whether DetoxAI can effectively remove biases and improve predictive performance. Then, in Section 5.3.2, we explore DetoxAI's impact on feature relevance in the model's decision-making process. Next, in Section 5.3.3, we examine how the relationship between the target and the protected attribute in the dataset influences DetoxAI's ability to mitigate bias. After that, in Section 5.3.4, we analyze whether the size of the underlying model affects DetoxAI's ability to mitigate bias. Finally, in Section 5.3.5, we investigate whether the underlying model architecture impacts DetoxAI's performance.

5.3.1 Relationship between the presence of bias and model's performance

RQ1: Can DetoxAI be used effectively to remove biases and improve predictive performance?

To address this question, we performed experiments with all DetoxAI's debiasing methods to mitigate three different biases on resampled variants of CelebA dataset. The results presented in Figures 5.4 to 5.6 indicate that **DetoxAI is capable of mitigating biases and, in some cases, also improving predictive performance compared to a biased model - *Vanilla*.**

DetoxAI methods generally shift the performance-fairness trade-off curve favorably. In most of the cases, DetoxAI achieves comparable or improved performance with reduced bias compared to the *Vanilla* model. This is visualized by DetoxAI methods generally positioning closer to the ideal point of low bias and high performance in the plots. Another observation is that methods effective in reducing Equalized Odds (*EO_GAP*) and Demographic Parity (*DP_GAP*) gaps can sometimes lead to an increase in the Accuracy Parity (*AP_GAP*) gap. This highlights the inherent complexities in simultaneously optimizing for different fairness definitions, and the importance of choosing debiasing methods based on the specific fairness goals of the application [13, 20].

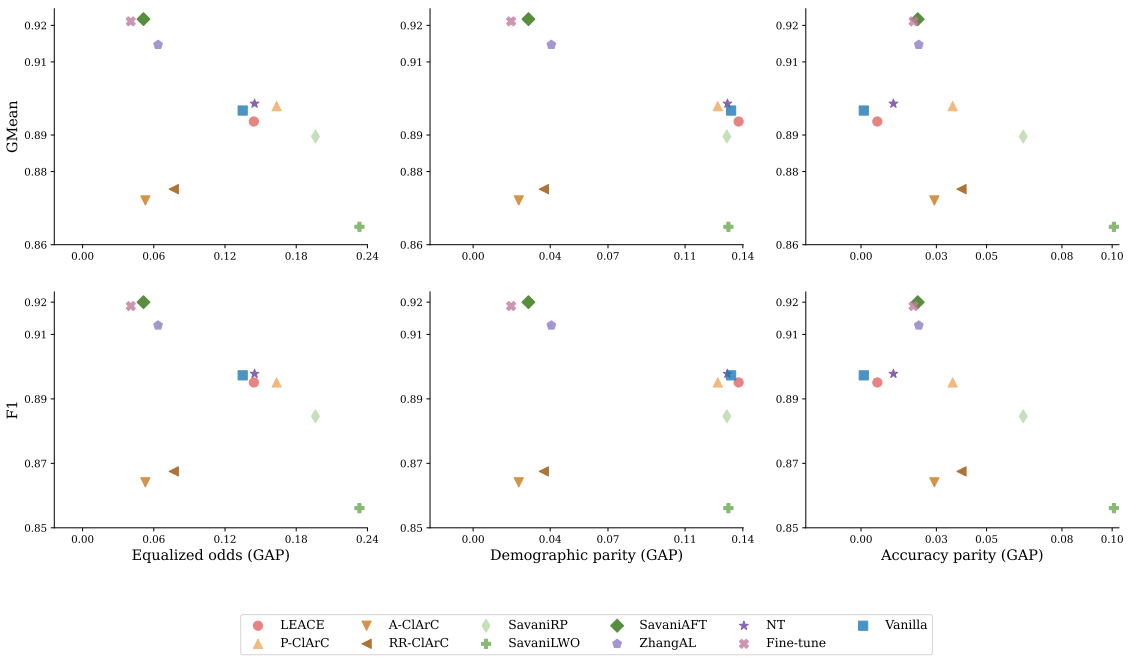


FIGURE 5.4: Performance-fairness trade-off for the "Smiling-Male" dataset variant, comparing DetoxAI methods performance and fairness metrics. Each subplot visualizes the trade-off between a performance metric on the y-axis and a fairness metric on the x-axis. The ideal point is (0, 1) indicating both high performance and low bias.

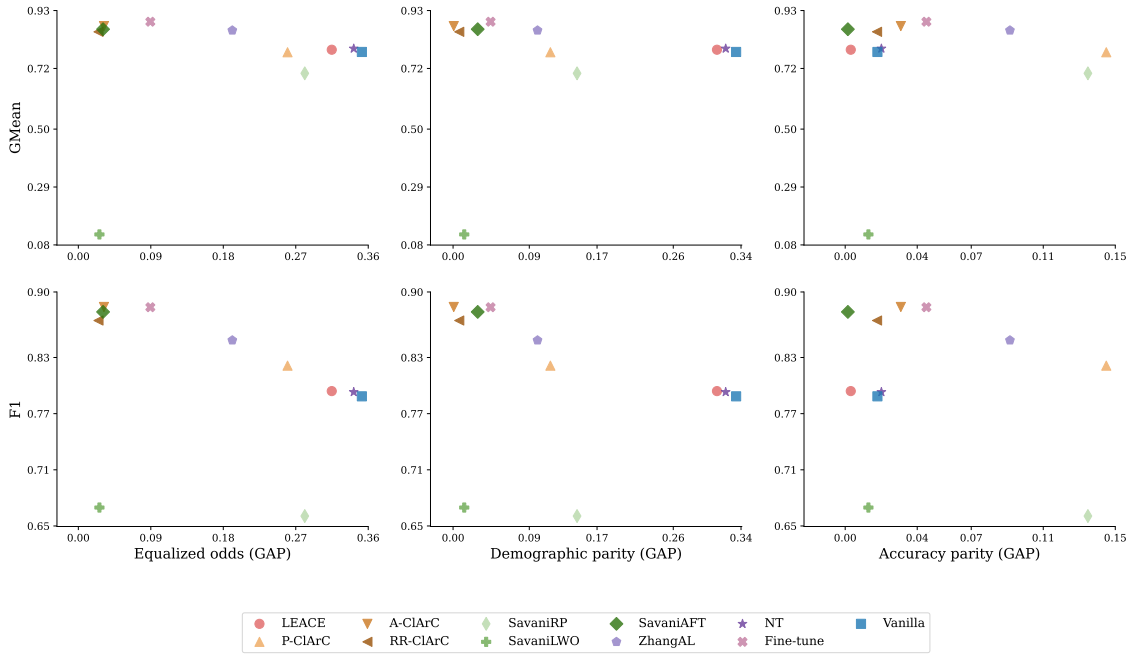


FIGURE 5.5: Performance-fairness trade-off for the "Smiling-Wearing Hat" dataset variant. Figure structure and interpretation are similar to Figure 5.4.

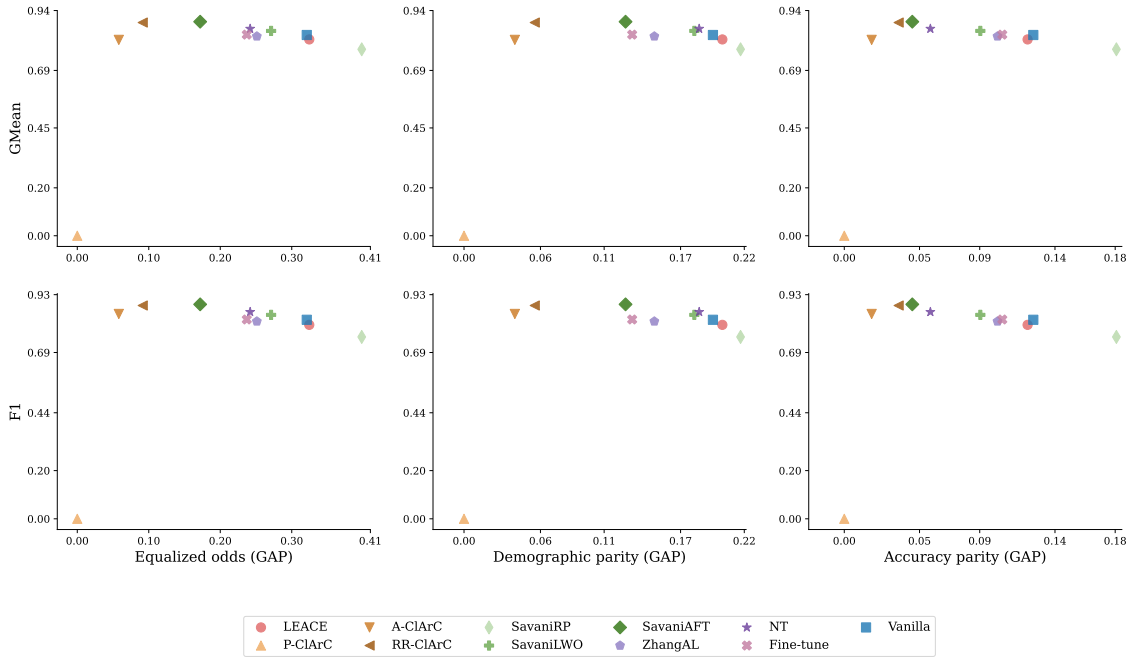


FIGURE 5.6: Performance-fairness trade-off for the "Smiling-Wearing Necktie" dataset variant. Figure structure and interpretation are similar to Figure 5.4.

Most DetoxAI debiasing methods outperform the *Vanilla* baseline, demonstrating the tool's ability to reduce bias without sacrificing, and sometimes enhancing, model predictive performance. These methods quite consistently achieve a strong balance between fairness and performance, often appearing as the most effective methods across all fairness metrics and datasets. This is especially true for *ZhangAL*, *SavaniAFT*, *FineTune*, *RR-ClArC*, *A-ClArC*, which all usually push the Pareto frontier towards the ideal point.

We included *FineTune* as a baseline method to compare against more complex debiasing approaches. Unlike other methods that explicitly target bias reduction, *FineTune* simply adjusts the model using the same debiasing data as other techniques without any added fairness rules or constraints. This comparison helps isolate whether improvements come from advanced debiasing strategies or just from adapting to unbiased training data. Despite its simplicity *FineTune* achieves competitive fairness and performance results, showing that even basic retraining on carefully prepared data can reduce bias.

Naive Threshold Optimization (*NT*) offers fairness improvements, particularly for Equalized Odds (*EO_GAP*) - this is the specific fairness measure that was specified in the *NT* objective, and Demographic Parity (*DP_GAP*) gaps. *NT* maintains or slightly improves performance metrics while reducing *EO_GAP* and *DP_GAP* bias. Its impact on Accuracy Parity (*AP_GAP*) gap is less pronounced. Considering its simplicity and lack of intervention in the model's reasoning process, its performance is surprisingly good.

Concept-based methods – *LEACE*, *ClArC* family don't directly optimize specific fairness metrics, instead, they work with Concept Activation Vectors that represent the bias. In their case, we observe improvements in all fairness metrics and the same or worse performance. *LEACE* seems to be the least stable method considering that its results land very closely to the *Vanilla* model and often lead to some deterioration on the performance-fairness frontier. *ClArC* methods usually improve the trade-off with fine-tuning based methods – *A-ClArC* and *RR-ClArC* being visibly better than the *P-ClArC*. Moreover, in a lot of cases, the whole *ClArC* family results in poorer predictive performance, suggesting that classification *for the right reasons* leads to performance-fairness trade-offs.

Both Savani's family and *ZhangAL* methods are set to optimize *EO_GAP*, however, the improvement in *EO_GAP* compared to, for instance, *DP_GAP* is not significantly better, which is an interesting finding. We hypothesize that the way optimization is framed for these methods – that is, constrained optimization of *BA* with a strict requirement on $EO\_GAP < 0.1$ – admits a lesser trade-off with *DP_GAP* because it has no further incentive to maximize *EO_GAP* past 0.1 and focuses strictly on *BA*.

5.3.2 Feature relevance in model's reasoning process

RQ2: Can DetoxAI shift feature relevance in the model's reasoning process?

To investigate whether DetoxAI can alter the model's focus from protected attributes to task-relevant features, we visualized feature relevance maps for three dataset variants: *"Smiling-Wearing Hat"*, *"Smiling-Male"*, and *"Smiling-Wearing Necktie"*. We compare the feature relevance maps of the *Vanilla* model with those obtained after applying three selected DetoxAI methods: *RR-ClArC* and *SavaniAFT*. *SavaniAFT* was chosen for its consistently strong performance in preformance-fairness trade-offs as observed in RQ1. *RR-ClArC* was selected to represent concept-based debiasing methods that utilize CAVs to mitigate bias. We considered positive and negative values of protected attributes in the comparison and fixed the target to recognize if the person in the photo is smiling.

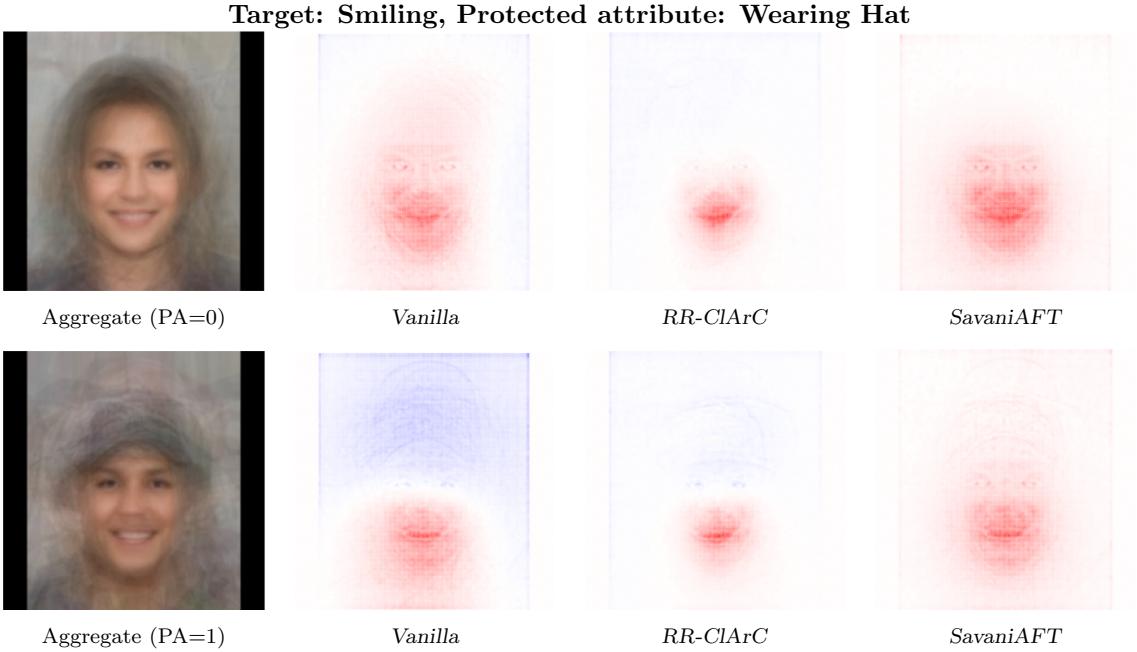


FIGURE 5.7: Relevancy maps for wearing hat protected attribute (0 meaning that was not present, 1 meaning that this person was wearing a hat). Red indicates a positive contribution to prediction, blue indicates negative.

Figure 5.7 presents the feature relevance maps for the *"Smiling-Wearing Hat"* variant. For the *Vanilla* model, the relevance maps often highlight regions around the face of the person, but some of the relevancy goes to the forehead. This is especially visible in the cases where the person is wearing a hat (PA=1). When the person is not wearing a hat (PA=0), the *Vanilla* model already shows some concentration of relevance around the mouth, but still with noticeable activation in the forehead region. However, when the person is wearing a hat, the *Vanilla* model's focus shifts upwards, with increased relevance on the forehead and hat area, and even shows some negative (blue) relevance around the mouth. This suggests that the *Vanilla* model might be incorrectly associating the presence of a hat with a reduced likelihood of smiling, or at least focusing on hat features instead of smile features when a hat is present. Both debiasing methods we considered in this comparison were able to shift relevancy onto the mouth region of the face, causing the model to look at the parts of the face that actually matter. In both cases, whether the person is wearing a hat or not, *RR-ClArC* and *SavaniAFT* demonstrate a clear focus on the mouth region, with minimal relevance assigned to the forehead or hat area. In other words, the model, after debiasing, started to perform classification *for the right reasons*. This observation was expected in the case of *RR-ClArC* (because of direct penalization of sensitivity to the concept), but surprisingly also

for *SavaniAFT* which optimized *EO_GAP*.

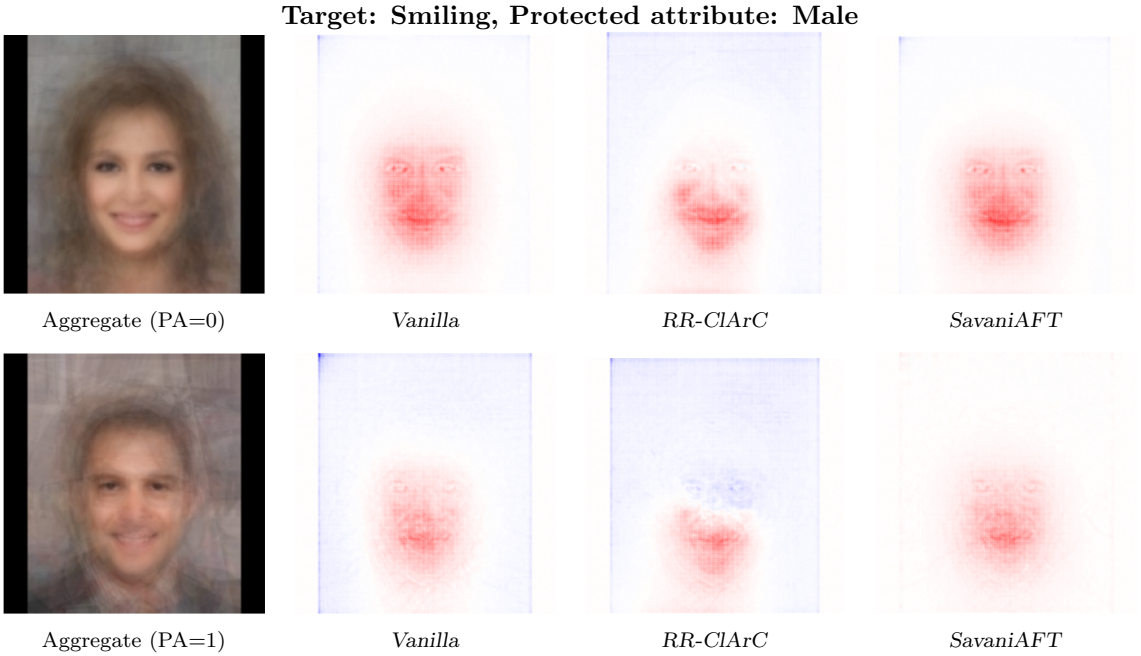


FIGURE 5.8: Relevancy maps for male protected attribute (0 meaning the person was labeled as a female, 1 meaning that this person was labeled as a man). Red indicates a positive contribution to prediction, blue indicates a negative.

Figure 5.8 presents the feature relevance maps for the *"Smiling-Male"* variant. In the *Vanilla* case, before debiasing, the relevancy seems to be spread out around the whole face of the person, suggesting that the model may be trying to identify the gender of the person instead of looking at their mouth in search of a smile. When considering females (PA = 0), the *Vanilla* model shows a somewhat diffuse relevance map across the face, with some concentration around the mouth and eyes but also significant activation in the cheek and jawline areas. For men (PA = 1), the *Vanilla* model's relevance map becomes even more dispersed, less focused on the mouth, and exhibits blue regions around the mouth and chin, potentially indicating a negative correlation of male gender with smiling. In this case, the shift to the region of the mouth is only visible in the case of *RR-ClArC*, while *SavaniAFT* still spreading relevancy throughout the whole face. *RR-ClArC* demonstrates a clear shift of relevance to the mouth region for both females and males, effectively ignoring gender-related features and focusing on the smile. In contrast, *SavaniAFT* maintains a more diffuse relevance pattern across the face for both genders, similar to the *Vanilla* model, suggesting it is less effective in shifting feature relevance away from gender bias in this scenario.

Figure 5.9 is considering the same target as before, but now the bias is focused on the person wearing a necktie. The base case (*Vanilla*) in the case where the necktie is in the photo shifts some relevancy there. When a necktie is absent (PA = 0), the *Vanilla* model shows relevance across the face and neck area. However, when a necktie is present (PA = 1), the *Vanilla* model's relevance map prominently highlights the necktie and neck region, with some negative (blue) relevance appearing around the mouth area. This suggests the model is influenced by the presence of a necktie and might be associating it with a lower likelihood of smiling, or again prioritizing necktie features over smile features. Similarly to Figure 5.7, both of the methods here cause the model to look more at the region around the mouth and shift relevancy away from the neck region, where the necktie is expected to be. Both *RR-ClArC* and *SavaniAFT* effectively concentrate relevance on the mouth area, regardless of whether a necktie is present or not. In both scenarios (necktie present

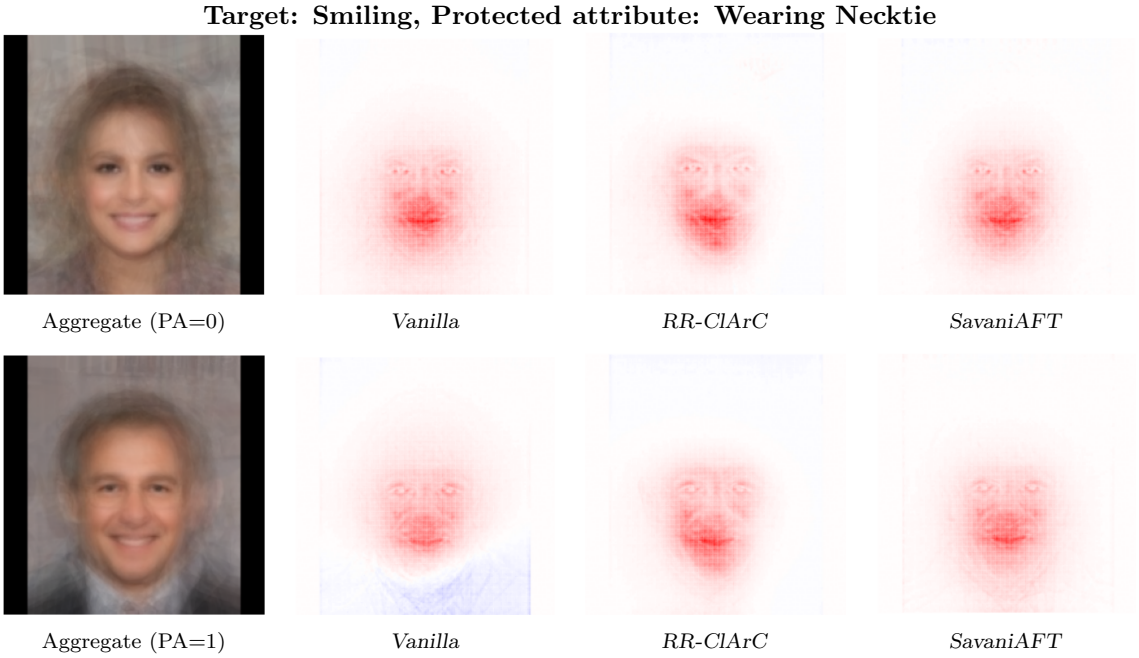


FIGURE 5.9: Relevancy maps for wearing necktie attribute protected attribute (0 meaning necktie was not present, 1 meaning that this person was wearing a necktie). Red indicates positive contribution to prediction, blue indicates negative.

or absent), these debiasing methods successfully mitigate the influence of the necktie feature and guide the model to focus on the task-relevant smile features.
Several additional attribution map visualizations are also available in Appendix C.

This analysis indicates that DetoxAI can indeed shift the relevance away from the protected attributes. This is an important finding, as it provides a glimpse into the qualitative aspect of bias mitigation methods, which is often overlooked in standard evaluations of algorithmic fairness. The ability to shift the model's relevancy to the correct features provides more trust in how these models behave, especially in sensitive contexts. Furthermore, by ensuring models focus on genuinely relevant features like smiles rather than spurious correlations with protected attributes like hats, gender, or neckties, we can expect improved generalization and robustness, especially in diverse and real-world scenarios where these biases might not hold. The observed shift in feature relevance underscores the potential of DetoxAI not just as a fairness-enhancing toolkit but also as a means to develop models that are more aligned with human understanding of the task and less susceptible to unintended biases embedded in the data.

5.3.3 Relationship between the target and the protected attribute

RQ3: Does the relationship between the target and the protected attribute of the dataset affect the ability of DetoxAI to mitigate bias?

To address this research question, we perform a sensitivity analysis that examines how the correlation between the target and protected attributes in a dataset influences DetoxAI's bias mitigation capabilities. Specifically, we synthetically undersample datasets to enforce specific correlation coefficients and measure their impact. We conduct two complementary analyses: in the first, we ensure that the debiasing dataset is balanced while sweeping the correlation coefficients in the training dataset. In the second, the correlation coefficient of the training dataset is fixed at 60%, and we sweep the correlation coefficients in the debiasing dataset.

Correlation coefficient sweep in training data In this analysis, we investigate the effect of varying the correlation coefficient between the target and protected attribute in the training dataset. To isolate this effect, we fix the correlation coefficient in the debiasing dataset to $\phi = 0$. The correlation coefficient of the training dataset is swept across $\phi \in \{1.0, 0.8, 0.6, 0.4, 0.2, 0.0\}$.

As shown in Figure 5.10, the performance improvement achieved by DetoxAI over the *Vanilla* model is visualized under different correlations of the training set. To aid in interpretation, we invert the balanced accuracy (BA) metric to balanced accuracy error ($BA\ error = 1 - BA$), which aligns with EO_GAP by making both metrics minimization objectives. Thus, the optimal result lies at the origin (0, 0) of the axes in the plot.

The results indicate that as the correlation coefficient in the training dataset decreases, the *Vanilla* model exhibits reduced bias (measured by EO_GAP) and improved predictive performance (measured by BA). Regardless of that, DetoxAI consistently improves the performance-fairness Pareto frontier at all correlation levels. The gray dots in Figure 5.10 represent all the methods run within DetoxAI, most of which outperform the *Vanilla* model. Importantly, these findings demonstrate that even under challenging training data imbalances, resulting in poor performance and high unfairness of the *Vanilla* model, DetoxAI can deliver substantial gains provided the unlearn dataset is balanced. This is especially visible at the macro level in Figure 5.11 and at the per-method level in Figure 5.14.

However, it is crucial to avoid overgeneralizing these results. Although data balance in the debiasing dataset is desirable, it is not a sufficient condition for fairness. Existing studies [10, 42] show that the balance of the dataset alone does not eliminate all biases, as specific artifacts or biases can persist despite balance.

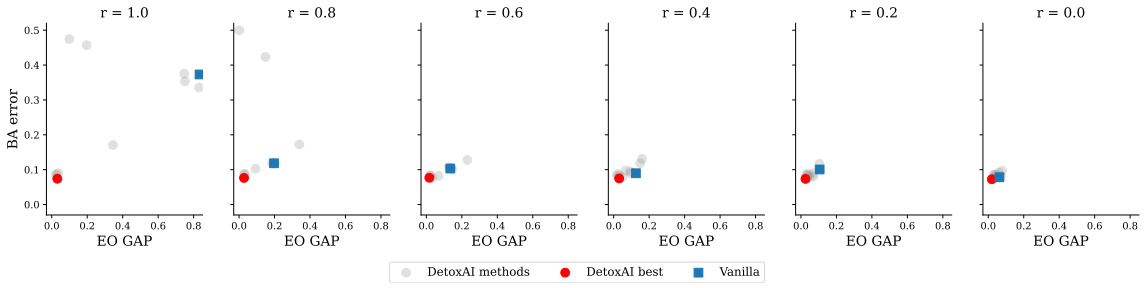


FIGURE 5.10: DetoxAI's improvement over a *Vanilla* (blue square) model on the performance-fairness trade-off. The red dot represents DetoxAI's best method for a given setting, identified as the closest point to the ideal (0, 0). Gray dots represent results from all methods tested within DetoxAI.

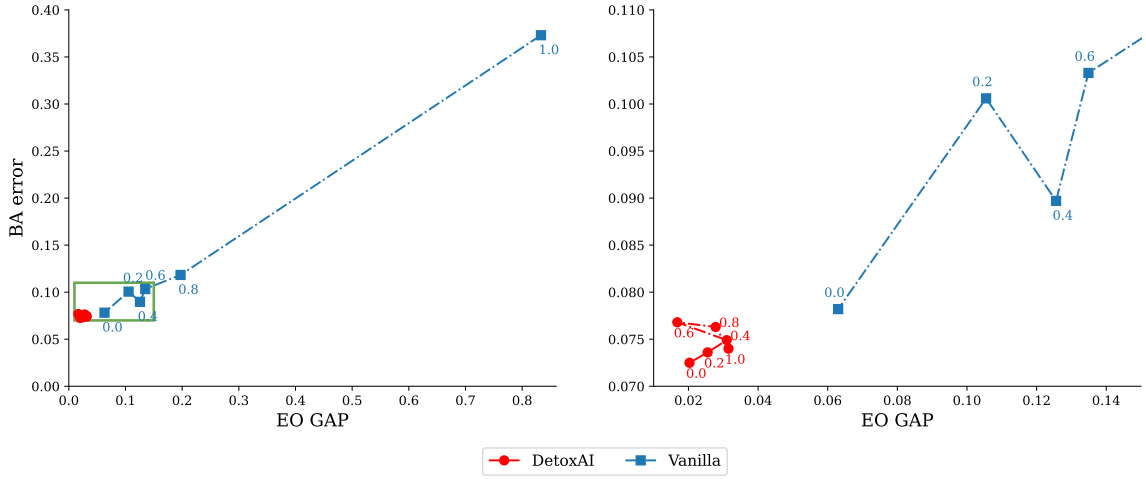


FIGURE 5.11: Rate of improvement by the best-performing DetoxAI method over the *Vanilla* model. The plot on the right zooms in on the area outlined by the green square.

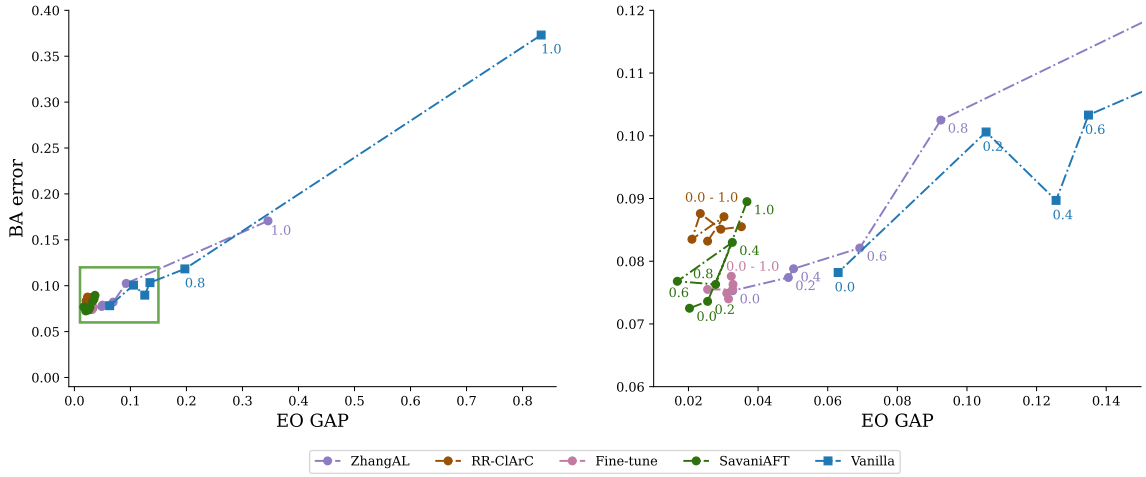


FIGURE 5.12: Per-method performance breakdown, highlighting contributions of selected methods to performance-fairness improvements within DetoxAI.

Correlation coefficient sweep in debiasing data In this scenario, the correlation coefficient in the training dataset is fixed at $\phi = 0.6$, while we sweep the correlation coefficient in the debiasing dataset across 12 scenarios. From the results depicted in Figures 5.13 and 5.14, we observe a clear trend: as the correlation coefficient in the debiasing dataset decreases (indicating improved balance), the performance of DetoxAI improves both in terms of fairness and predictive performance. In particular, the performance-fairness trade-off is consistently improved until the debiasing dataset's correlation coefficient reaches $\phi = 0.8$, beyond which DetoxAI struggles to deliver significant gains over the *Vanilla* model.

This finding reinforces the notion that a well-balanced debiasing dataset aids an effective model correction. Although complete balance (i.e., $\phi = 0.0$) is not always necessary, achieving a sufficiently low correlation in the debiasing dataset helps DetoxAI's methods to succeed. Specifically, in this experiment, the critical threshold appears to be below $\phi = 0.8$. This aligns with prior research [42, 10], which highlights that while balance alone does not guarantee fairness, there exists an optimal range of balance that maximizes fairness and performance for a given dataset. Moreover, note that obtaining a balanced dataset for bias mitigation often can be achieved by undersampling whatever dataset is available. This is much cheaper to do than to have a balanced dataset in the model training phase because, in deep learning, the fine-tuning step (and in our case, debiasing) typically requires much fewer data.

Furthermore, many debiasing techniques implemented within DetoxAI rely on fine-tuning mechanisms that are inherently sensitive to dataset balance. Fine-tuning on a balanced debiasing set achieves competitive performance but is often surpassed by methods explicitly designed to optimize fairness alongside predictive performance. For more granular, per-method results, please refer to Appendix D.

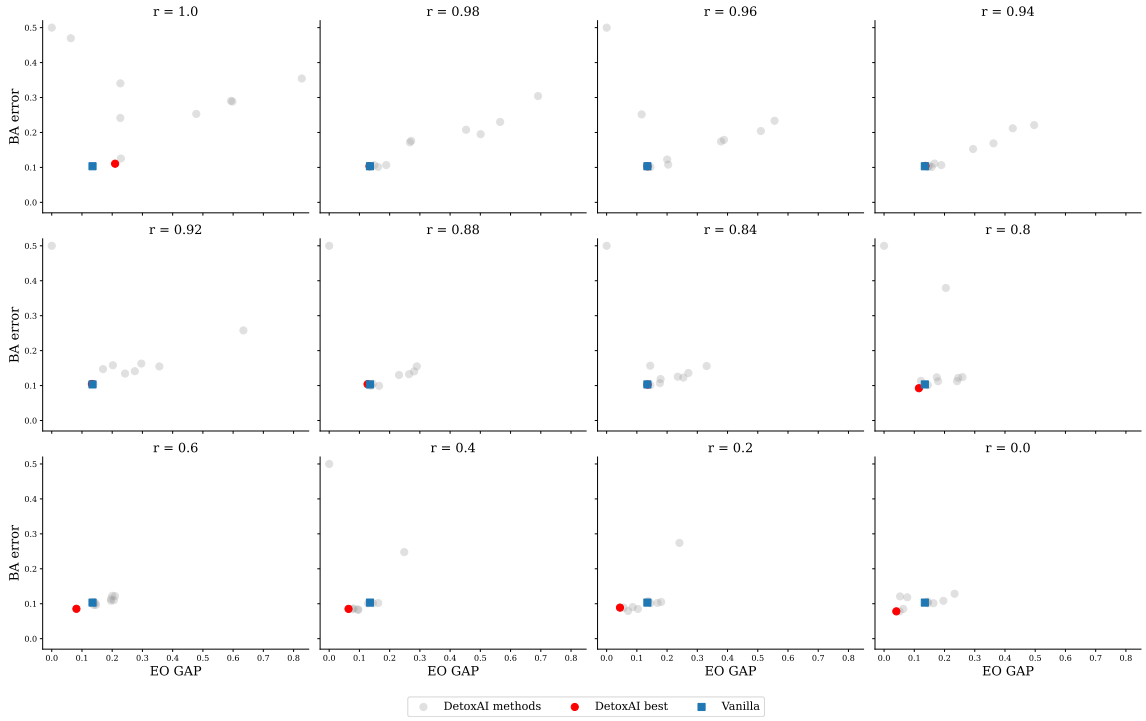


FIGURE 5.13: DetoxAI's improvement over the *Vanilla* model (blue square) on the performance-fairness trade-off. The red dot marks the best-performing DetoxAI method for a given setting, chosen as the closest point to the ideal (0, 0). Gray dots represent results from all methods tested within DetoxAI.

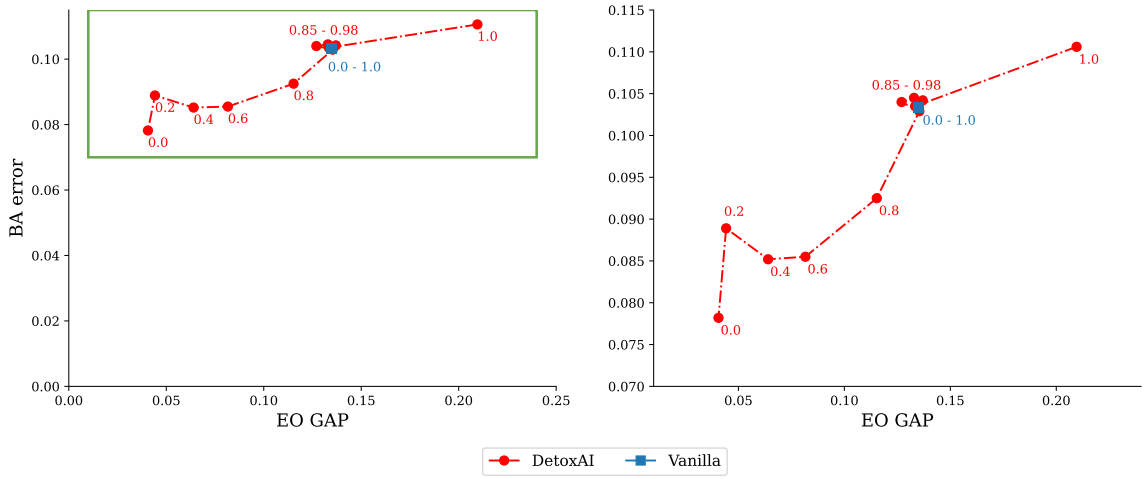


FIGURE 5.14: Zoomed-in view of DetoxAI's improvements. The right-hand side highlights results within the region outlined by the green rectangle.

5.3.4 Impact of the backbone model size

RQ4: Does the underlying model size affect the ability of DetoxAI to mitigate bias?

In this section, we examine how the size of the underlying model influences DetoxAI's ability to improve the performance-fairness trade-off. The experimental setup considers a scenario with $\phi = 0.6$ in the training dataset and a balanced debiasing set. We chose ResNet [1] as the backbone model due to its availability in five size variants:

Variant	Parameters
ResNet-18	11.7 M
ResNet-34	21.8 M
ResNet-50	25.6 M
ResNet-101	44.6 M
ResNet-152	60.2 M

TABLE 5.4: Parameter count for ResNet variants.

As shown in Figures 5.15 and 5.16, DetoxAI achieves similar levels of performance-fairness improvement across all model sizes. The experiments indicate that the greatest improvement was observed for ResNet-101, the second-largest model. However, the difference in improvement compared to other ResNet variants is not significant, which is less than 2 percentage points in both *BA* and *EO_GAP*.

These findings suggest that DetoxAI is not sensitive to model size and can be effectively applied to a range of backbone models. However, it is important to note that, due to computational limitations, we were unable to test models with hundreds of millions of parameters or more. Therefore, our claims are limited to models with fewer than approximately 150M parameters. The largest verified model in this context is VGG11, discussed in the next section.

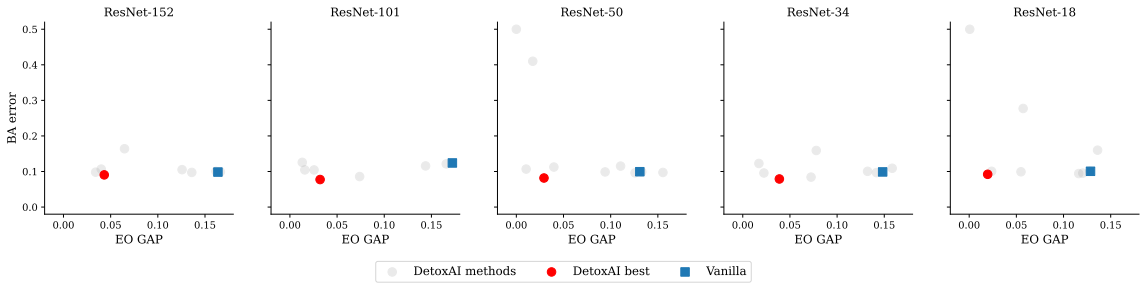


FIGURE 5.15: Performance-fairness trade-off improvements across ResNet variants. The red dots denote DetoxAI's best correction, selected as the closest in Euclidean distance from the ideal point (0, 0).

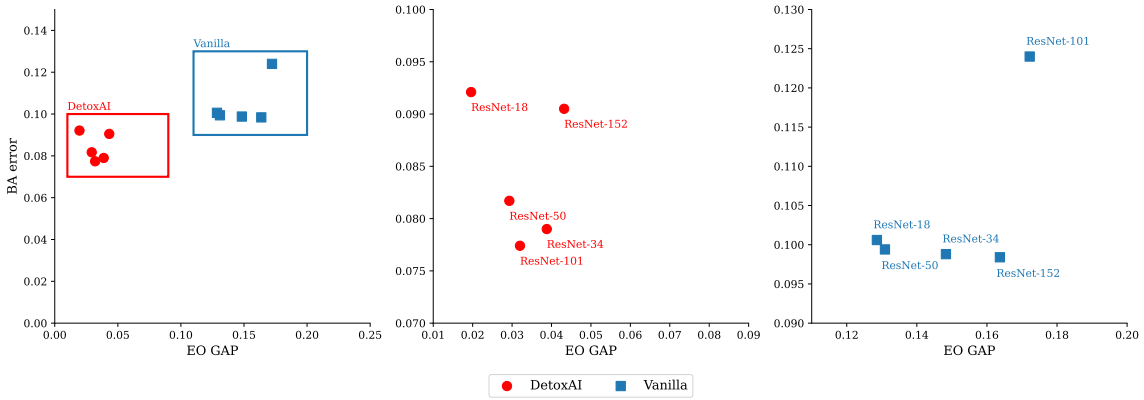


FIGURE 5.16: Detailed view of performance-fairness improvements for ResNet variants.

5.3.5 Impact of the backbone model architecture

RQ5: Does the underlying model architecture affect the ability of DetoxAI to mitigate bias?

In this section, we examine whether DetoxAI's performance is influenced by the architecture of the underlying model. The experimental setup mirrors the previous section, with a fixed scenario involving $\phi = 0.6$ in the training dataset and a balanced debiasing set. Instead of focusing on variants of ResNet, we evaluate a diverse range of convolutional architectures commonly used in computer vision tasks, all implemented in the popular *torchvision* Python library.

From supported models, where possible, we select those with the closest parameter counts to 25M to ensure a certain level of comparability across architecture variants. These are:

Model	Parameters
EfficientNetV2 (small) [32]	28.6 M
ConvNeXt (tiny) [34]	21.8 M
MobileNetV3 (large) [33]	5.5 M
VGG-11 [2]	132.9 M
ResNet-50 [1]	25.6 M

TABLE 5.5: Parameter counts for evaluated architectures.

As shown in Figures 5.17 and 5.18, DetoxAI demonstrates robust performance across all tested architectures, indicating that its effectiveness is largely independent of the model architecture. These results suggest that DetoxAI can generalize well across a variety of practical settings.

However, it is important to note that all the architectures evaluated here are CNN-based. We have not verified DetoxAI's performance on transformer-based or state-space models, and as such, our conclusions should be interpreted with caution in the context of these alternative architectures.

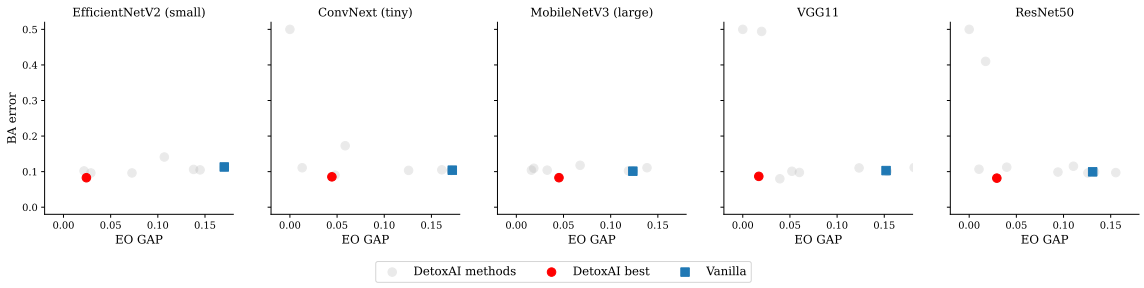


FIGURE 5.17: Performance-fairness trade-off improvements across various model architectures. The red dots denote DetoxAI's best correction, selected as the closest in Euclidean distance from the ideal point (0, 0).

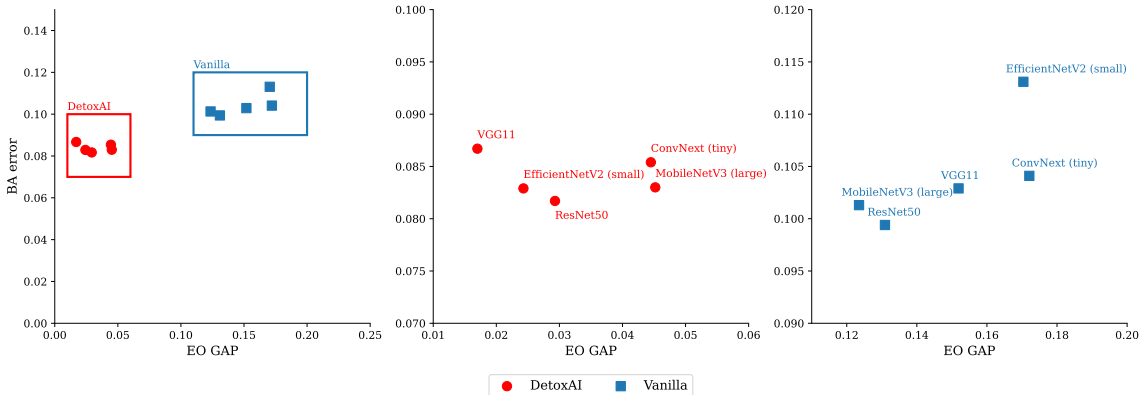


FIGURE 5.18: Detailed view of performance-fairness improvements for selected architectures.

Chapter 6

Final Remarks

6.1 Conclusions

In this work, we identified a significant gap in the fair machine learning ecosystem: the lack of debiasing toolkits specifically tailored for deep vision models, alongside the absence of qualitative tools for analyzing the outcomes of debiasing intervention methods. To bridge this gap, we proposed DetoxAI, a production-ready Python toolkit designed to enable developers and researchers to debias models in a post-hoc fashion using a variety of implemented methods (*LEACE*, *P-ClArC*, *A-ClArC*, *RR-ClArC*, *SavaniAFT*, *SavaniRP*, *SavaniLWO*, *ZhangAL*, *NT*).

DetoxAI offers a range of debiasing techniques, allowing practitioners to prioritize specific fairness and performance metrics. This flexibility underscores its utility in addressing the intricate challenges of attaining fairness in machine learning. Additionally, the toolkit includes visualization utilities in the form of attribution maps, which facilitate qualitative assessments of the debiasing process. To the best of our knowledge, DetoxAI is the first toolkit explicitly built for deep learning image classification tasks, integrating multiple debiasing methods into a unified and user-friendly framework.

Summary of results Our comprehensive experimental evaluation of the implemented methods was conducted on many variants of the CelebAdataset, and combination of biases, target features, and protected attributes. Visual analysis of biases using attribution maps (Section 5.3.2) demonstrated that the *Vanilla* model assigns relevance to image regions corresponding to specific protected attributes (e.g., *"Wearing Hat"* when the target feature is *"Smiling"*). The results also showed that certain methods within DetoxAI (particularly *RR-ClArC*) successfully redirected the model's focus away from protected attributes. Moreover, the quantitative experiments (Section 5.3.1) demonstrated that the methods implemented in DetoxAI frequently enhance model fairness (as measured by e.g., *Equalized Odds* and *Accuracy Parity*) while maintaining or even improving performance (as measured by e.g., *Geometric Mean* and *F1*) compared to the base *Vanilla* model. The experiments revealed varying performances of debiasing methods across different scenarios, with methods such as *SavaniAFT*, *ZhangAL*, *RR-ClArC*, and *A-ClArC* consistently achieving the best results. We hypothesize that these methods excel due to their fairness-aware fine-tuning mechanisms, which involve deeper interventions in model representations compared to methods relying solely on direct latent-space transformations without fine-tuning (*LEACE*, *P-ClArC*), threshold optimization without modifying model parameters (*NT*), or layer-wise iterative parameter perturbation and optimization (*SavaniRP*, *SavaniLWO*).

Furthermore, the set of experiments carried out in Sections 5.3.3 to 5.3.5 demonstrated that DetoxAI as a toolkit, consistently achieves significant improvements in the performance-fairness

trade-off over the *Vanilla* model. This evaluation encompassed various levels of biased associations in both training and debiasing datasets, multiple backbone model architectures, and parameter sizes, showcasing DetoxAI's effectiveness across a wide spectrum of convolutional models with sizes under 150 million parameters.

Closing remarks In conclusion, this thesis provides a comprehensive overview of the architecture and capabilities of DetoxAI. We implemented nine post-processing debiasing methods from the ground up and integrated them into a unified code structure. We examined the performance-fairness trade-off of the implemented techniques and provided a detailed analysis. In addition, we analyzed changes in the attribution maps for different debiasing methods, contrasting these changes with biased models, and verifying the utility of these methods from an Explainable AI perspective. Furthermore, we developed and deployed a Python package on PyPI, emphasizing DetoxAI's readiness for seamless integration into modern machine learning workflows, allowing practitioners to adopt it with minimal friction. As such, we successfully met the objectives outlined in the aims and scope of this work (Section 1.2).

The project offered us a unique opportunity to acquire both theoretical knowledge and practical skills. It pushed us beyond the curriculum of our academic major, requiring extensive familiarization with machine learning and explainability. Additionally, we grounded our familiarity in PyTorch and deep learning, as our work involved modifying and extending functionality deep within the models, well beyond typical usage scenarios.

Our team worked in close collaboration, with frequently overlapped responsibilities and contributions. This dynamic allowed each member to gain a comprehensive understanding of the entire process, from conducting initial literature reviews and identifying research gaps to formulating hypotheses, validating ideas, rapid prototyping, and testing. We also engaged in software validation, modern MLOps experiment pipelines, code versioning, and best practices in software engineering. Many of these tasks were tackled during pair programming sessions, following Extreme Programming (XP) principles. This collaborative approach proved especially valuable during the ideation phase, where we collectively reviewed the literature and rapidly tested proof-of-concept solutions. The experience not only deepened our technical expertise but also strengthened our ability to work collaboratively on complex technical challenges.

Ultimately, we successfully developed a robust and well-structured software library. This open-source project, licensed under CC-BY-4.0, is now freely available for anyone to use and build upon.

6.2 Limitations and future work

Despite meeting the set aims of this work and demonstrating DetoxAI's utility, we acknowledge several limitations that could be addressed in future research.

First, our evaluation focused on the CelebA dataset and its variants. Broader claims about the utility of DetoxAI would benefit from testing the toolkit on a wider range of datasets, particularly those representing diverse real-world applications. Testing DetoxAI in real-world production environments is an essential next step.

While the current version of DetoxAI focuses on computer vision, its principles could be extended to other modalities beyond the scope of this thesis, particularly text. Recent work on large language models (LLMs) has revealed significant biases in these systems [76], and the high cost of training them makes post-hoc debiasing particularly relevant. Adapting the methods in DetoxAI to

text-based models could provide critical tools for ensuring fairness in these increasingly widespread systems.

The current implementation primarily focuses on post-processing techniques, specifically post-training model adaptations through classifier corrections. Although we implemented a threshold optimization technique (NT) as part of the output correction methods, expanding the range of available debiasing tools to include additional post-processing techniques could further enhance DetoxAI's versatility.

Our experimental evaluation involved various model sizes (e.g., ResNet-18 through ResNet-152) and architectures (e.g., EfficientNet, ResNet, VGG, MobileNet, ConvNeXt). However, including transformer-based models and the largest convolutional neural network variants remains an open challenge. Due to their computational demands, these larger models were not evaluated in this study. For smaller transformers, compatibility issues with the prediction attribution software used in DetoxAI posed additional challenges. Addressing this limitation by updating the attribution software would enable broader applicability to transformer-based architectures.

Despite these limitations, DetoxAI lays a strong foundation for advancing research on debiasing techniques for ensuring fairness in deep learning. Future efforts should focus on expanding its scope, improving compatibility, and addressing computational challenges to maximize its impact across diverse application domains. The development of DetoxAI presents numerous opportunities for future research. Its modular architecture creates an environment for designing and evaluating new debiasing algorithms, fostering systematic comparisons across different approaches while maintaining consistent evaluation protocols.

Bibliography

[1] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[2] K Simonyan and A Zisserman. Very deep convolutional networks for large-scale image recognition. In *3rd International Conference on Learning Representations (ICLR 2015)*, 2015.

[3] Mingxing Tan and Quoc Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pages 6105–6114. PMLR, 2019.

[4] Omkar M. Parkhi, Andrea Vedaldi, and Andrew Zisserman. Deep face recognition. In *British Machine Vision Conference*, 2015.

[5] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in Neural Information Processing Systems*, 33:1877–1901, 2020.

[6] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[7] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, pages 8748–8763. PMLR, 2021.

[8] Luca Longo, Mario Brcic, Federico Cabitza, Jaesik Choi, Roberto Confalonieri, Javier Del Ser, Riccardo Guidotti, Yoichi Hayashi, Francisco Herrera, Andreas Holzinger, Richard Jiang, Hassan Khosravi, Freddy Lecue, Gianclaudio Malgieri, Andrés Páez, Wojciech Samek, Johannes Schneider, Timo Speith, and Simone Stumpf. Explainable artificial intelligence (xai) 2.0: A manifesto of open challenges and interdisciplinary research directions. *Information Fusion*, 106:102301, 2024. ISSN 1566-2535.

[9] Lisa Anne Hendricks, Kaylee Burns, Kate Saenko, Trevor Darrell, and Anna Rohrbach. Women also snowboard: Overcoming bias in captioning models. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 771–787, 2018.

[10] Vítor Albiero, Kai Zhang, and K. Bowyer. How does gender balance in training data affect face recognition accuracy? *2020 IEEE International Joint Conference on Biometrics (IJCB)*, pages 1–10, 2020.

[11] Jeff Mattu, Lauren Larson, Surya Kirchner, and Julia Angwin. Machine Bias, 2016. ProPublica.

[12] Julia Dressel and Hany Farid. The accuracy, fairness, and limits of predicting recidivism. *Science Advances*, 4(1):eaao5580, 2018.

[13] Maarten Buyl and Tijl De Bie. Inherent limitations of ai fairness. *Communications of the ACM*, 67(2):48–55, 2024.

[14] Ziad Obermeyer, Brian Powers, Christine Vogeli, and Sendhil Mullainathan. Dissecting racial bias in an algorithm used to manage the health of populations. *Science*, 366(6464):447–453, 2019.

[15] Pinar Barlas, Kyriakos Kyriakou, Olivia Guest, Styliani Kleanthous, and Jahna Otterbacher. To "See" is to Stereotype: Image Tagging Algorithms, Gender Recognition, and the Accuracy-Fairness Trade-off. *Proc. ACM Hum.-Comput. Interact.*, 4(CSCW3):232:1–232:31, January 2021.

[16] Tolga Bolukbasi, Kai-Wei Chang, James Y Zou, Venkatesh Saligrama, and Adam T Kalai. Man is to computer programmer as woman is to homemaker? debiasing word embeddings. *Advances in Neural Information Processing Systems*, 29, 2016.

[17] Simon Caton and Christian Haas. Fairness in machine learning: A survey. *ACM Computing Surveys*, 56:1 – 38, 2020.

[18] Max Hort, Zhenpeng Chen, J Zhang, Federica Sarro, and Mark Harman. Bias mitigation for machine learning classifiers: A comprehensive survey. *ACM Journal on Responsible Computing*, 1:1 – 52, 2022.

[19] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. A survey on bias and fairness in machine learning. *ACM Computing Surveys*, 54(6), July 2021. ISSN 0360-0300.

[20] Han Zhao and Geoffrey J Gordon. Inherent tradeoffs in learning fair representations. *Journal of Machine Learning Research*, 23(57):1–26, 2022.

[21] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 2015.

[22] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[23] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer-Verlag, Berlin, Heidelberg, 1st edition, 2010. ISBN 1848829345.

[24] Yuanqing Lin, Fengjun Lv, Shenghuo Zhu, Ming Yang, Timothée Cour, and Liangliang Cao. Large-scale image classification: Fast feature extraction and svm training. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1, pages 1689–1696, 06 2011.

[25] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition. *Neural Computation*, 1(4):541–551, 1989.

[26] David H. Hubel and Torsten N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *The Journal of Physiology*, 195, 1968.

[27] Alexander Mordvintsev, Christopher Olah, and Mike Tyka. Deepdream-a code example for visualizing neural networks. *Google Research*, 2(5), 2015.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton. Imagenet classification with deep convolutional neural networks. *Advances in Neural Information Processing Systems*, 25, 2012.

[29] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 248–255, 2009.

[30] Jorge Sánchez and Florent Perronnin. High-dimensional signature compression for large-scale image classification. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1665–1672, 06 2011.

[31] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. In *International Conference on Learning Representations*, 2021.

[32] Mingxing Tan and Quoc Le. Efficientnetv2: Smaller models and faster training. In Marina Meila and Tong Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 10096–10106. PMLR, 18–24 Jul 2021.

[33] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *ArXiv*, abs/1704.04861, 2017.

[34] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. A convnet for the 2020s. *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2022.

[35] Andreas Holzinger, Anna Saranti, Christoph Molnar, Przemyslaw Biecek, and Wojciech Samek. *Explainable AI Methods - A Brief Overview*, pages 13–38. Springer International Publishing, Cham, 2022. ISBN 978-3-031-04083-2.

[36] Scott M. Lundberg and Su-In Lee. A unified approach to interpreting model predictions. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 4768–4777, Red Hook, NY, USA, 2017. Curran Associates Inc. ISBN 9781510860964.

[37] Ramprasaath R Selvaraju, Michael Cogswell, Abhishek Das, Ramakrishna Vedantam, Devi Parikh, and Dhruv Batra. Grad-cam: visual explanations from deep networks via gradient-based localization. *International Journal of Computer Vision*, 128:336–359, 2020.

[38] K Simonyan, A Vedaldi, and A Zisserman. Deep inside convolutional networks: visualising image classification models and saliency maps. In *Proceedings of the International Conference on Learning Representations (ICLR)*. ICLR, 2014.

[39] Daniel Smilkov, Nikhil Thorat, Been Kim, Fernanda Viégas, and Martin Wattenberg. Smoothgrad: removing noise by adding noise. *Workshop on Visualization for Deep Learning at International Conference on Machine Learning*, 2017.

[40] Sebastian Bach, Alexander Binder, Grégoire Montavon, Frederick Klauschen, Klaus-Robert Müller, and Wojciech Samek. On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PLOS ONE*, 10(7):1–46, 07 2015.

[41] Been Kim, Martin Wattenberg, Justin Gilmer, Carrie Cai, James Wexler, Fernanda Viegas, et al. Interpretability beyond feature attribution: Quantitative testing with concept activation vectors (tcav). In *International Conference on Machine Learning*, pages 2668–2677. PMLR, 2018.

[42] Maximilian Dreyer, Frederik Pahde, Christopher J. Anders, Wojciech Samek, and Sebastian Lapuschkin. From hope to safety: Unlearning biases of deep models via gradient penalization in latent space. *Proceedings of the AAAI Conference on Artificial Intelligence*, 38(19):21046–21054, Mar. 2024.

[43] S Wachter, B Mittelstadt, and C Russell. Counterfactual explanations without opening the black box: automated decisions and the gdpr. *Harvard Journal of Law and Technology*, 31 (2), 2018.

[44] Michael A. Madaio, Luke Stark, Jennifer Wortman Vaughan, and Hanna Wallach. Co-designing checklists to understand organizational challenges and opportunities around fairness in ai. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–14, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367080.

[45] Anna Jobin, Marcello Ienca, and Effy Vayena. The global landscape of ai ethics guidelines. *Nature Machine Intelligence*, 1(9):389–399, 2019.

[46] A. Feder Cooper, Ellen Abrams, and NA NA. Emergent unfairness in algorithmic fairness-accuracy trade-off research. In *Proceedings of the 2021 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '21, page 46–54, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384735.

[47] Jahna Otterbacher, Pınar Barlas, Styliani Kleanthous, and Kyriakos Kyriakou. How Do We Talk about Other People? Group (Un)Fairness in Natural Language Image Descriptions. *Proceedings of the AAAI Conference on Human Computation and Crowdsourcing*, 7:106–114, October 2019. ISSN 2769-1349.

[48] Cynthia Dwork, Moritz Hardt, Toniann Pitassi, Omer Reingold, and Richard Zemel. Fairness through awareness. In *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, pages 214–226, 2012.

[49] Colin R. Blyth. On simpson's paradox and the sure-thing principle. *Journal of the American Statistical Association*, 67(338):364–366, 1972. ISSN 01621459, 1537274X.

[50] Joy Buolamwini and Timnit Gebru. Gender shades: Intersectional accuracy disparities in commercial gender classification. In Sorelle A. Friedler and Christo Wilson, editors, *Proceedings of the 1st Conference on Fairness, Accountability and Transparency*, volume 81 of *Proceedings of Machine Learning Research*, pages 77–91. PMLR, 23–24 Feb 2018.

[51] Reuben Binns. On the apparent conflict between individual and group fairness. In *Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency*, pages 514–524, 2020.

[52] Dariusz Brzezinski, Julia Stachowiak, Jerzy Stefanowski, Izabela Szczech, Robert Susmaga, Sofya Aksenyuk, Uladzimir Ivashka, and Oleksandr Yasinskyi. Properties of fairness measures in the context of varying class imbalance and protected group ratios. *ACM Trans. Knowl. Discov. Data*, 18(7), June 2024. ISSN 1556-4681.

[53] Amalia Luque, Alejandro Carrasco, Alejandro Martín, and Ana de las Heras. The impact of class imbalance in classification performance metrics based on the binary confusion matrix. *Pattern Recognition*, 91:216–231, 2019. ISSN 0031-3203.

[54] Indre Zliobaite. On the relation between accuracy and fairness in binary classification. In *The 2nd workshop on Fairness, Accountability, and Transparency in Machine Learning (FATML) at ICML'15*, 2015.

[55] Sorelle A Friedler, Carlos Scheidegger, Suresh Venkatasubramanian, Sonam Choudhary, Evan P Hamilton, and Derek Roth. A comparative study of fairness-enhancing interventions in machine learning. In *Proceedings of the Conference on Fairness, Accountability, and Transparency*, pages 329–338, 2019.

[56] Max Hort, Zhenpeng Chen, Jie M Zhang, Mark Harman, and Federica Sarro. Bias mitigation for machine learning classifiers: A comprehensive survey. *ACM Journal on Responsible Computing*, 2023.

[57] Faisal Kamiran and Toon Calders. Classifying without discriminating. In *2nd International Conference on Computer, Control and Communication*, pages 1–6. IEEE, 2009.

[58] Michael Feldman, Sorelle A Friedler, John Moeller, Carlos Scheidegger, and Suresh Venkatasubramanian. Certifying and removing disparate impact. In *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 259–268, 2015.

[59] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, 16: 321–357, 2002.

[60] Toon Calders and Sicco Verwer. *Three naive Bayes approaches for discrimination-free classification*, volume 21, page 277–292. Kluwer Academic Publishers, USA, September 2010.

[61] Flavio Calmon, Dennis Wei, Bhanukiran Vinzamuri, Karthikeyan Natesan Ramamurthy, and Kush R Varshney. Optimized pre-processing for discrimination prevention. In *Advances in Neural Information Processing Systems*, pages 3992–4001, 2017.

[62] Harrison Edwards and Amos Storkey. Censoring representations with an adversary. In *4th International Conference on Learning Representations*, 2016.

[63] Christos Louizos, Kevin Swersky, Yujia Li, Max Welling, and Richard S. Zemel. The variational fair autoencoder. In Yoshua Bengio and Yann LeCun, editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016.

[64] Toshihiro Kamishima, Shotaro Akaho, Hideki Asoh, and Jun Sakuma. Fairness-aware classifier with prejudice remover regularizer. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 35–50. Springer, 2012.

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

[66] Brian Hu Zhang, Blake Lemoine, and Margaret Mitchell. Mitigating unwanted biases with adversarial learning. In *Proceedings of the 2018 AAAI/ACM Conference on AI, Ethics, and Society*, AIES '18, page 335–340, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360128.

[67] Sina Aghaei, Mohammad Javad Azizi, and Phebe Vayanos. Learning optimal and fair decision trees for non-discriminative decision-making. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 1418–1426. AAAI, 2019.

[68] Philip Adler, Casey Falk, Sorelle A Friedler, Tionney Nix, Gabriel Rybeck, Carlos Scheidegger, Brandon Smith, and Suresh Venkatasubramanian. Auditing black-box models for indirect influence. *Knowledge and Information Systems*, 54(1):95–122, 2018.

[69] Moritz Hardt, Eric Price, Eric Price, and Nati Srebro. Equality of opportunity in supervised learning. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016.

[70] Yash Savani, Colin White, and Naveen Sundar Govindarajulu. Intra-processing methods for debiasing neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M.F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 2798–2810. Curran Associates, Inc., 2020.

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

[72] Jerome H. Friedman. Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, 29(5):1189 – 1232, 2001.

[73] Christopher J. Anders, Leander Weber, David Neumann, Wojciech Samek, Klaus-Robert Müller, and Sebastian Lapuschkin. Finding and removing clever hans: Using explanation methods to debug and improve deep models. *Information Fusion*, 77:261–295, 2022. ISSN 1566-2535.

[74] Nora Belrose, David Schneider-Joseph, Shauli Ravfogel, Ryan Cotterell, Edward Raff, and Stella Biderman. Leace: perfect linear concept erasure in closed form. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 66044–66063, 2023.

[75] Ziwei Liu, Ping Luo, Xiaogang Wang, and Xiaoou Tang. Deep learning face attributes in the wild. In *Proceedings of International Conference on Computer Vision (ICCV)*, December 2015.

[76] Isabel O Gallegos, Ryan A Rossi, Joe Barrow, Md Mehrab Tanjim, Sungchul Kim, Franck Dernoncourt, Tong Yu, Ruiyi Zhang, and Nesreen K Ahmed. Bias and fairness in large language models: A survey. *Computational Linguistics*, pages 1–79, 2024.

APPENDICES

Appendix A

DetoxAI main interface

DetoxAI is a package available on PyPI and can be installed simply using `pip` in the standard way.

```
pip install detoxai
```

Listing A.1: Installing DetoxAI using pip

To use DetoxAI, one needs a standard `torch` model that inherits from the `torch.nn.Module` class. Additionally, a dataloader (`torch.utils.data.DataLoader`) must be prepared, returning batches of inputs, class labels, and protected attributes. These should be appropriately set via a collate function, as the dataloader will be used directly for debiasing.

A minimal code snippet required to run DetoxAI via the `.debias` interface is presented below. The snippet also demonstrates how to extract model copies with debiasing interventions applied, along with the corresponding performance and fairness metrics.

```
import detoxai

model = ...  # torch model
dataloader = ...  # torch dataloader returning (x, y, prot_attr) tuples

corrected = detoxai.debias(model, dataloader)

metrics = corrected.get_all_metrics()
model = corrected.get_model()
```

Listing A.2: Minimal example of using DetoxAI

Arguments of the `.debias` method The `.debias` method provides a wide range of parameters to customize the debiasing process. Below is a description of the most relevant parameters:

- `model`: The PyTorch model (`nn.Module`) to be debiased.
- `dataloader`: A `DetoxaiDataLoader` that provides batches of data, including input features, class labels, and protected attributes.
- `methods` (optional): Specifies the debiasing methods to run. Can be a list of method names or `"all"` (default). Supported methods include `SAVANIRP`, `SAVANILWO`, `SAVANIAFT`, `ZHANGM`, `RRCLARC`, `PCLARC`, `ACLARC`, `LEACE`, `NT`, and `FINETUNE`.
- `metrics` (optional): Specifies the performance and fairness metrics to compute. Can be a list of metric names or `"all"` (default). Supported metrics include: *Accuracy*, *Precision*, *Recall*, *Specificity*, *F1*, *GMean*, *Balanced_accuracy*, *Equal_opportunity*, *Equalized_odds*, *Demographic_parity*, *Accuracy_parity*.

- `methods_config` (optional): A structured dictionary providing detailed configuration for each correction method, by default it is empty and methods' default parameters are used. In order to view how to pass proper config, please refer to online documentation.
- `pareto_metrics` (optional): A list of metrics used to compute the Pareto front. Defaults to `["Balanced_accuracy", "Equalized_odds"]`.
- `return_type` (optional): Specifies the type of results to return:
  - `"pareto-front"` (default): Returns only the results on the Pareto front.
  - `"all"`: Returns the results of all correction methods.
  - `"best"`: Returns the best correction method based on the ideal point method from the Pareto front.
- `device` (optional): Specifies the device to run the correction methods on. Defaults to `"cpu"`.
- `include_vanila_in_results` (optional): If `True`, includes the vanilla model will be evaluated and included in the results. Defaults to `True`.
- `test_dataloader` (optional): A separate dataloader, which will be used just for evaluation. If not provided, the original `dataloader` is used.

The flexibility of these parameters allows practitioners to tailor DetoxAI to their specific use case, balancing fairness and performance according to their requirements.

Appendix B

DetoxAI software requirements

Python Version DetoxAI requires Python `>=3.11`. Verify that the appropriate Python version is installed in your environment before proceeding.

Dependencies The package depends on the following libraries:

- `gdown >=5.2.0`: Used for downloading pre-trained models and datasets from Google Drive.
- `lightning >=2.4.0`: A framework for simplifying PyTorch workflows and scaling model training.
- `torchvision >=0.20.1`: Provides datasets, transforms, and pre-trained models for computer vision tasks.
- `pillow >=11.0.0`: Used for image loading and processing.
- `tqdm >=4.67.1`: Provides a progress bar for iterative operations.
- `concept-erasure >=0.2.4`: Implements concept erasure bias mitigation.
- `scikit-learn >=1.6.0`: Provides utilities for CAV computations.
- `scikit-optimize >=0.10.2`: Used in *SavaniLWO*for zeroth-order optimization.
- `pandas >=2.2.3`: Facilitates data manipulation and management.

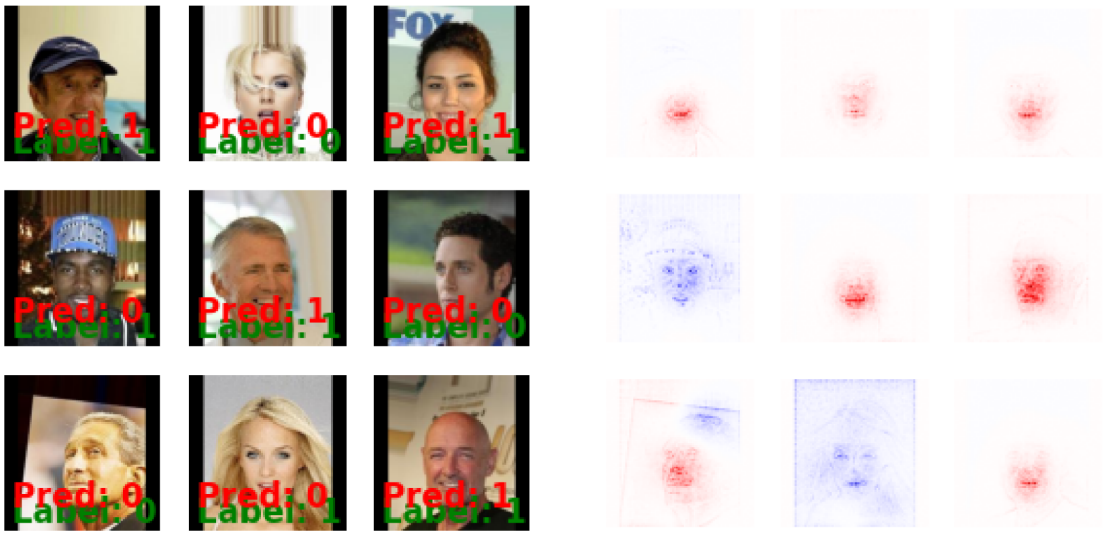
Build System DetoxAI uses the `hatchling` build system, which manages the package build process.

Appendix C

Additional visualizations with attribution maps

In the following, we present a few additional selected visualizations using attribution maps for the target attribute *Smiling* and protected attribute *Wearing Hat*.

(A) *Vanilla*



(B) *RR-ClArC*

(C) *SavaniAFT*

FIGURE C.1: Visualization of selected per-sample attribution maps for target *Smiling* and protected attribute *Weating Hat* CelebA variant. Images on the left have an overlay of the true label in green and the predicted label in red.

(A) *A-ClArC*

(B) *FineTune*

(C) *LEACE*

(D) *NT*

(E) *P-ClArC*

(F) *RR-ClArC*

(G) *SavaniAFT*

(H) *SavaniLWO*

(I) *SavaniRP*

(J) *Vanilla*

(K) *ZhangAL*

FIGURE C.2: Detailed method comparisons for target *Smiling* and protected attribute *Weating Hat* CelebA variant.

Appendix D

Additional quantitative visualizations

In the following Figures D.1 and D.2, we present a few selected per-method visualizations, extending experiments for RQ3 in Section 5.3.3.

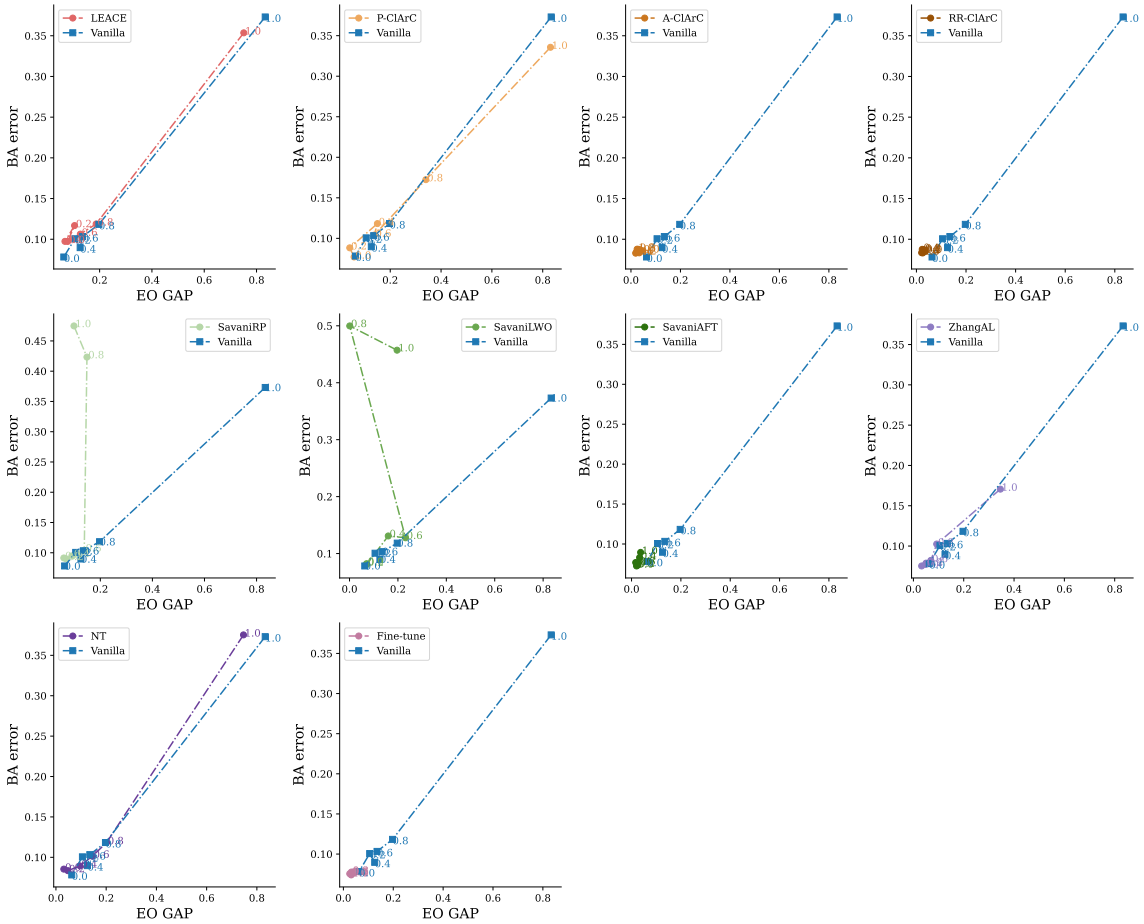


FIGURE D.1: This figure visualizes the per-method effect of the correlation coefficient in the training dataset, while having a balanced ($\phi = 0$) debiasing dataset. The number next to given data point corresponds to the ϕ coefficient in the training dataset.

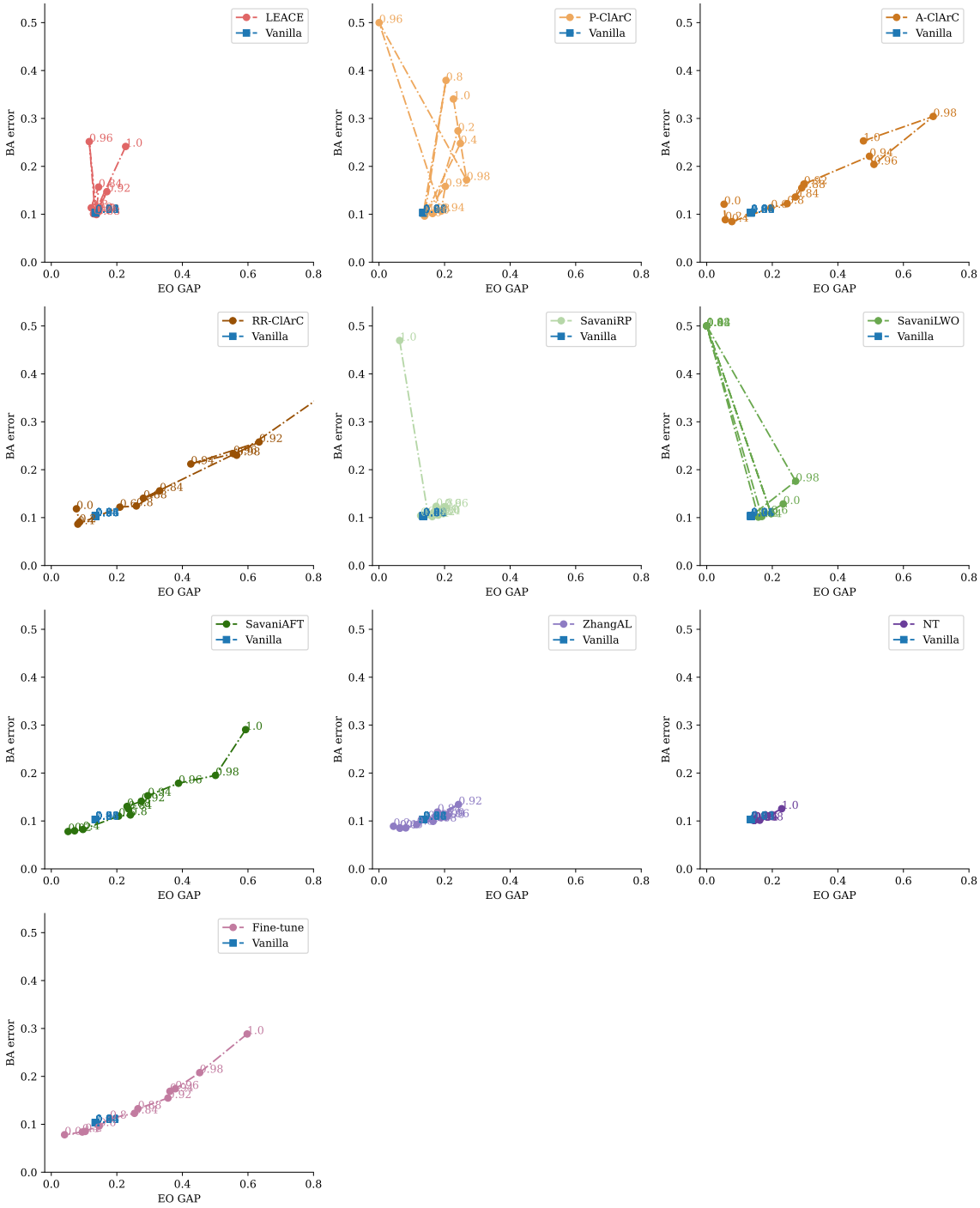


FIGURE D.2: This figure visualizes the per-method effect of the correlation coefficient in the debiasing dataset, while having a the training dataset fixed with $\phi = 0.6$. The number next to given data point corresponds to the ϕ coefficient in the debiasing dataset.

Appendix E

Comprehensive hyperparameter settings

In all methods which adjust classification threshold, we performed that operation on the last layer of the model.

NT

- `threshold_range`: $(0.05, 0.95)$
- `objective_function`: Optional[Callable[[float, float], float]]
- `threshold_steps`: 100
- `metric`: `"EO_GAP"`

FineTune

- `fine_tune_epochs`: 1
- `lr`: 1×10^{-4}

For all the following methods, Concept Activation Vectors (CAVs) were systematically extracted from the second-to-last layer of the model, i.e. the layer before squeezing the output to the target feature's dimensionality.

LEACE

- `use_n_examples`: $15,000$ (maximum samples to fit *LEACE*)

A-ClArC

- `fine_tune_epochs`: 1

RR-ClArC

- `fine_tune_epochs`: 1
- `ft_lr`: 1×10^{-3}

P-ClArC No additional parameters.

SavaniRP

- `epsilon`: 0.1
- `T_iters`: 15
- `bias_metric`: `BiasMetrics.EO_GAP`
- `optimizer_maxiter`: 100
- `tau_init`: 0.5
- `eval_batch_size`: 128
- `n_eval_batches`: 3
- `soft_thresh_temperature`: 100.0

SavaniLWO

- `epsilon`: 0.1
- `bias_metric`: `BiasMetrics.EO_GAP`
- `n_layers_to_optimize`: 4
- `thresh_optimizer_maxiter`: 100
- `beta`: 2.2
- `params_to_opt`: 0.5
- `never_more_than`: $50,000$
- `tau_init`: 0.5
- `n_eval_batches`: 3
- `eval_batch_size`: 128
- `skopt_verbose`: `True`
- `skopt_njobs`: 4
- `skopt_npoints`: 1000
- `skopt_maxiter`: 10
- `soft_thresh_temperature`: 100.0

SavaniAFT

- `epsilon`: 0.1
- `bias_metric`: `BiasMetrics.EO_GAP`
- `iterations`: 10
- `critic_iterations`: 5
- `model_iterations`: 5
- `train_batch_size`: 128
- `thresh_optimizer_maxiter`: 100
- `tau_init`: 0.5
- `lam`: 1.0
- `delta`: 0.01
- `critic_lr`: 1×10^{-4}
- `model_lr`: 1×10^{-4}
- `critic_filters`: $[8, 16, 32]$
- `critic_linear`: $[32]$
- `n_eval_batches`: 3
- `soft_thresh_temperature`: 100.0

Critic Architecture:

1. Convolutional layers: `nn.Conv2d(filters) -> nn.ReLU -> nn.MaxPool2d` (for each filter layer).
2. Adaptive Pooling: `nn.AdaptiveAvgPool2d(3)`.
3. Flattening: `nn.Flatten(dim=0)` (across batch dimensions as well).
4. Linear layers: `nn.Linear -> nn.ReLU` (for each linear layer).
5. Loss: `nn.MSELoss`.

ZhangAL

- `epsilon`: 0.1
- `bias_metric`: `BiasMetrics.EO_GAP`
- `iterations`: 5
- `critic_iterations`: 5
- `model_iterations`: 2
- `train_batch_size`: 128

- `thresh_optimizer_maxiter`: 100
- `tau_init`: 0.5
- `critic_lr`: 2×10^{-4}
- `model_lr`: 1×10^{-4}
- `critic_linear`: $[256, 256, 256]$
- `n_eval_batches`: 3
- `soft_thresh_temperature`: 100.0

Critic Architecture:

1. Linear layers: `nn.Linear -> nn.ReLU -> nn.Dropout(0.2)` (for each linear layer).
2. Loss: `nn.CrossEntropyLoss`.

Appendix F

Synthetic dataset for methods validation

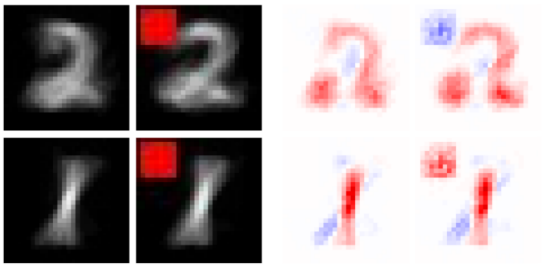
During the early stages of developing the DetoxAI toolkit, we utilized a specially crafted synthetic dataset to validate the implementation of bias mitigation algorithms. This step was essential, particularly for methods such as *RR-ClArC* and *A-ClArC*, which were originally designed for artifact unlearning rather than fairness. The synthetic dataset allowed us to systematically assess whether the relevance of a protected attribute (or concept) could be effectively minimized in the model's reasoning process after debiasing. We conducted this qualitative evaluation using the visualization tools for attribution maps integrated into DetoxAI.

The benchmarking dataset was derived from the widely recognized MNIST Digits dataset[1], which consists of grayscale images of handwritten digits. To introduce a controlled bias, we added a red square in the top-right corner of each image. The presence of the red square was correlated with specific classes of digits (1 and 2), creating an artificial bias. In the training set, the red square was perfectly correlated ($\phi = 1.0$) with the 1 class. In the unlearn set, the correlation was removed ($\phi = 0$). This setup allowed us to rigorously test whether the debiasing methods could effectively eliminate reliance on the red square (the protected attribute) in the model's predictions.

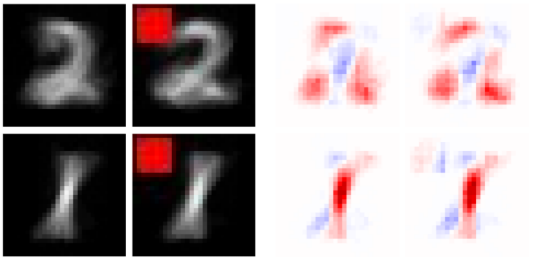
For this phase, we implemented a simple convolutional neural network (CNN) architecture to enable rapid experimentation given limited hardware resources. The CNN consisted of four convolutional blocks, each comprising a 2D convolutional layer (kernel size: 3), a ReLU activation function, and a pooling layer. This was followed by a fully connected layer and a final softmax activation for classification. This lightweight architecture was sufficient to detect and mitigate the artificially induced bias while facilitating fast iteration cycles.

The final implementation of several debiasing methods—including *RR-ClArC*, *A-ClArC*, *SavaniAFT*, and *SavaniLWO* - was evaluated alongside a reference *Vanilla* model and a baseline *FineTune* method. The results (Figure F.1) highlighted the following observation: the baseline model's attribution maps revealed a clear bias, with the model assigning significant relevance to the red square when predicting the class of the digit. This demonstrated the model's reliance on the protected attribute (red square) as part of its reasoning process. After applying the debiasing methods, the relevance of the red square in the attribution maps was substantially reduced. This qualitative evidence suggested that the protected attribute had been successfully removed from the model's reasoning process.

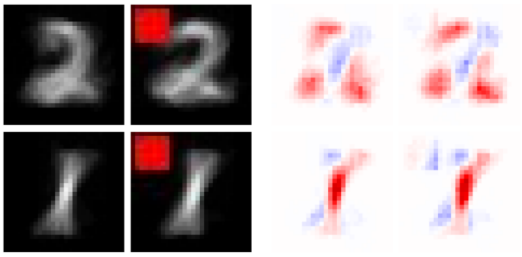
[1] `https://www.tensorflow.org/datasets/catalog/mnist`



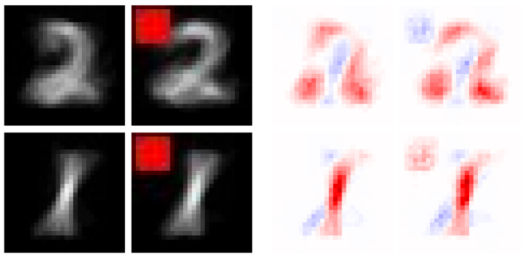
(A) Vanilla model



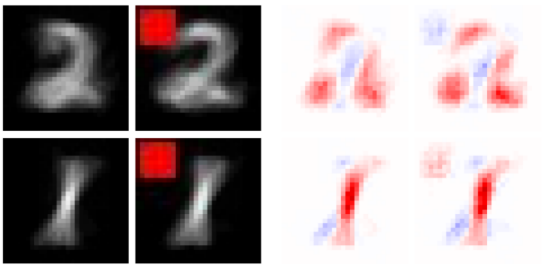
(B) *A-ClArC*



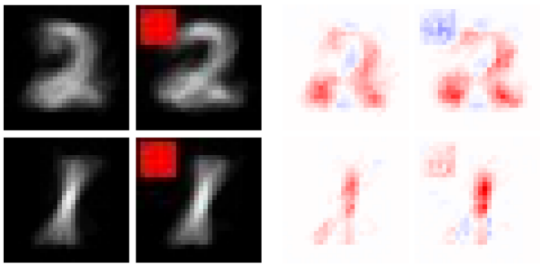
(C) *RR-ClArC*



(D) *FineTune*



(E) *SavaniAFT*



(F) *SavaniLWO*

FIGURE F.1: Attribution maps for selected debiasing methods on the MNIST dataset, modified with synthetic red square artifacts in the upper left corner of the images.

Poznan University of Technology
Faculty of Computing and Telecommunications
Institute of Computing Science

Typeset using LaTeX.