UNIVERSITÀ DEGLI STUDI DELL'AQUILA
DEPARTMENT OF INFORMATION ENGINEERING AND COMPUTER SCIENCE AND MATHEMATICS

DOCTORAL THESIS

Engineering Fair and Efficient Learning-Based Software Systems

Candidate:
Giordano D'ALOISIO

Advisor:
Prof. Antinisca DI MARCO

Co-Advisors:
Prof. Giovanni STILO
Prof. Davide DI RUSCIO

Doctoral Program Coordinator:
Prof. Davide DI RUSCIO

A thesis submitted in fulfillment of the requirements for the degree of Doctor of Philosophy in

Information and Communication Technology
Curriculum: Emerging computing models, software architectures, and intelligent systems

XXXVII Cycle

SSD INF/01

A.Y. 2023/2024

Abstract

Giordano D'ALOISIO

Engineering Fair and Efficient Learning-Based Software Systems

Learning-based systems – i.e., systems including machine learning (ML) models – are now employed in all aspects of our lives. The wide adoption of these systems has raised several concerns about their *quality*, as highlighted by the United Nations Sustainability Development Goals and the European Union AI Act. Unlike traditional software systems, learning-based systems employ an additional set of relevant quality properties (such as *fairness*, *explainability*, and *privacy*) that must be addressed. In this thesis, we focus on two of the most relevant quality properties of these systems—namely, *fairness* and *efficiency*—and propose a set of contributions that span different phases of a general learning-based systems development workflow.

Concerning the fairness quality property, we first address a significant lack of fairness-enhancing methods by proposing a novel pre-processing algorithm to improve fairness in both binary and multi-class classification settings. Next, we formally model the workflows for fairness assessment and select the best combination of ML model and fairness-enhancing method. We propose two low-code approaches leveraging these formal models to support data scientists in developing fair learning-based systems. Additionally, motivated by the desire to further support data scientists in the early identification of variables leading to high bias in a system, we performed an extensive empirical evaluation of the ability of dataset structural features—termed *bias symptoms*—to detect algorithmic bias early, before training a model. Finally, we begin to investigate bias issues of learning-based systems employing Large Language Models (LLMs) and how the fairness of these systems is currently assessed in GitHub projects.

Concerning the efficiency of learning-based systems, we first investigate the ability of existing approaches to estimate the training time of ML models early. This investigation is motivated by the need to assist data scientists in the early selection of ML models that meet a given training time constraint. Next, we examine the efficiency of LLMs regarding inference time and memory size. First, we conduct a thorough empirical investigation of the impact of LLM compression strategies on the efficiency and effectiveness of models fine-tuned for software engineering tasks. From this investigation, we derive a set of recommendations for practitioners and researchers to guide them in selecting the best compression strategy for a given task. Finally, we propose a novel search-based approach that identifies the optimal hyperparameter setting and prompt structure to reduce the inference time of text-to-image generation models while maintaining high quality in the generated images.

With this thesis, we aim to support data scientists and practitioners in developing fair and efficient learning-based systems and to help standardize some phases of the development workflow.

Acknowledgements

This dissertation would not have been possible without the support of all the people who shared this journey with me.

First and foremost, I would like to express my gratitude to my PhD advisor, Prof. Antinisca Di Marco. She gave me so much advice that helped me grow both scientifically and personally. It is also thanks to her that I am the person I am today.

I would also like to thank my PhD co-advisors, Prof. Giovanni Stilo and Prof. Davide Di Ruscio, for all the support they gave me and for all the opportunities they introduced me to.

Another thanks goes to Prof. Federica Sarro and all the members of the SOLAR and CREST research groups for welcoming me as a research visitor at UCL. I felt extremely honored to be part of these prestigious groups, and I am happy to have established a successful research collaboration.

I have been extremely lucky to be surrounded by many colleagues that, at the end of this journey, I can consider close friends.

First, I would like to thank my long-time friend Claudio for all the good times we spent together and for all the help he has given me over the years. I wish him the best in his career and his life.

Other big thanks go to my office mates (along with temporary additions) Mashal, Andrea D., Gennaro, Alina, Andrea M., Katiuscia, Diletta, and Alfonso. Some of them have been with me for a longer time, while others shared a smaller part of this journey with me, but all significantly contributed in some way. Thank you all for the moments we shared together. You made this journey much brighter.

I want to extend my thanks to all the other friends I did not mention before but who shared this path with me. Federico, Luca, Riccardo, Roberta, Gianluca, Andrea B., and Isabella are just a few. The list would be extremely long, but all of you (even the ones I did not explicitly mention – sorry!) will forever have my gratitude.

Finally, I want to express my gratitude to my family, who have always supported and trusted me in every decision I made.

Contents

List of Figures

List of Tables

List of Algorithms

Ai miei genitori e ai miei nonni

Chapter 1

Introduction

LEARNING-based software systems - i.e., systems that embed machine learning (ML) models - are nowadays employed in all application domains and affect our real life. If we consider the impact that those applications have in our lives, it is clear how ensuring that those systems are of *high* quality is of paramount importance. However, the specific nature of those systems introduce new quality properties (like *fairness*, *explainability*, or *privacy*) that are not common to traditional software systems [1]–[3].

The quality-based development of learning-based systems is a challenging task, given its data-centered and empirical nature [4], [5]. For this reason, different works have been proposed in recent years to formalize and address this topic. At the same time, many quality properties, metrics, and definitions can now be extracted from the literature [6]–[12]. However, the quality-based development of learning-based systems still presents different open research challenges. This research gap is even amplified by the introduction of Large Language Models (LLMs), which are even more affected by quality constraints like *efficiency* or *fairness* [13].

This thesis aims to address some of the open challenges that affect the quality-based development of learning-based systems. In particular, we focus on two relevant quality properties of those systems, namely *fairness* and *efficiency*. The choice of these properties is driven by their relevance, as highlighted by previous literature [2], [14]–[17], as well as by some of the 17 Sustainable Development Goals (SDG) proposed by the United Nations. In primis SDG 5 (Gender Equality), SDG 10 (Reduced Inequalities), SDG 12 (Responsible Consumption and Production), and SDG 13 (Climate Action) [18]. In addition, the recently introduced *AI Act* from the European Commission explicitly states that developers of *high-risk* systems (i.e., learning-based systems employed in sensitive domains like healthcare, education, or banking) must clearly describe their possible risks and mitigate them during the whole development process.[1]

Fairness is a property specifically introduced for learning-based systems [2]. It is defined as the absence of prejudice or discrimination of a system toward individuals or groups characterized by a set of legally protected *sensitive attributes* (e.g., their *ethnicity*, *gender*, or *religion*) [19]. If not adequately addressed, the inequalities reinforced by an unfair (or *biased*) learning-based systems can lead to severe societal consequences or reinforce existing discrimination [20]. For instance, we have highlighted in a previous study how historical data about academic promotions in the Italian Software Engineering (SE) and Informatics communities show a *gender* discrimination in the promotions from assistant to associate professor and from associate to full professor [21], [22]. Thus, a learning-based system trained on those data may reinforce this bias if not appropriately treated.

[1] `https://artificialintelligenceact.eu/`

By *efficiency*, we mean the amount of resources, like memory or time, required by a learning-based system for its operation [23]–[25]. This property is also considered in traditional software systems and affects the system's usability and performance [26]. However, it has regained considerable relevance in recent years in the context of *green* and *sustainable SE* [17]. Previous works have highlighted how the development and deployment process of learning-based systems is significantly resource-consuming [17], [23]–[25]. Hence, reducing the amount of resources required by those systems for their operations is essential to limit their environmental impact. The relevance of this property is even increased with the introduction of LLMs, which are extremely resource intensive [16], [17].

In the following, we describe in detail the contributions of this thesis in terms of identified challenges and how those have been addressed. In more detail, Section 1.1 presents the contributions of this thesis in the context of a general workflow to develop learning-based systems. Section 1.2 describes the list of publications on which this thesis is grounded. Finally, Section 1.3 presents the thesis's roadmap.

1.1 Thesis Contributions

Figure 1.1 exploits the thesis *contributions* (CN) in the context of a general workflow for the development of learning-based systems. Each contribution addresses a specific *challenge* (CH) through a given *result*. The presented workflow has been adapted from the previous work of Amershi *et al.* [4] and comprises several phases that are strictly connected. In addition, some phases can loop back to previous steps (as highlighted by the large arrows on top of the workflow). The first step in the workflow is *model requirement*, where the stakeholders define the functionalities that have to be implemented and the possible use cases of the system. In addition, a first selection of the possible ML models to include in the learning-based system is performed at this stage. The next four steps are related to the data *collection* and *engineering*. Based on the functionalities and use cases identified during the *requirement* phase, data are collected, cleaned, and labeled. Next, the data are further pre-processed during the *feature engineering* step to identify additional features useful for the later phases and remove additional noise. After the data are correctly engineered, they are used to train and evaluate the ML models selected during the *requirement* phase. The ML models are evaluated using a set of metrics selected during the *requirement* based on the use cases and functionalities that have to be performed. Note how the *model evaluation* phase may loop back to previous phases of the workflow to improve the overall quality of the system. When the evaluation phase provides satisfactory results, the system is deployed on the target platform and continuously monitored. The *model monitoring* phase may loop back to previous steps of the pipeline to address potential degradation in the system's quality. In the following, we describe the identified challenges. Finally, we present the thesis contributions and explain where each contribution could be placed in the context of a general learning-based system development pipeline.

In the following, we first present the identified challenges in Section 1.1.1. Next, we detail the contributions proposed to address each of them in Section 1.1.2.

1.1.1 Identified Challenges

Even though different works have been proposed in the context of improving the fairness and efficiency of learning-based systems, several challenges are still open.

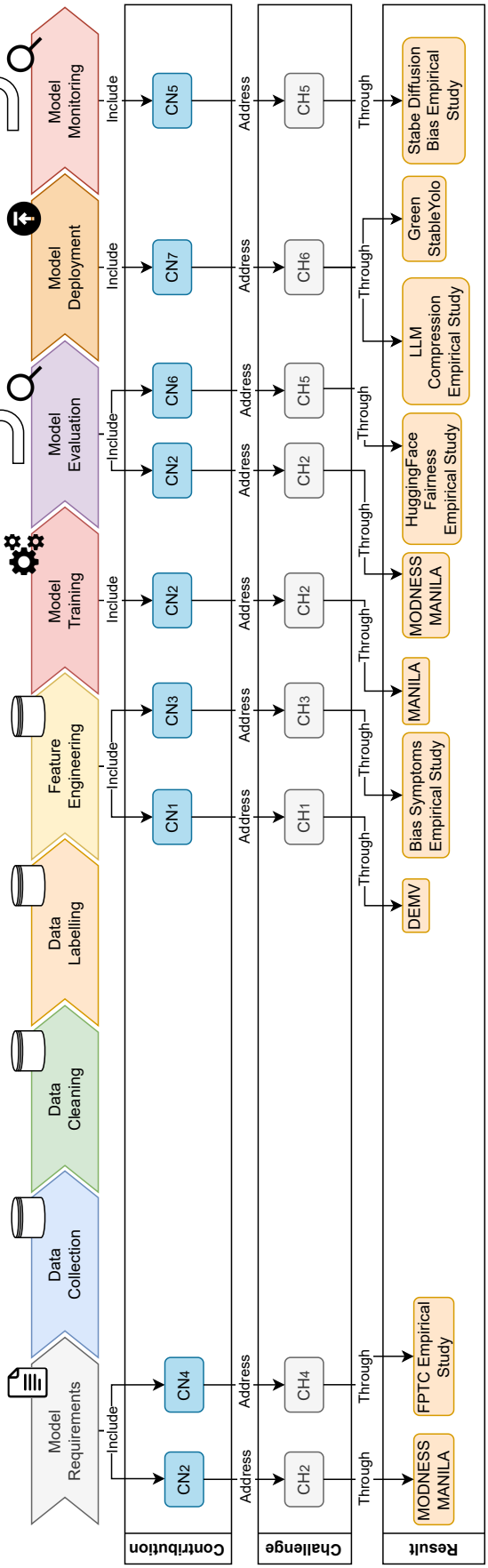


FIGURE 1.1: Thesis contributions in the context of a general workflow for the development of learning-based systems (adapted from [4]).

In the following, we describe the challenges that have been identified and addressed in this thesis:

CH$_1$ *Developing approaches for bias mitigation both in binary and multi-class classification settings.*

This challenge arises from the observation that most tools and libraries for bias mitigation, such as the widely used `aif360` [27] and `Fairlearn` [28], are primarily designed for binary classification settings — i.e., where the possible predictions of the ML model are limited to two outcomes. However, numerous studies describe learning-based systems that perform multi-class classification tasks — i.e., where the ML model can provide more than two possible predictions — in sensitive areas like *healthcare* [29] and *education* [30]. Ensuring fairness in learning-based systems employed in these contexts is essential.

CH$_2$ *Democratizing the development of fair learning-based systems to actors with different expertise.*

This challenge arises from analyzing traditional workflows used for assessing and improving the fairness of learning-based systems [19], [27], [28], [31]. Different stakeholders with varying areas of expertise are typically needed to define, measure, and address the bias these systems expose. However, most existing methods for measuring and mitigating bias require a deep understanding of bias and fairness concepts, as well as strong technical skills. This complexity makes integrating these methods into traditional workflows challenging since many actors may lack the necessary knowledge to use these tools effectively [32], [33]. Therefore, there is a need for approaches that can formalize and democratize the development of fair learning-based systems.

CH$_3$ *Investigating approaches for bias detection in early stages of a learning-based system development process.*

Many approaches have been proposed to promote automatic bias assessment [34]–[36]. However, all these techniques assess bias starting from the predictions of the ML model embedded in the learning-based system. Hence, they can only be performed after a model has been trained or deployed, which are late steps of a learning-based system development pipeline [4]. However, detecting early signals of bias in earlier phases of the development pipeline could help in tasks such as early identifying sensitive attributes (i.e., attributes that may lead to discrimination) or early bias mitigation [37].

CH$_4$ *Predicting a priori the training time of machine learning models could support early design decisions for learning-based systems development.*

Selecting the proper ML model to employ in a learning-based system is always a challenging task and difficult to engineer. The effectiveness of a particular ML model often depends on the data used and the specific use case [4], [5]. One important factor that can help stakeholders narrow down their options is the training time required for each model, particularly when computational resources are limited. Therefore, having an estimate of the training time in advance can help streamline certain phases of the learning-based system development process.

CH$_5$ *Highlighting the bias and the fairness assessment of learning-based systems embedding Large Language Models.*

After the release of ChatGPT in November 2022, Large Language Models (LLMs) have been employed in more and more aspects of our lives. Those models are generally pre-trained on a large set of heterogeneous data and then fine-tuned on an additional set of data for specific tasks [38]. However, being trained on a large set of data, these models are more subject to learning and expose possible bias. Moreover, their pre-trained nature makes the bias assessment and mitigation process more challenging. However, comprehensive analyses investigating the bias issues and the fairness assessment process of learning-based systems embedding LLMs are still missing.

CH$_6$ *Analyzing and improving the efficiency-effectiveness trade-off of resource-intensive Large Language Models.*

Transformer-based Large Language Models (LLMs) are effectively employed in numerous tasks nowadays. However, despite their impressive capabilities, the widespread adoption of these models is often hindered by practical challenges, particularly their high computational cost [39]. The deployment of LLMs typically requires computations across millions or even billions of learned parameters, resulting in significant memory demands and high inference time. To address this issue, AI researchers have developed various strategies over the past decade to reduce the size and computational cost of LLMs [40]–[42]. These strategies can reduce the memory demand of LLMs and/or speed up their inference times. However, their side effects concerning the LLMs' effectiveness (i.e., prediction correctness) are still not fully explored and addressed.

1.1.2 Contributions

Rounded blue boxes in Figure 1.1 represent individual contributions (CN), while the grey boxes indicate the challenges that each contribution addresses (CH). The orange boxes highlight the tools or studies used to tackle those challenges. It is important to note that some challenges are addressed by multiple contributions.

CN$_1$ *A novel approach for fairness improvement in binary and multi-class classification tasks [43], [44].*

To tackle CH$_1$, we propose the *Debiaser for Multiple Variables (DEMV)*. DEMV is an algorithm that can improve the fairness of a learning-based system in both binary and multi-class classification contexts by balancing the distribution of sensitive groups in the dataset [43], [44]. By working directly on the dataset, DEMV is model-agnostic, meaning that it can be applied to any classification method without influencing its behavior. This contribution can be applied to the *feature engineering* phase of the pipeline depicted in Figure 1.1 when the dataset is further pre-processed before training an ML model. DEMV is available as a package in the PyPi Python repository[2] and as a method in the SoBigData RI [45]. This contribution is described in depth in Chapter 4.

CN$_2$ *Conceptualization and development of low-code frameworks to support the development of fair and effective learning-based systems [46], [47].*

To address CH$_2$, we formalized the pipelines to assess and improve the fairness of a learning-based system. From the engineering of those pipelines, we

[2] `https://pypi.org/project/demv/`

derive two low-code applications. The first tool is *MANILA*, a low-code application to benchmark the effectiveness (i.e., prediction correctness) and fairness of different ML models and fairness-enhancing methods combinations [46]. It is based on the Extended Feature Models (ExtFM) formalism, which models a generic pipeline for the evaluation of ML models and fairness-enhancing methods as a Software Product Line (SPL) [48]. MANILA supports data scientists during the *model requirements* phase by assisting them in selecting the different features required to evaluate ML models and fairness-enhancing methods. In addition, by automatically training and testing the selected configurations, MANILA supports data scientists during the *model training* and *evaluation* phases. MANILA is available as an application in the SoBigData RI [45][3].

The second proposed application is *MODNESS*, a model-driven framework to automate the fairness assessment process of learning-based system [47]. It features a domain-specific language (DSL) that enables stakeholders to specify their definitions of fairness and related metrics and generates a Python implementation of the modeled analysis. Like MANILA, MODNESS also supports domain experts in the specification of high-level bias definitions during the *model requirements* phase. In addition, by automatically generating the Python implementation, MODNESS assists data scientists during the *model evaluation* phase. We release the source code of MODNESS publicly [49].

Although they have been developed and presented as different applications, MANILA and MODNESS are designed to be integrated in the future to guide data scientists in the development of fair learning-based systems while providing a high degree of expressiveness in specifying fairness definitions and metrics.

Chapter 5 describes the modeling formalism behind both approaches, while Chapter 6 describes their technical implementations and evaluations.

CN$_3$ *An extensive empirical analysis on the ability of datasets bias symptoms to early predict algorithmic fairness.*

To tackle CH$_3$, we performed an extensive empirical analysis of the ability of datasets' structural features (namely *bias symptoms*) to perform early detection and explanation of algorithmic bias, i.e., bias inducted by the ML model. We extract those symptoms using binary variables from 24 datasets well-known in the fairness literature. Next, we use them to detect early bias signals in a dataset. The rationale for analyzing bias symptoms is manifold. First, we aim to assess to what extent bias symptoms can be employed during the *feature engineering* phase to detect signals of bias before training an ML model, allowing the early identification of variables that could lead to *high* bias. Secondly, bias symptoms could be employed to explain why a particular variable may lead to a specific type of bias in a system. The empirical analysis is described in Chapter 7, and the replication package is publicly available [50].

CN$_4$ *An empirical study on the effectiveness of approaches to estimate the training time of traditional machine learning models [51].*

To address CH$_4$, we perform an extensive empirical evaluation of the Full Parameter Time Complexity (FPTC) approach proposed by Zheng *et al.* [52]. This approach is, to the best of our knowledge, the only method so far that formulates the ML training time as a function of the dataset's and ML model's

[3] `https://sobigdata.d4science.org/group/sobigdata.it/manila-univaq`

parameters. Thus, it could be employed during the *model requirements* phase to select ML models that satisfy specific constraints related to their training time. In addition, it could be integrated into MANILA to further assist data scientists in selecting ML models that are below a given training time threshold. Those motivations led us to analyze in depth the effectiveness of this approach. Specifically, we use the FPTC to predict the training time of a Logistic Regression [53] and Random Forest [54] classifier on a heterogeneous number of data. Next, we compare the predicted time with the actual training time of the method and highlight the main strengths and weaknesses of the approach. This contribution is presented in Chapter 10, and the replication package is publicly available [55].

CN_5 *A preliminary empirical analysis on the bias exposed by Image Generation Models towards Software Engineering tasks.*

To tackle CH_5 we perform an extensive empirical study of the *gender* and *ethnicity* bias exposed by three versions of the open source Stable Diffusion (SD) text-to-image generation model – Stable Diffusion 2 (SD 2) [56], Stable Diffusion XL (SD XL) [57], and Stable Diffusion 3 (SD 3) [58] – towards SE tasks. We chose Stable Diffusion as a reference model since, due to its open-source nature, it is nowadays the most adopted text-to-image generation model. From a survey conducted by the Everypixel company, around 80% of all artificially generated images in 2023 were from systems embedding Stable Diffusion models [59]. Following previous works [60], [61], we ask each SD version to generate images for 56 software-related tasks using two different prompt styles: one style including the "*Software Engineer*" keyword and one with no role specification. We collect a total of 6,720 images and compare the *gender* and *ethnicity* bias exposed by each SD version in generating images with a specific prompt style. Results show that including the "*Software Engineer*" keyword significantly increases the bias exposed by those models. Following this evaluation, we provide recommendations for practitioners and researchers to address and mitigate the bias exposed by those deployed models during the *model monitoring* phase. The results obtained from this study drive the motivation for further research toward bias mitigation in image-generation models.

The study is reported in Chapter 8, and the replication package is available at [62].

CN_6 *A preliminary study on the coupled usage of pre-trained Large Language Models and fairness assessment libraries.*

Another contribution addressing CH_5 is a preliminary study exploring how publicly trained models (PTMs) stored on Hugging Face (HF) are currently utilized within the prominent open-source software ecosystem, GitHub. Our focus is on identifying PTMs that could be integrated with fairness assessment libraries, which are dedicated tools and frameworks that support fairness assessment during the *model evaluation* phase. To conduct this analysis, we utilize the latest HF dump provided by the HF community project [63] and concentrate on PTMs that support classification tasks, including those related to text, tokens, images, and tabular data. We then examine the GitHub platform to explore how these models are being used, gathering relevant metadata such as repository content, stars, forks, and the source code found in Python files. Additionally, we look for key fairness-related terms and the import statements

associated with three notable fairness assessment libraries: AIF360 [27], Fairlearn [28], and Fairkit-learn [64].

From our initial set of PTMs supporting classification tasks, we discovered that only a small number have been utilized on GitHub. More importantly, none of these models include references to the three fairness libraries mentioned above. This finding underscores the need for further research in this area. We view our study as an initial step toward understanding how PTMs can be integrated into the fairness assessment process, thereby revealing various research opportunities.

This contribution is described in Chapter 8, and the replication package is publicly available at [65].

CN$_7$ *An empirical study on the effectiveness of compression strategies for Large Language Models of Code.*

Finally, CH$_6$ is also addressed by two contributions, which both apply to the *model deployment* phase. In the first contribution, we investigate the impact of different model compression strategies across three SE tasks: vulnerability detection (*code classification*), code summarization (*code-to-text generation*), and code search (*text-to-code recommendation*). We fine-tune a well-known large language model for code, CodeBERT [66], on each of these tasks. Subsequently, we assess how three widely-adopted model compression strategies – namely, knowledge distillation [40], quantization [67], and pruning [68] – affect (i) the effectiveness of the LLM in performing the task, (ii) inference time, and (iii) the model's memory size. Our results provide practitioners and researchers with guidelines on balancing the trade-offs between effectiveness and efficiency when selecting a model compression strategy.

This contribution is described in Chapter 11, and the replication package of this experiment is publicly available at [69].

CN$_8$ *A novel approach to improve inference time and image quality of Image Generation Models [70].*

The second contribution proposed to address CH$_6$ is *GreenStableYolo*, a novel approach that addresses the challenge of optimizing the trade-off between inference time and image quality of Stable Diffusion models [70]. By using a search-based multi-objective optimization algorithm – i.e., the Non-dominated Sorting Genetic Algorithm (NSGA-II) [71] – GreenStableYolo finds the best hyperparameters and prompt structure able to improve the quality of the generated images while reducing the time for their generation. We provide initial empirical evidence that, by using GreenStableYolo, Stable Diffusion (SD) models achieve a satisfactory equilibrium between inference time and image quality, making it suitable to be employed during the *model monitoring* phase to increase the efficiency of deployed SD models.

The approach and its evaluation are described in Chapter 11. GreenStableYolo is publicly available at [72].

As mentioned above, DEMV and MANILA have been deployed, respectively, as a method and application into the SoBigData RI. SoBigData is a European project that aims to develop and share analyses and tools in the field of Big Data following the Open Science principle. Its RI is built on top of the D4Science platform [73] and provides datasets, analyses, methods, and applications on several data science

topics. In this perspective, following open-science rules, both approaches may be reused and integrated with other items provided in the RI for future research [45]. In the future, we plan to include additional contributions presented in this thesis to this infrastructure.

1.2 List of Publications

All the contributions of this thesis are based on publications submitted to scientific conferences or journals. The complete list of journal, conference, and workshop publications is reported below in chronological order.

Journal Papers

J1. d'Aloisio, G., Di Sipio, C., Di Marco, A., & Di Ruscio, D. (2025). Towards Early Detection of Algorithmic Bias from Dataset's Bias Symptoms: An Empirical Study. Information and Software Technology (under review). Preprint: `http://dx.doi.org/10.2139/ssrn.5143265`

This paper presents the empirical study on dataset bias symptoms reported in Chapter 7.

J2. d' Aloisio, G., Di Sipio, C., Di Marco, A. & Di Ruscio, D. (2025). How fair are we? From conceptualization to automated assessment of fairness definitions. International Journal of Software and Systems Modeling. `https://doi.org/10.1007/s10270-025-01277-2`

This paper presents the MODNESS tool, which is described in Chapters 5 and 6.

J3. D'Angelo, A., d'Aloisio, G., Marzi, F., Di Marco, A., & Stilo, G. (2024). Uncovering gender gap in academia: A comprehensive analysis within the software engineering community. Journal of Systems and Software, 217, 112162. `https://doi.org/10.1016/j.jss.2024.112162`.

This paper describes an empirical analysis of the gender gap in academic promotions in Italian SE and Informatics communities and is used as a motivating example in Chapter 1.

J4. d'Aloisio, G., D'Angelo, A., Di Marco, A., & Stilo, G. (2023). Debiaser for multiple variables to enhance fairness in classification tasks. Information Processing & Management, 60(2), 103226. `https://doi.org/10.1016/j.ipm.2022.103226`.

This paper describes the DEMV algorithm presented in Chapter 4.

Conference Papers

C1. d'Aloisio, G. (2025). MANILA: A Low-Code Application to Benchmark Machine Learning Models and Fairness-Enhancing Methods. *International Conference on the Foundations of Software Engineering (FSE) - Demonstration Track.*

This paper describes the technical implementation of the MANILA web-application presented in Chapter 6.

C2. d'Aloisio, G., Traini, L., Sarro, F., & Di Marco, A. (2025). On the compression of language models for code: An empirical study on CodeBERT. *International Conference on Software Analysis, Evolution, and Reengineering (SANER 2025)*. https://doi.org/10.48550/arXiv.2412.13737.

This paper describes the empirical analysis of compression strategies for large language models of code presented in Chapter 11.

C3. Gong, J., Li, S., d'Aloisio, G., Ding, Z., Ye, Y., Langdon, W. B., & Sarro, F. (2024). GreenStableYolo: Optimizing inference time and image quality of text-to-image generation. In *International Symposium on Search Based Software Engineering*. Springer Nature Switzerland Cham, pp. 70–76. **SSBSE 2024 Challenge Track Winner.** https://doi.org/10.1007/978-3-031-64573-0_7

This paper presents the GreenStableYolo algorithm described in Chapter 11.

C4. d'Aloisio, G., Di Marco, A., & Stilo, G. (2023). Democratizing quality-based machine learning development through extended feature models. In L. Lambers & S. Uchitel (Eds.), *Fundamental Approaches to Software Engineering - 26th International Conference, FASE 2023, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2023, Paris, France, April 22-27, 2023, Proceedings* (Vol. 13991, pp. 88–110). Springer. https://doi.org/10.1007/978-3-031-30826-0_5

This paper presents the extended feature model and the first version of MANILA. It is referred in Chapters 5 and 6.

C5. d'Aloisio, G. (2022). Quality-driven machine learning-based data science pipeline realization: A software engineering approach. In *44th IEEE/ACM International Conference on Software Engineering: Companion Proceedings, ICSE Companion 2022, Pittsburgh, PA, USA, May 22-24, 2022* (pp. 291–293). ACM/IEEE. https://doi.org/10.1145/3510454.3517067

This paper presents the idea of quality-based development of learning-based systems, which is the core topic of this thesis. In addition, it provides the first contributions of MANILA described in Chapters 5 and 6.

Workshop Papers

W1. Fadahunsi, T., d'Aloisio, G., Di Marco, A., & Sarro, F. (2025). How Do Generative Models Draw a Software Engineer? A Case Study on Stable Diffusion Bias. In *1st International Workshop on Fairness in Software Systems*, co-located with *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. **Best Paper Award**. https://doi.org/10.48550/arXiv.2501.09014

This paper presents the empirical study conducted to analyze the bias exposed by Stable Diffusion models towards SE tasks described in Chapter 8.

W2. d'Aloisio, G., D'Angelo, A., Marzi, F., Di Marco, D., Stilo, G., & Di Marco, A. (2023). Data-driven analysis of gender fairness in the software engineering academic landscape. In B. Tekinerdogan, R. Spalazzese, H. Sözer, S. Bonfanti, & D. Weyns (Eds.), *Software Architecture. ECSA 2023 Tracks, Workshops, and Doctoral Symposium - Istanbul, Turkey, September 18-22, 2023, Revised Selected Papers* (Vol. 14590, pp. 89–103). Springer. https://doi.org/10.1007/978-3-031-66326-0_6

This paper describes the first results of the empirical analysis of gender bias in academic promotions used as a motivating example in Chapter 1.

W3. Marzi, F., d'Aloisio, G., Di Marco, A., & Stilo, G. (2023). Towards a prediction of machine learning training time to support continuous learning systems development. In B. Tekinerdogan, R. Spalazzese, H. Sözer, S. Bonfanti, & D. Weyns (Eds.), *Software Architecture. ECSA 2023 Tracks, Workshops, and Doctoral Symposium - Istanbul, Turkey, September 18-22, 2023, Revised Selected Papers* (Vol. 14590, pp. 169–184). Springer. https://doi.org/10.1007/978-3-031-66326-0_11

This paper describes the empirical study of the FPTC approach presented in Chapter 10.

W4. d'Aloisio, G., Stilo, G., Di Marco, A., & D'Angelo, A. (2022). Enhancing fairness in classification tasks with multiple variables: A data- and model-agnostic approach. In L. Boratto, S. Faralli, M. Marras, & G. Stilo (Eds.), *Advances in Bias and Fairness in Information Retrieval - Third International Workshop, BIAS 2022, Stavanger, Norway, April 10, 2022, Revised Selected Papers* (Vol. 1610, pp. 117–129). Springer. https://doi.org/10.1007/978-3-031-09316-6_11

This paper introduces the first version of the DEMV algorithm described in Chapter 4

Publications not included in this thesis

J1. Di Ludovico, D., Capannolo, C., & d'Aloisio, G. (2023). The toolkit disaster preparedness for pre-disaster planning. International Journal of Disaster Risk Reduction, 96, 103889. https://doi.org/10.1016/j.ijdrr.2023.103889.

This paper describes a toolkit of recommendations for pre-disaster planning.

C1. D'Angelo, A., & d'Aloisio, G. (2024). Grammar-based anomaly detection of microservice systems execution traces. In S. Balsamo, W. J. Knottenbelt, C. L. Abad, & W. Shang (Eds.), *Companion of the 15th ACM/SPEC International Conference on Performance Engineering, ICPE 2024, London, United Kingdom, May 7-11, 2024* (pp. 77–81). ACM. **Best ICPE Data Challenge Award.** https://doi.org/10.1145/3629527.3651844.

This paper presents a grammar-based approach for anomaly detection in microservice systems execution traces.

C2. Palomba, F., Di Sorbo, A., Di Ruscio, D., Ferrucci, F., Catolino, G., Giordano, G., Di Dario, D., Voria, G., Pentangelo, V., Tortorella, M., Sgueglia, A., Di Sipio, C., d'Aloisio, G., & Di Marco, A. (2024). FRINGE: Context-aware FaiRness engineerING in complex software systEms. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '24)* (pp. 608–612). Association for Computing Machinery. https://doi.org/10.1145/3674805.3695394

This paper presents the FRINGE project, which aims to engineer the development of fair software systems.

C3. d'Aloisio, G., Fortz, S., Hanna, C., Fortunato, D., Bensoussan, A., Mendiluze Usandizaga, E., & Sarro, F. (2024). Exploring LLM-driven explanations for quantum algorithms. In *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (pp. 475–481). Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3674805.3690753

This paper describes an empirical study of the ability of LLMs to explain quantum algorithms.

C4. d'Aloisio, G., et al. (2024). Engineering a digital twin for diagnosis and treatment of multiple sclerosis. In *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems (MODELS Companion '24)* (pp. 364–369). Association for Computing Machinery, New York, NY, USA. `https://doi.org/10.1145/3652620.3688249`

This paper describes the initial architecture of a digital twin for the diagnosis and treatment of multiple sclerosis.

C5. Bianchi, A., d'Aloisio, G., et al. (2022). DIORAMA: Digital twin for sustainable territorial management. In M. Anisetti, A. Bonifati, N. Bena, C. A. Ardagna, & D. Malerba (Eds.), *Proceedings of the 1st Italian Conference on Big Data and Data Science (itaDATA 2022), Milan, Italy, September 20-21, 2022* (Vol. 3340, pp. 144–155). CEUR-WS.org. Available at: `https://ceur-ws.org/Vol-3340/paper43.pdf`

This paper presents the architecture of a digital twin for sustainable territorial management.

W1. Bianchi, A., d'Aloisio, G., Marzi, F., & Di Marco, A. (2023). A decision tree to shepherd scientists through data retrievability. In *Second Workshop on Reproducibility and Replication of Research Results (RRRR 2023)*. `https://doi.org/10.48550/ARXIV.2304.05767`

This paper presents a decision tree to support scientistic through data retrievability.

W2. Caroccia, F., D'Agostino, D., d'Aloisio, G., Di Marco, A., & Stilo, G. (2021). SismaDL: An ontology to represent post-disaster regulation. In P. Forbrig, K. Hinkelmann, M. Kirikova, B. Lantow, C. Møller, A. Morichetta, P. Plebani, B. Re, K. Sandkuhl, & U. Seigerroth (Eds.), *Joint Proceedings of the BIR 2021 Workshops and Doctoral Consortium Co-located with 20th International Conference on Perspectives in Business Informatics Research (BIR 2021), Vienna, Austria, September 22-24, 2021* (Vol. 2991, pp. 99–112). CEUR-WS.org. Available at: `https://ceur-ws.org/Vol-2991/paper09.pdf`

This paper presents an ontology to represent and query post-disaster regulations.

1.3 Thesis Outline

This thesis is divided into three parts:

- Part I provides all the contributions related to **Fairness of Learning-Based Systems**. Chapter 2 provides background knowledge on the key concepts related to fairness and introduces the two general workflows for fairness assessment and to evaluate fairness-enhancing methods mentioned throughout this thesis part. Chapter 3 describes related work to the concepts presented in this thesis part. Chapter 4 describes the DEMV algorithm. Chapter 5 presents the formal modeling of the two workflows for fairness assessment and to identify the best ML model and fairness-enhancing methods. Chapter 6 describes the two low-code approaches leveraging the formal models. Chapter 7 presents the empirical study on dataset bias symptoms. Finally, Chapter 8 shows initial insights into fairness issues about LLMs.

- Part II is devoted to describe the contributions related to the **Efficiency of Learning-Based Systems**. Chapter 9 reports work related to the concepts presented in this thesis part. Chapter 10 describes the initial evaluation of approaches to early estimate the training time of machine learning models. Finally, Chapter 11 presents the empirical evaluation of LLM compression methods and the GreenStableYolo algorithm.
- Part III describes the **Conclusions** of this thesis and future works.

Part I

Fairness of Learning-Based Systems

Chapter 2

Background Knowledge

In this chapter, we recall the main concepts of fairness assessment and mitigation and describe a general process for those tasks.

When defining bias and conducting fairness analysis, it is important to consider the specific domain being studied [19], [31]. Thus, we introduce two case studies that we use throughout this chapter to illustrate the fundamental concepts of the underlying workflows. These examples are deliberately chosen from significantly different domains to highlight the heterogeneity of key fairness concepts:

1. **University Admission (*University*).** It was originally presented in [74] and involves a university that uses an ML-based system to determine student admissions. It is necessary to ensure that the system is fair. Specifically, the concern is whether or not the system is biased against *women*.

2. **Popularity Bias in TPL Recommendation (*TPL*).** It concerns the problem of *popularity bias* in Third-Party Library (TPL) recommendations [75]. TPL recommendation concerns the development of recommender systems that can suggest TPL to developers based on the software they are developing. In the original paper, authors highlight how very popular libraries (i.e., with high *"reputation"*) are more likely to be recommended mainly because several developers are using them. On the contrary, more specific or recent libraries are less likely to be recommended, even if they are more appropriate to the development task at hand.

The rest of this chapter is structured as follows: Section 2.1 highlights key concepts and a general workflow for fairness assessment while Section 2.2 presents the main concepts and a general workflow to evaluate fairness-enhancing methods. Finally, Section 2.3 concludes the chapter by motivating the need to formalize and develop low-code approaches to support the development of fair learning-based systems.

2.1 Fairness Assessment: Key Concepts and a General Workflow

Figure 2.1 illustrates a general fairness assessment workflow, highlighting the main concepts at each step. Additionally, the top of the figure shows the main actors involved in each step, while the bottom provides an instantiation of the key concepts for the *University* use case. The workflow has been derived by analyzing the behavior of some of the most adopted fairness toolkits and libraries (i.e., Aequitas [76], IBM AIF360 [27], and Fairlearn [28]) and by reviewing foundational papers on bias and fairness [14], [15], [19]. Recalling the general workflow for the development of

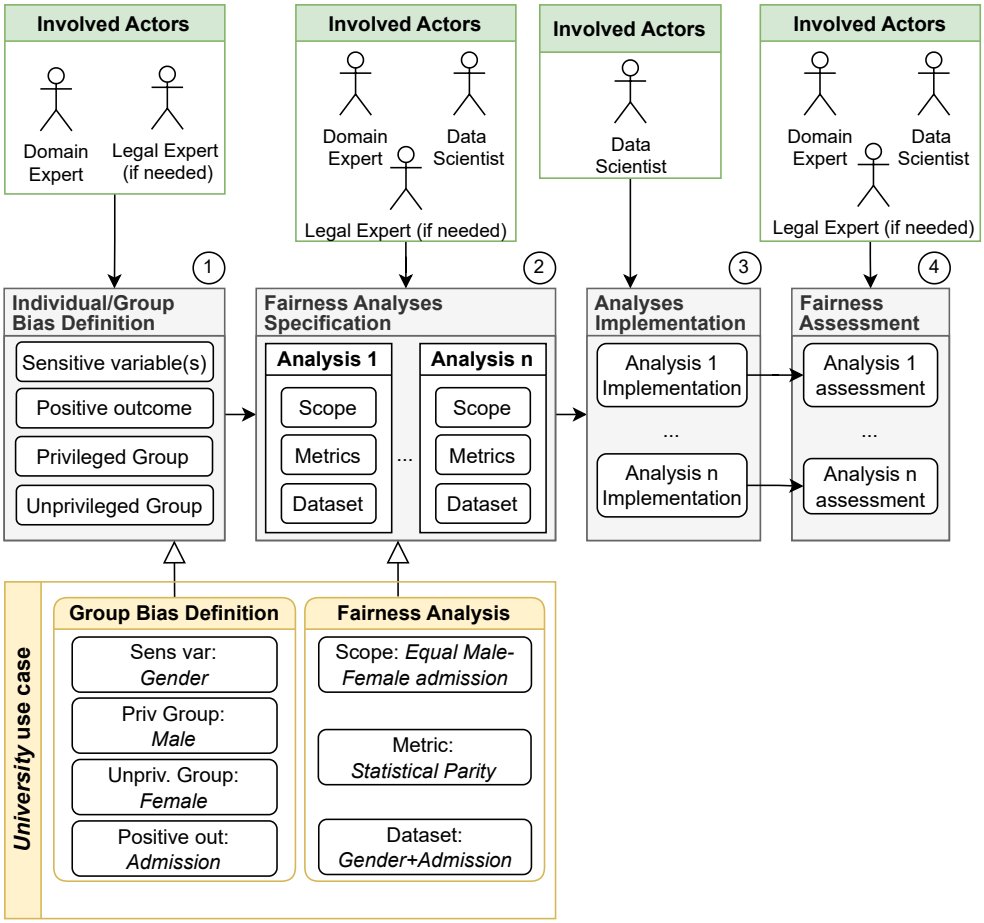


FIGURE 2.1: General fairness assessment workflow. On top, there are the main actors involved in each step, while on the bottom, there is the instantiation of the key concepts for the *Univerisity* use case.

learning-based systems, the depicted process usually may span through the *model requirements*, *model evaluation*, and *model monitoring* phases.

The fairness assessment process may involve multiple parties, with three key actors typically identified:

- **Domain expert** provides domain-specific knowledge (in our examples, the person in charge of managing admissions to the university or the developer using the recommender system);
- **Data scientist** provides knowledge about fairness metrics and their implementation (in our examples, the data scientist provides knowledge about methods and metrics to assess possible bias in the university admission and recommender systems, based on the bias definitions provided by the domain expert);
- **Legal expert** provides knowledge about specific regulations if needed in the fairness assessment process (in our examples, the legal expert provides knowledge about possible regulations concerning university admissions or recommender systems for particular domains).

2.1.1 Bias Definition

The first step in the workflow depicted in Fig. 2.1 is the *bias definition*, which involves both the domain and legal experts (step ① in the Figure). In general, bias can be of two types: [19], [31]:

- **Group bias:** if discrimination or favoritism is assessed at the group level (e.g., all the *women* for the *University* use case, or all the *less popular* libraries for the *TPL* use case). **Individual bias:** if discrimination or favoritism is assessed for any individual, regardless of their belonging to a specific group. For instance, in our use cases, a student must not be admitted to the university only because she is a relative of the university rector, or a library must not be recommended only based on previous user preferences (regardless of its relevance).

It is worth mentioning that, even if considered by the literature, use cases regarding *individual bias* are less common and, in general, *individual bias* is more difficult to mitigate compared with *group bias* [15], [19], [77].

To establish a definition of bias within a specific domain, it is crucial to pinpoint the following key concepts, as outlined in [19]:

- **Sensitive variables:** these are the variables that have the potential to lead to discrimination;
- **Positive outcome:** This refers to a specific prediction generated by the ML system that could potentially result in discriminatory consequences. Note that this concept is domain-dependent and may change based on the perspective from which we observe a given use case (e.g., a prediction about a customer positively subscribing to a bank term deposit may be good for the business but not for the customer's wallet)
- **Privileged** and **Unprivileged** groups: These are the groups or entities identified based on specific values of the sensitive variables. Privileged groups may be favored by the system, while unprivileged groups may face discrimination. Note how these two groups are often referred to as *sensitive groups*, i.e., groups that must be protected from discrimination or favoritism (e.g., by some regulations).

Concerning the *University* use case (as reported at the bottom of Fig. 2.1), domain and legal experts want to assess if the ML system under analysis discriminates against women. Thus, the type of examined bias is *group bias*, with the sensitive variable being *gender*. A positive outcome would be *successful admission* to the university, and the privileged group is *men* while the unprivileged one is *women*. For the *TPL* use case, domain experts want to ensure that not only popular libraries are recommended. Hence, the type of bias is still *group bias*, where the sensitive variable is *popularity*. A positive outcome is a *recommendation* from the system. The privileged group is *popular libraries* while the unprivileged group is *unpopular libraries*.

2.1.2 Fairness Analysis and Metrics

From an abstract bias definition, several fairness analyses can be depicted (step ② in Fig 2.1). In general, a fairness analysis has a *scope*, a set of *metrics* compliant with the given scope (which can be fairness metrics known in the literature or can be custom ones), and has a *dataset*, which contains all the information needed to perform the given analysis.

There are at least 26 different definitions and related metrics of fairness proposed by the literature over the years [19], [78]. However, previous work also highlighted how most metrics correlate or provide the same information [79]. In the following, we describe the most common definitions and metrics of fairness adopted in previous work [15], [77], [80].

Statistical (Demographic) Parity (SP) is one of the first definitions of *group fairness* proposed by the literature [81]. It assumes the *independence* among the predicted positive label and the sensitive variables. It is defined formally as follows:

Definition 1 (Statistical Parity) *Let $\hat{Y}$ be the predicted value, y_p the positive label, and S a generic binary sensitive variable where $S = 1$ and $S = 0$ identify, respectively, the privileged and unprivileged groups. A predictor is fair under Statistical Parity if:*

$$P(\hat{Y} = y_p|S = 1) = P(\hat{Y} = y_p|S = 0)$$

The standard metric corresponding to this definition is the *Statistical Parity Difference*, which measures the difference among those probabilities:

$$SP = P(\hat{Y} = y_p|S = 1) - P(\hat{Y} = y_p|S = 0) \tag{2.1}$$

This metric ranges from 0 to $|1|$, where 0 highlights fairness.

A different formulation for SP is *Disparate Impact (DI)* [82], which considers the ratio among the two probabilities. In this case, following the *80% rule*, the value must be between 0.8 and 1.2 to have *fairness* [82]. DI is defined formally as follows:

Definition 2 (Disparate Impact) *Let $\hat{Y}$ be the predicted value, y_p the positive label, and S a generic binary sensitive variable where $S = 1$ and $S = 0$ identify the privileged and unprivileged groups, respectively. A predictor is fair under Disparate Impact if:*

$$0.8 \leq \frac{P(\hat{Y} = y_p|S = 1)}{P(\hat{Y} = y_p|S = 0)} \leq 1.2 \tag{2.2}$$

Note how both SP and DI formulations do not consider the relation between the model's predictions and the ground truth values. Instead, they only assess the independence between sensitive variables and positive predictions. Thus, they have been classified as *independence* definitions [78].

Differently, definitions belonging to the *separation* category consider the relation between the model's outcome and the ground truth values [78].

Equal Opportunity (EO) [83] is a *separation* definition that considers the probability of obtaining the positive prediction for instances belonging to the sensitive groups given a positive value of the ground truth label. It is formally defined as follows:

Definition 3 (Equal Opportunity) *Let $\hat{Y}$ be the predicted outcome, Y the ground true value, y_p the positive label, and S a generic binary sensitive variable where $S = 1$ and $S = 0$ identify the privileged and unprivileged groups, respectively. A predictor is fair under Equal Opportunity if:*

$$P(\hat{Y} = y_p|Y = y_p, S = 1) = P(\hat{Y} = y_p|Y = y_p, S = 0)$$

The metric corresponding to this definition is the *Equal Opportunity Difference*, which, like SP, computes the difference among those probabilities:

$$EO = P(\hat{Y} = y_p|Y = y_p, S = 1) - P(\hat{Y} = y_p|Y = y_p, S = 0) \tag{2.3}$$

Again, this metric ranges from 0 to $|1|$, where 0 means fairness.

Finally, *Equalized Odds* [83] is an extension of EO that assesses if the probability of an item to be positively classified is the same with respect to the sensitive variable and any possible ground truth value. It is formally defined as follows:

Definition 4 (Equalized Odds) *Let $\hat{Y}$ be the predicted value, Y the true value, y_p the positive label, and S a generic binary sensitive variable where $S = 1$ and $S = 0$ identify the privileged and unprivileged groups, respectively. A predictor is fair under Equalized Odds if:*

$$P(\hat{Y} = y_p | Y = y, S = 1) = P(\hat{Y} = y_p | Y = y, S = 0) \;\; \forall y \in \{y_1, \ldots, y_n\}$$

The metric corresponding to this definition is the *Average Odds Difference (AO)*, which computes the average of the difference among those probabilities:

$$AO = \frac{\begin{array}{r}[P(\hat{Y} = y_p | Y = y_1, S = 1) - P(\hat{Y} = y_p | Y = y_1, S = 0)] + \ldots \\ + [P(\hat{Y} = y_p | Y = y_n, S = 1) - P(\hat{Y} = y_p | Y = y_n, S = 0)]\end{array}}{n} \quad (2.4)$$

A value of 0 for this metric highlights bias.

All these definitions were initially proposed for binary classification problems ($y_p = 1$). Still, they can be easily extended to the multi-class classification domain by identifying one positive label value among the possible ones ($y_p \in \{y_1, \ldots, y_n\}$).

Recalling the presented use cases, assume that in the *University* scenario (see Fig. 2.1), the domain and legal experts decide that the ML system is fair if *men* and *women* have the same probability of being admitted to the university. This is the scope of the analysis. Hence, the data scientist suggests using the *Statistical Parity* fairness metric, which is compliant with the given fairness definition. Finally, the domain expert and the data scientist collect all the needed information (i.e., the predictions of the ML model and the gender of each person) inside a dataset that will be used for the analysis. It is important to mention that other analyses can be conducted. For example, domain and legal experts may want to examine whether someone is only admitted to a university based on their high school grades, regardless of gender. Hence, other metrics (e.g., *EO*) can be used to cover this fairness definition, and further information has to be provided in the dataset.

For the *TPL* use case, domain experts can state that a recommender system is free from popularity bias if relevant libraries are recommended despite their popularity [75]. To assess this definition, they adopt a variation of the *Coverage* metric, which measures the percentage of *non-popular* items recommended over the total recommendations [84]. Note how this metric does not come from the fairness literature but is an adaptation of a metric from the recommender systems domain to assess popularity bias [75]. Finally, like for the *University* use case, a dataset containing the recommended libraries and their popularity is collected to perform the fairness assessment.

2.1.3 Fairness Evaluation

The next step in the fairness assessment process is the implementation of the defined fairness analyses (step ③ in Fig. 2.1). Concretely, this means implementing an automatic procedure that, given a specific dataset as input, computes all the selected metrics and returns the calculated values. In our examples, the data scientist has to implement software using a programming language (e.g., Python) that takes as input the dataset and computes the *Statistical Parity* or *Coverage* fairness metrics.

Once the results are computed, all parties must evaluate them to determine the system's fairness (step ④ in Fig. 2.1). For example, in our scenarios, the data scientist notes that a SP score of zero or a *Coverage* score of one indicates fairness. Next, legal and domain experts analyze the results to assess whether the ML system is fair based on this definition.

The process depicted in Fig. 2.1 and described earlier can be challenging and prone to mistakes, mainly because it involves several stakeholders and interconnected tasks. To address these issues, we introduce MODNESS, a model-driven-based approach that automates the assessment of fairness. This approach enables users to define their notions of bias and fairness, enlarging the applicability of fairness assessment to multiple application domains. We present the details of this approach in Chapters 5 and 6.

2.2 Bias Mitigation: Key Concepts and a General Workflow

This section presents the key concepts related to bias mitigation and describes a general workflow to evaluate the effectiveness of fairness-enhancing methods.

2.2.1 Background on Fairness-Enhancing Methods

Over the years, many methods have been proposed to mitigate bias at different phases of a learning-based system development workflow [19], [31]. We distinguish among [85]:

- **Pre-processing** methods. Those approaches can be applied during the *feature engineering* phase to reduce the underlying bias in data before training an ML model. Examples of those models are presented in [82], [86];
- **In-processing** methods, which are employed during the *model training* phase to change the learning process to remove discrimination. Widely adopted examples are reported in [87], [88];
- **Post-processing** methods, which can be applied during *model evaluation* phase to re-calibrate an already trained model to reduce the learned bias. Standard models of this category are [83], [89].

In general, the sooner a technique can be applied, the better because it can be chained with other bias mitigation methods in the later processing phases [27], [90].

Among the different ML tasks, classification has been the task most addressed for bias mitigation [19], [31].

Most of the methods available in the literature focus solely on binary classification with one sensitive variable [19]. Among them, one widely adopted *pre-processing* method is the *Sampling* algorithm proposed by [86]. This method balances both privileged and unprivileged users in the case of binary classification with a single sensitive variable. Formally, let be S the sensitive variable with $\{w, b\} \in S$ representing the privileged and unprivileged groups, respectively, and let be Y the target label with $\{+, -\} \in Y$ defining the positive and negative outcomes. The *Sampling* algorithm first splits the original dataset into four groups:

- Deprived group with Positive label (DP): all instances with $S = b \wedge Y = +$;
- Deprived group with Negative label (DN): all instances with $S = b \wedge Y = -$;

- Favored group with Positive label (FP): all instances with $S = w \wedge Y = +$;
- Favored group with Negative label (FN): all instances with $S = w \wedge Y = -$.

Then, for each group, the algorithm computes its *observed* and *expected* sizes. Finally, it balances the groups iteratively by randomly adding and removing instances until the *observed* sizes of the groups are equal to their *expected* ones.

The *Sampling* algorithm is the starting point for the definition of the DEMV algorithm described in Chapter 4. In fact, we have extended this algorithm to the multi-class classification domain with multiple sensitive variables, and we have employed different instance-generation strategies during the balancing process.

Very few methods are able to mitigate the bias in the multi-class classification domain [88], [89]. Among those, we mention the *Blackbox post-processing* algorithm proposed by Putzel *et al.* [89]. This method extends the *Equality of Odds* algorithm [83] to the multi-class classification setting. It involves the construction of a linear program over the conditional probabilities of the adjusted predictor $P(Y^{adj} = y^{adj} | \hat{Y} = \hat{y}, A = a)$ such that the desired fairness criterion is satisfied by those probabilities. To build the linear program, the authors formulate both the loss and fairness criteria as linear constraints of the protected attribute conditional probability matrices. Then, this linear program is used to find the label value, among the possible ones, that minimizes both the loss and the fairness constraints.

An *in-processing* method that solves unfairness in multi-class classification settings is the one presented by Agarwal *et al.* [88]. This algorithm addresses two definitions of fairness at once: *Demographic Parity* and *Equalized Odds*. The authors formulate such definitions as linear constraints and then build an Exponentiated Gradient (EG) reduction algorithm [91] that yields a randomized classifier with the lowest error subject to the desired fairness constraints. The method follows a MinMax approach in which the players try to minimize the given constraint and maximize the classifier's score. The authors also propose a simplified Grid Search version of the algorithm (GRID), which generates a sequence of labeling and weights and trains a predictor for each one. The values yielding the best *accuracy* and *fairness* trade-off are selected and thus returned. Although the authors study their algorithms mainly in binary classification problems, they also show how their method can be applied to regression and multi-classification problems.

To the best of our knowledge, most of the methods in the literature are primarily designed for binary classification problems, and few of them can be applied in the *pre-processing* phase. Moreover, we identified a few approaches realized to mitigate bias in multi-class classification problems, and none works in the pre-processing phase.

2.2.2 Workflow for Benchmarking Fairness-Enhancing Methods

Algorithm 1 presents the pseudo-code for a generic process used to train and evaluate various combinations of machine learning models and fairness-enhancing methods. It is important to note that the process outlined here complements the fairness assessment workflow described in Section 2.2. Specifically, this process assumes that domain experts and data scientists have already established a high-level definition of bias, have mapped the sensitive groups and the positive outcomes to a concrete dataset, and have selected a set of metrics compliant with the given bias definition. As for the fairness assessment workflow, also this process has been derived by analyzing the behavior of the most adopted fairness-related libraries (i.e., IBM AIF360

Algorithm 1: Fairness-Enhancing Methods Benchmarking Process

Input: Dataset d, ML Algorithms ML, Fairness Methods F, Effectiveness and Fairness Metrics M

```
1  for m ∈ ML do
2      for q ∈ Q do
3          if q works on d then
4              Apply q on d;
5          if q works on m before training then
6              Apply q on m;
7          f = train m;
8          if q works on f then
9              Apply q on f;
10         Compute selected metrics M on f;
11 Evaluate the results;
12 Q = Select best Fairness Method;
13 M = Select best ML Algorithm;
14 M* = Train M with full dataset applying Q;
15 return M*
```

[27] and Fairlearn [28]) and by reviewing foundational papers on bias and fairness [14], [15], [19], [77].

Recalling the general workflow for the development of learning-based systems presented in Figure 1.1, the depicted process may span during the *feature engineering*, *model training*, and *model evaluation* phases. Thus, we assume that the data scientist has already collected and cleaned the dataset to use and identified the set of ML models, fairness-enhancing methods, and metrics more suited for the use case. For each of the chosen ML algorithms, the data scientist applies the selected fairness-enhancing methods according to their category (lines from 3 to 9 in Algorithm 1)[77]:

- if it is a *pre-processing* method, it has to be applied to the dataset during the *feature engineering* phase before training the ML algorithm [43], [82], [86];
- if it is an *in-processing* method, it has to be applied during the *model training* phase [87], [88];
- if it is a *post-processing* method, it can be applied in any phase after a first *model training* phase (e.g., on an already deployed model [92] or on an additional *training* phase [93]).

After training an ML model and applying a fairness-enhancing method, the data scientist computes the selected metrics to assess the fairness and effectiveness of the specific combination (line 10 in Algorithm 1). After repeating the process for all the combinations of ML model and fairness-enhancing methods, the data scientist selects the combination achieving the best fairness-effectiveness trade-off (lines from 11 to 15 in Algorithm 1). If the data scientist has a threshold to achieve, then they can verify if at least one of the combinations satisfies the constraint. If so, one of the suitable pairs is selected. Otherwise, they have to relax the threshold and repeat the process.

From the workflow described in Algorithm 1, we extract a set of general steps that a data scientist has to perform. Figure 2.2 sketches such a generalization. First,

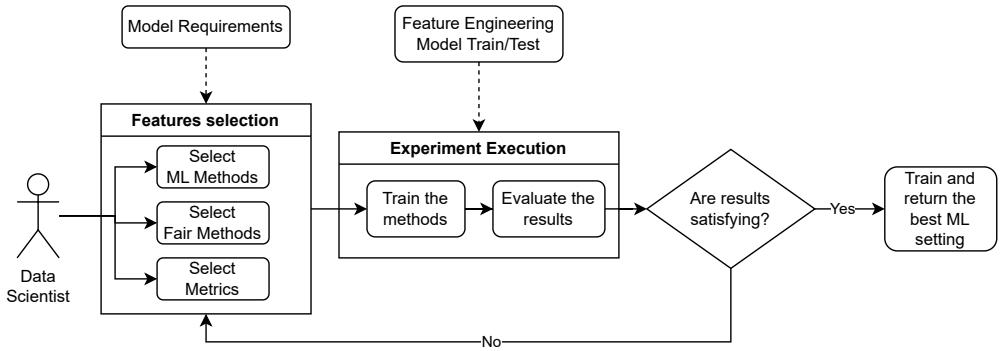


FIGURE 2.2: Execution of the Fairness-Enhancing Methods Benchmarking Process

the data scientist selects all the experiment features, i.e., the dataset, the ML models, the fairness-enhancing methods, and the related metrics (*Features Selection* step). Those steps happen during the *model requirements* phase. Next, the data scientist benchmarks the different ML models and fairness-enhancing methods combinations using the general approach described in algorithm 1 (*Experiment Execution* in Figure 2.2). This step spans through the *feature engineering*, *model training* and *model evaluation* phases. If the results are satisfying (i.e., they satisfy a given threshold), then the best combination is returned. Otherwise, the data scientist has to repeat the process, possibly relaxing the constraints.

It is worth mentioning that the selection of ML models and fairness-enhancing methods for evaluation is closely related to the specific task and use case. Additionally, the choice of ML models and fairness-enhancing methods may vary depending on whether the system is intended for deployment or is already in use. For a system that is still being developed, different ML models can be assessed. Conversely, for a system that is already deployed, various fairness-enhancing methods should be tested on the same ML model. For instance, recalling the *University* and *TPL* examples introduced at the beginning of this chapter, assume that, for the *University* use case, the system needs to be deployed. Thus, the data scientist may select all ML models used for classification and combine them with all *pre-, in-, and post-processing* fairness-enhancing methods. Eventually, they will select the best combination. On the contrary, assume that for the *TPL* use-case, the system is already deployed, but a fairness issue has been detected during the *model monitoring* phase. Hence, to avoid retraining the model from scratch, the data scientist may select all *post-processing* fairness-enhancing methods and test them with the deployed ML model.[1] Ultimately, they can choose the best post-processing technique.

2.3 Conclusion

This chapter presented the key concepts related to fairness assessment and bias mitigation. Additionally, we outlined two general workflows for defining and assessing fairness, as well as evaluating different methods that enhance fairness. Although these two processes have been presented separately, they are strictly related and

[1]It is relevant to note that the model used for testing alongside these methods does not necessarily have to be the deployed model; it could also be a new model of the same type that has been trained on the same data. This design decision may also depend on the domain [15].

span through many phases of a general learning-based systems development workflow (see Figure 1.1).

Those processes can be challenging and prone to mistakes, mainly because they involve several stakeholders and interconnected tasks. Moreover, not all fairness-enhancing methods can be applied to all ML models. This may create errors and inconsistencies during the fairness-enhancing methods evaluation workflow. As shown in our review, existing approaches for fairness assessment are strictly tied to specific fairness definitions, and there is no low-code approach that guides the different stakeholders through the developments of fair learning-based systems. In addition, most bias mitigation approaches are for binary classification, while relevant multi-class classification tasks are still poorly covered.

To address these issues, we first introduce in Chapter 4 a pre-processing algorithm able to mitigate bias in binary and multi-class classification tasks with multiple sensitive variables. Next, we introduce in Chapter 5 two formal models of the fairness assessment and bias mitigation workflows introduced in this chapter. Those formal models are the foundations of two low-code approaches presented in Chapter 6. Finally, we present in Chapter 7 the results of a first attempt towards the early detection of algorithmic bias (i.e., bias in the predictions of an ML model) starting from datasets' structural features (i.e., *bias symptoms*).

Chapter 3

Related Work on Fairness

This chapter discusses existing works related to the concepts presented in this part of the thesis. Section 3.1 presents a survey of existing approaches for fairness assessment, highlighting the main features provided. Section 3.2 describes existing approaches related to early bias assessment. Finally,Section 3.3 discusses previous works on bias in text-to-image generation models and Section 3.4 presents related studies on model repositories.

3.1 Review of Existing Approaches for Fairness Assessment

This section presents the procedure exploited to gather pertinent literature within the realm of fairness assessment. First, we describe the adopted procedure in Section 3.1.1, including the search string and the inclusion and exclusion criteria. Subsequently, Section 3.1.2 delves into an exploration of key features and sub-features constituting the fairness assessment workflow that we use to classify the selected approaches, while Section 3.1.3 provides an overview and classification of the collected approaches.

3.1.1 Methodology

In this section, we provide an overview of prominent approaches within the domain of bias and fairness assessment in ML-based systems focusing on the SE community. Please note that our aim is not to present a comprehensive survey of the entire field, as it goes beyond the scope of this section. Instead, we have adopted a tool-supported procedure inspired by the well-established "four W-question strategy" [94] to select existing approaches that perform bias detection using automated or tool-supported methods. In particular, our analysis is confined to peer-reviewed scientific works that make significant contributions in two key areas: *i)* defining or addressing fairness concerns within software systems, and *ii)* employing automated methods to mitigate identified biases while considering notable datasets.

The four W-questions guiding our approach are as follows:

- *Which?* We conducted a comprehensive search, combining both automated and manual methods, to gather relevant papers from a variety of sources, including conferences and journals.
- *Where?* Our literature analysis focused on prominent software engineering venues, encompassing ten conferences: ASE, ESEC/FSE, ESEM, ICSE, ICSME, ICST, ISSTA, MSR, SANER, and MODELS as well as five journals: EMSE, IST, JSS, TOSEM, and TSE. In particular, we collect relevant information for those venues, i.e., title and abstract, that we used in the filtering process. To automate

this process, we utilized the *Scopus* database[1] and employed advanced search and export functions to retrieve all papers published in specific venues within the temporal range we decided.

- *What?* For each article, we extracted information from the title and abstract by applying predefined keywords to ensure relevance to our research focus.
- *When?* Given that automated fairness assessment is a relatively recent research area, our search was limited to the most recent five years, spanning from 2017 to 2023. This temporal constraint allowed us to capture the latest developments and trends in the field. It is worth noticing how the query was executed in April of 2024, hence 2024 has not been considered.

TABLE 3.1: Number of papers for the related topics.

	FAIR	**ML**	**TOOL**
FAIR	241	-	-
ML	**51**	1,931	-
TOOL	**56**	123	3,438

We export the relevant papers from Scopus and exploit dedicated Python scripts to search in title and abstract the following set of keywords in *AND* conjunction:

(i) **FAIR**: "*fairness*" or "*bias*"; *(ii)* **ML**: "*data science*" or "*machine learning*; *(iii)* **TOOL**: "*toolkit*," or "*definition*", or "*audit*", or "*testing*" or "*model-based*". Table 3.1 reports the number of papers that contain such keywords in the corresponding column and row (e.g., 123 papers contain at least one term belonging to ML and TOOL sets). Our ideal targets are papers containing at least one keyword for all the defined sets of terms, i.e., FAIR, ML, and TOOL. By running such a combination, we obtain only 15 works considering the abovementioned criteria. Therefore, we enlarged the set of eligible papers to two additional combinations highlighted in bold, *(i)* FAIR and ML; or *(ii)* FAIR and TOOL. In the end, we obtained a total of 107 papers, including duplicate papers. By removing those ones, we ended up with 61 scientific papers, including journal and conference publications.

Starting from this initial set of works, we manually inspected the title and abstract to scale down the search to meet our requirements. In particular, we defined the following inclusion and exclusion criteria:

✔ **Inclusion criteria:** We included all the approaches that use traditional bias definitions, i.e., group or individual. Furthermore, we consider toolkits or frameworks that provide an automatic or semi-automatic strategy to assess fairness on a set of use cases. Some of them have been published as extensions of initial works. Therefore, we consider the most recent version of the tool in such cases.

✖ **Exclusion criteria:** The study we conducted intentionally excludes foundational papers that primarily provide a high-level abstract definition of fairness, such as surveys [19], [31], empirical studies [80], or position papers [14]. Additionally, we excluded papers focusing on improving the fairness of underlying ML models [43], [86], as our focus is on automating the assessment process through the explicit specification of the application domain.

[1] `https://scopus.com`

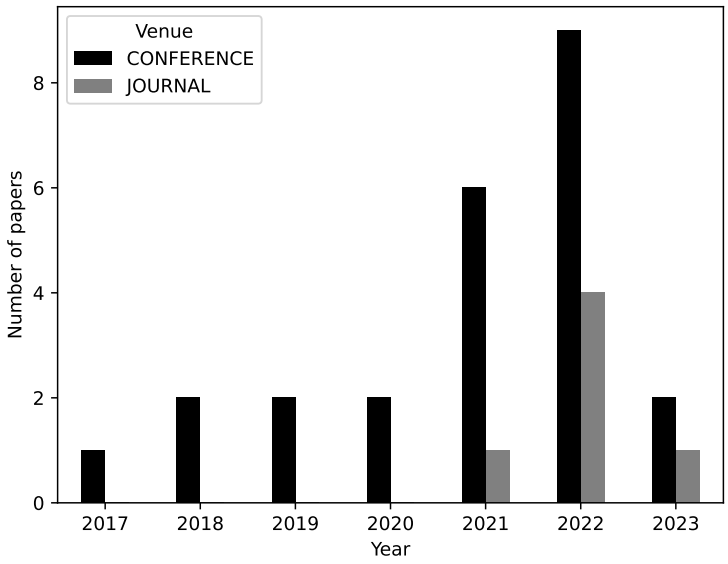


FIGURE 3.1: Number of selected papers per year.

To ensure an unbiased selection process, we employed a rigorous approach. Two different people independently evaluated all the papers, and the two senior researchers thoroughly reviewed the entire selection process. Ultimately, this meticulous process yielded a total of 26 works.[2] Figure 3.1 depicts the retrieved papers divided by year. Notably, there is an increasing trend with a peak in 2022, indicating that automating fairness assessment is becoming increasingly relevant in the SE community. The rising number of journal publications confirms this, suggesting that researchers share more mature results than the initial studies that appeared in 2017. However, there is a noticeable decrease in the number of published papers in 2023, although this trend represents frameworks selected using the abovementioned procedure. Therefore, it is not representative of the whole trend in SE concerning fairness assessment.

3.1.2 Elicited features

Table 3.2 summarises the list of papers we collected by means of the previously described process. For each approach, we list the name of the tool, the venue, the year, and the underpinning mechanism used to define and assess fairness (if any). Furthermore, starting from the four steps of the general fairness assessment workflow described in Section 2.2, we elicit six different features to evaluate the degree of automation and customization of the selected approaches. These features are grouped in Table 3.2 by the primary step of the workflow described in Figure 2.1 they belong to, i.e., *bias definition*, *fairness analysis specification*, *analysis implementation & fairness assessment*. The selected features are as follows:

➤ **F1 - Bias definition**: The approach models and assesses individual bias definitions, group bias definitions, or both;

[2]The complete list of selected and excluded papers is available at the following link `https://github.com/giordanoDaloisio/MODNESS/blob/main/Selected_papers.md`

➤ **F2 - Abstract bias definition**: The approach implements an extension mechanism to provide a bias definition that is tailored for a specific domain and agnostic from a specific fairness analysis or dataset (see steps ① and ② of Fig 2.1);
➤ **F3 - Custom metric definition**: Similar to the previous one, the approach allows the definition of additional metrics to detect bias (i.e., metrics for less common use cases like *TPL*);
➤ **F4 - Metric composition**: It is possible to combine defined metrics to create new ones, for instance, by means of aggregation functions;
➤ **F5 - Automated fairness assessment**: The underlying system assesses the fairness by automatically generating the corresponding source code;
➤ **F6 - Tool availability**: The paper is supported by a publicity available tool;

For each tool, we marked these features with *supported* (✔) in Table 3.2 while we left blank unsupported features. Concerning the feature *fairness definition*, the symbols Ⓘ and Ⓖ are used for *individual* and *group* bias, respectively.

TABLE 3.2: Comparison of the existing fairness toolkit and approaches.

			Bias Definition		Fairness Analysis Specification		Analysis Implementation & Fairness Assessment	
Approach	**Venue & Year**	**Base strategy**	**F1 - Bias def.**	**F2 - Abstract bias def.**	**F3 - Custom metric def.**	**F4 - Metric Comp.**	**F5 - Automated fairness assess.**	**F6 - Tool avail.**
Aequitas [76]	ASE (2018)	Search-based	Ⓖ				✔	✔
Themis [95]	ESEC/FSE (2018)	Search-based	Ⓖ		✔		✔	✔
TILE [96]	ICST (2019)	Metamorphic testing	Ⓘ				✔	
ADF [97]	ICSE (2020)	Adversarial DL	Ⓘ					✔
Fairway [98]	ESEC/FSE (2020)	Search-based	Ⓖ		✔		✔	✔
DeepInspect [99]	ICSE (2020)	Deep learning	Ⓖ				✔	✔
AITEST [100]	ICSE (2021)	Search-based	Ⓘ				✔	
EIDG [101]	ISSTA (2021)	Search-based	Ⓘ					✔
Fair-SMOTE [102]	ESEC/FSE (2021)	Situation testing	Ⓘ,Ⓖ				✔	✔
Biswas and Rajan [103]	ESEC/FSE (2021)	Casual fairness	Ⓖ				✔	✔
Fairea [35]	ESEC/FSE (2021)	Mutation testing	Ⓖ				✔	✔
BiasFinder [36]	TSE (2021)	Mutation testing	Ⓘ				✔	✔
FairKit-learn [64]	ICSE (2022)	Search-based	Ⓘ,Ⓖ				✔	✔
PAIRFAIT-ML[104]	ICSE (2022)	Search-based	Ⓖ					✔
MAAT[34]	ESEC/FSE (2022)	Ensemble learning	Ⓖ				✔	✔
FairMask[105]	TSE (2022)	Hybrid	Ⓖ				✔	✔
ExpGA [106]	ICSE (2022)	Genetic algorithm	Ⓘ				✔	✔
Astraea [107]	TSE (2022)	Grammar-based gen.	Ⓘ,Ⓖ	✔			✔	
SBFT [108]	EMSE (2022)	Genetic algorithm	Ⓘ					✔
NeuronFair [109]	ICSE (2022)	Adversarial DL	Ⓘ				✔	✔
LTDD [110]	ICSE (2022)	Linear regression	Ⓖ				✔	✔
iRec2.0 [111]	TOSEM (2022)	Optimization problem	Ⓘ					✔
FairML [112]	MODELS (2022)	MDE-based	Ⓘ,Ⓖ	✔			✔	✔
AequeVox [113]	FASE (2022)	Metamorphic testing	Ⓖ				✔	✔
FairiFy [114]	ICSE (2023)	Satisfiability modulo theories	Ⓘ				✔	✔
DICE [115]	ICSE (2023)	Search-based testing	Ⓘ				✔	✔

3.1.3 Selected approaches

Aequitas [76] exploits three different search-based strategies to assess the group fairness of benchmarking ML-based classifiers, i.e., random, semi-directed, and fully directed. The results of the conducted evaluation show that Aequitas can reduce the unfairness of the examined ML models.

Similarly, Themis [95] provides a GUI to specify the schema of the dataset on which a user wants to assess fairness and generates a test case for it. Furthermore, it generates a report showing the relative group fairness for each variation in the value of the variables.

Sharma *et al.* [96] investigate fairness in the learning phase of an ML algorithm. The proposed tool, called TILE, relies on a metamorphic testing approach to analyze

the so-called *balanced data usage*, i.e., the learner should treat all data in the training set equally. TILE assesses fairness regarding this metric by being tested on several `scikit-learn` ML models.

A scalable gradient-based algorithm called Adversarial Discrimination Finder (ADF) has been proposed to assess individual bias by injecting individual discriminatory instances into a given dataset [97]. The global generation phase generates discriminatory entities by combining generative models and clustering techniques. Such data are refined by the local generation phase using underpinning gradients. As stated in the evaluation, the ADF algorithm overcomes two state-of-the-art tools regarding effectiveness and efficiency.

Fairway is a tool proposed by Chakraborty *et al.* that covers both bias detection and mitigation [98]. This tool works under the *Equal Opportunity* (EO) [83] and *Average Odds* (AO) [116] group definitions of fairness by identifying *ambiguous* data points. Next, it removes the bias learned by the ML algorithm through an optimization approach. Fairway succeeds in improving fairness under the EO and AO definitions.

Fairness in image classification has been investigated in [99]. The authors propose DeepInspect, a deep learning approach to mitigate two types of discrimination, i.e., confusion and bias. The underpinning network uses the neuron activation probability (NAP) matrix to predict the abovementioned discriminations. The results show that DeepInspect performs better than existing approaches in terms of accuracy, thus detecting misclassification correctly.

AITEST [100] is a tool that combines constraint-based linear optimization with the local interpretable model-agnostic explanation (LIME) techniques to perform individual fairness assessment. The proposed hybrid search strategy outperforms two notable fairness toolkits, i.e., Themis and Aequitas.

The same authors of [97] extend their former work by proposing an *Efficient Individual Discriminatory Instances Generator* (EIDIG) to generate individual fairness test cases for DNN models systematically [101]. The evaluation demonstrates that considering the gradient of the model output instead of the gradient of the loss improves the ADF's overall accuracy and F1 scores.

Fair-SMOTE [102] has been conceived to remove bias by exploiting a relabeling strategy. After the bias detection phase, it rebalances sensitive groups using the K-nearest neighbour algorithm. The results show that Fair-SMOTE solved biases before training the models compared to existing approaches.

Biswas and Rajan [103] apply fairness assessment to the preprocessing steps of ML pipelines. Built on top of the fairness causality definition, the approach automatically computes the fairness metrics at each preprocessing step of a given ML pipeline.

Fairea [35] mitigates group biases based on bias-mitigation models generated using a mutation engine. The tool identifies and tests five bias mitigation strategies to measure the trade-off between accuracy and fairness. Fairea has been evaluated using two different metrics, i.e., statistical parity difference and average odds difference.

Similarly, BiasFinder [36] adopts mutation testing to assess fairness in sentiment analysis (SA) systems. The mutant engine produces actual instances by relying on bias-targeting templates extracted from the textual content. The tool was evaluated quantitatively and qualitatively by considering two large SA datasets and human annotators.

Being built on top of sklearn and AIF360 frameworks, the Fairkit-learn toolkit [64] provides a comprehensive platform to train, test, and compare ML models by

considering fairness aspects. The tool retrieves fairer models than the two above-mentioned libraries by relying on a set of Pareto-optimal strategies.

PAIRFAIT-ML [104] exploits three different dynamic search algorithms to support hyper-parameters tuning by providing a set of bias-free configurations. The evaluation shows that the retrieved items reduce the bias by considering two metrics, i.e., equal opportunity and average odd difference.

Chen *et al.* [34] propose MAAT, an ensemble approach to optimize the bias removal by combining two different models, i.e., fairness and performance models. The former relies on the undersampling strategy to mitigate the group bias. The latter is combined with the fairness model to enhance the mitigation process regarding execution time. The empirical evaluation shows that MAAT outperforms the existing approaches, thus mitigating the detected biases in less time.

FairMASK [105] is a hybrid approach that exploits the explanation bias technique to infer possible biases before the training phase. In particular, the underpinning model is trained on non-protected attributes to use as the dependent feature in the classification task. SUbsequently, the approach performs the prediction phase by using a masking strategy to assess the overall performance. FairMASK has been compared with benchmarking tools, demonstrating that the adopted technique is more effective in mitigating group biases.

Fan *et al.* propose a model-agnostic individual fairness testing approach, namely ExpGA, based on genetic algorithms [106]. The proposed strategy can handle black-box models by feeding the underlying model with the prediction probabilities, thus optimizing the fitness value. The approach is evaluated by considering *i)* the overall performance, *ii)* the execution time, and *iii)* improvement through retraining.

Conceived explicitly for NLP systems, ASTRAEA [107] is a grammar-based instance-generation tool that identifies features causing fairness violations given an input model and mitigates them. To this end, it extracts sensitive attributes from the input grammar to cover individual and group fairness metrics.

Perera *et al.* propose the *Search-based Fairness Testing* (SBFT) tool to evaluate the individual fairness of ML regression systems [108] based on the *fairness degree* metric. The tool generates a set of unfair instances using a genetic algorithm approach. SBTF outperforms Aequitas and Themis in discovering individual unfairness.

Similar to [97], NeuronFair [109] exploits the adversarial strategy to generate individual discrimination instances (IDIs) and produce interpretable test cases for DNNs. The findings show that NeuronFair outperforms four baselines in terms of four different aspects, i.e., effectiveness, efficiency, interpretability, and generalization, by considering seven different datasets.

Li *et al.* [110] propose a logistic-regression-based training data debugging (LTDD) strategy to remove group bias from training feature values. In this respect, the approach predicts the biased part of the features and removes them from the training samples to predict the final label. The proposed strategy outperforms state-of-the-art methodologies in terms of notable fairness indicators.

iRecSys2.0 is a fairness-aware in-process crowdworker recommendation system proposed by Wang *et al.* [111]. The proposed approach is optimized to overcome popularity bias by means of a multi-objective optimization-based re-ranking component. The authors evaluated their approach in terms of the effectiveness of the predictions and fairness.

FairML [112], an MDE-based approach specifically conceived to conceptualize fairness by relying on a tailored metamodel. The dedicated DSL covers the definition of bias and the actual assessment using predefined metrics. FairML eventually

generates a YAML specification of the system that is compliant with the metamodel that the user can fine-tune.

Aequevox [113] is a testing-based approach to test fairness in automatic speech recognition (ASR) systems. Based on a tailored definition of group bias in the ASR domain, the system first uses the metamorphic testing technique to locate possible bias in the preprocessed speech. Afterward, fault localization is employed to find unrepresented groups by employing Levenshtein distance. Aequevox has been evaluated on three different commercial ASR systems, showing that the approach can automatically identify group bias in different languages

Farify [114] assesses the fairness of neural networks model using satisfiability modulo theories (SMT) technique. Given a trained neural network and targeted fairness expressed as a SAT formula, i.e., individual fairness, the approach applies input partitioning and sound pruning to identify neurons that are not activated. In addition, Fairify employs heuristic pruning to filter out neurons that can lead to bias, thus preserving fairness. The conducted experiment demonstrates that the approach is a light-weight solution for assessing fairness in neural networks.

Similarly, DICE [115] is an automatic test-generation approach to detect individual bias in Deep Neural Networks (DNNs). As the first step, the approach generates test cases to identify the amount of discrimination for a given dataset. Then, the generated test have been used to locate neurons with a significant causal contribution to the discrimination. To assess the tool's effectiveness, DICE has been run on ten different datasets, showing that the approach can locate and mitigate individual bias.

In summary, most reviewed tools primarily focus on applying pre-existing fairness definitions and metrics, ultimately conducting the final assessment within the social domain. Consequently, we recognize a compelling need to offer users a comprehensive and domain-agnostic framework that empowers them to define and evaluate their own bias and fairness criteria. Moreover, we recognize the lack of approaches that guide data scientists less experienced in the fairness domain through the development of fair learning-based systems. We address those gaps by proposing two low-code approaches to support the development of fair learning-based systems in Chapters 5 and 6.

3.2 Review of Existing Approaches for Early Bias Detection

This section overviews empirical studies on early detection of bias. Oneto *et al.* [117] employs a multi-task learning (MLT) approach to predict sensitive features based on non-sensitive ones. Openja *et al.* [118] employs a counterfactual approach to remove bias-inducing features. First, the divergence measurement metrics are computed to assess if there is some potential bias in the distribution. Then, the authors apply a data-swapping strategy to measure the impact of the features on the final predictions. ReFair [119] is a framework to automatically classify sensitive features from the textual requirements using a tailored fairness ontology. Due to the lack of publicly available requirements datasets for fairness auditing, the authors built a synthetic dataset of user stories using the GPT language model. By relying on the definition of *causal fariness* [120], Galhotra *et al.* [121] suggests a new set of features that can or cannot introduce bias given an initial set of variables extracted from the considered datasets, including a synthetic one used for assessing the complexity of the underpinning algorithm. This approach can be seen as complementary to ours since it is able to add an extended set of feature variables given an initial

one. Mecati *et al.* [122] adopts a mutation-based approach to predict possible discrimination using four different balance indexes, i.e., Gini, Shannon, Simpson, and Imbalance Ratio. Starting from an initial set of sensitive variables, the approach generates synthetic datasets according to different levels of balance in the data. Yik *et al.* [123] uses a functional specified complexity algorithm to identify potential bias in the dataset without training the ML classifiers. The approach is efficient in terms of computation even for a large number of ten different sensitive variables. Constantin *et al.* [124] propose FairAlign, a toolkit that supports the fairness auditing process using human feedback. In particular, the proposed tool has been used to annotate the variables that are perceived as biased. Afterward, FairAlign computes a set of fairness metrics and compares the prediction with the human judgment collected. In this respect, it is the only approach that can partially cover the MP feature, even though it is time-consuming since the whole procedure is performed manually. Zhang and Harmar [125] correlate the size of training data and features in group fairness assessment. The analysis involves three notable ML models and five different datasets that exhibit biases. The results show that a larger training set has a negative effect on the whole fairness metrics. On the contrary, increasing the features can lead to a reduction of the bias. Du and Chen [126] analyze fairness testing in deep learning models. In particular, they define the notion of *context* referred to other parts apart from the algorithm, e.g., hyperparameters and label bias configuration. They conducted an extensive empirical study that covered 12 datasets and 10,800 investigated cases by evaluating them in terms of fairness metrics and test adequacy. The conducted evaluation shows that better test adequacy does not necessarily lead to improvement in terms of fairness metrics.

The most relevant work to the one presented in Chapter 7 is [37]. In this empirical work, the authors perform a first investigation of the relationship of fairness metrics with data and the predictive model, i.e., data fairness metrics (DFMs) and model fairness metrics (MFMs). By testing four different ML models on five real-world tabular datasets, the analysis reveals a positive correlation between DMFs and MFMs, even though it decreases when the training sample size increases. In particular, DFMs can be used as an early warning system to identify fairness-related data drifts in automated ML pipelines. Compared to the work presented in this thesis, the authors consider at most only five datasets and focus on two metrics both residing in the *independence* category.

In Chapter 7, we introduce the concept of *bias symptoms* to support the early detection of different definitions of bias in the dataset. We evaluate the proposed approach on 24 different datasets.

3.3 Related Work on Bias in Text-to-Image Generation Models

Different works have analyzed the biases exposed by text-to-image generation models. Bianchi et al. [127] and Naik et al. [128] have shown how models like Dall-E or Stable Diffusion reinforce existing biases even with prompts simply describing occupations or traits. Sun et al. performed an extensive study of the Dall-E 2 image generation model, showing how it systematically under-represents women in specific job occupations like *computer programmer*, *sales manager*, or *criminal investigator* [129]. Luccioni et al. studied the *gender* and *ethnicity* bias exposed by Dall-E 2 and Stable Diffusion 1.4 and 2, showing how those models systematically discriminate *gender*

and *ethnicity* groups when using specific adjectives (like *ambitious*, *assertive*, *supportive*, or *sensitive*) and jobs (like *computer programmer*, *clerk* or *hostess*) in the prompt [130]. Wan et al. performed a survey study of different works analyzing the bias exposed by text-to-image generation models [131]. They show how most works focus on *gender* and *skin tone* bias while not focusing on other *ethnical* features. Moreover, they address how most works address bias exposed toward generic job categories without focusing on specific aspects. The only work analyzing the bias exposed by image generation models for SE tasks is the one proposed by Sami et al. [60]. In their study, the authors analyze the *gender* bias exposed by the Dall-E 2 model in generating images for SE tasks. To this aim, they employ and adapt the dataset proposed by Treude et al. for analyzing the bias exposed by GPT textual models for SE tasks [61]. In Section 8.1, we extend the study of Sami et al. by analyzing the *gender* and *ethnicity* bias exposed by three versions of the Stable Diffusion model - SD 2, SD XL, and SD 3 - towards SE tasks.

3.4 Related Studies on Model Repositories

Castano et al. [132] investigate to what extent the carbon footprint is reported on HF hub. After collecting 1,417 models hosted on the platform, the carbon dioxide emission has been evaluated and correlated with different factors such as model size, dataset size, and application domains.

Gong *et al.* [133] conducted a comprehensive study of the PTMs reuse stored in six different model repositories, including Hugging Face. After data cleaning and labeling steps, the authors propose code contract composed of pre- and post-conditions for re-usage in software development, e.g., input data, intended usage, and performance.

Di Sipio *et al.* [134] leverages the HF dump to classify PTMs automatically from the corresponding model card. In addition, they provide an initial overview of their usage in the most prominent SE venues and propose an initial mapping algorithm that maps generic labels to specific SE tasks. The findings of the paper show that *i)* curated model cards are adequate to train traditional ML classifiers and *ii)* the majority of the PTMs are used for supporting code and documentation generation.

Montes *et al.* [135] highlights discrepancies in the documentation of PTMs that support image classification across four different model repositories, i.e., TensorFlow Model Garden, ONNX Model Zoo, Torchvision Models, and Keras Applications. The conducted analysis of 36 selected models shows that there are severe discrepancies across the various platforms, suggesting that developers needs to carefully check the provided documentation on model zoos.

Pepe *et al.* [136] conducted a large-scale study on 159,132 models stored on HF by focusing on the documentation, licenses, and fairness aspects. Results of the study indicate that only 18% of the analyzed models declare a bias in the provided documentation. In addition, 31% of the models have a license declared with a prevalence toward permissive ones that impose a "responsible" model reuse.

The most close work related to the one presented in Chapter 8 is Gao *et al.* [137]. In their work, the authors search for ethical concerns in HF models leveraging the API and KeyBERT model, ending up with a dedicated taxonomy. Compared with the abovementioned works, we move a step forward by searching HF model directly exploited in open-source projects available on GitHub. In addition, we investigate the usage of fairness libraries in combination with the PTM models. We exploit this analysis in Section 8.2.

Chapter 4

Improving Fairness in Binary and Multi-Class Classification

In recent years, different methods have been proposed to mitigate bias at several levels of data processing [19], [31]. However, we notice that, even though it is widely adopted and constitutes a building block for personalization and search systems [29], [30], [138], the multi-class classification problem is still not effectively addressed [139]. In addition to fairness, a system must also have high effectiveness to be usable in the real world. However, in most cases, the mitigation of bias negatively influences the model's effectiveness [77], [80]. This trade-off must be managed, for example, by adequately generating or modifying the instances of a dataset in a pre-processing context or by suitably modifying the behavior of a method in an in-processing context.

For the aforementioned reasons, in this chapter, we present, as a building block of other learning-based systems, a bias mitigation method capable of *i)* managing an arbitrary number of sensitive variables in the multi-class classification scenario *ii)* preserving the accuracy of the predictions.

Referring to the challenges and contributions described in Chapter 1, this chapter presents the contribution CN_1 proposed to address the challenge CH_1.

The rest of this chapter is structured as follows: Section 4.1 describes the research questions that we will use throughout this chaper. In Section 4.2, we describe in detail the proposed approach; Section 4.3 focuses on the experimental analysis conducted to evaluate the most effective generation strategy and to compare DEMV in both binary and multi-classification scenarios. Additionally, it includes a description of how to reproduce the experiments. In Section 4.4, we discuss the results, and we answer the four RQs. Finally, Section 4.5 reports possible points of improvement of DEMV and concludes the chapter.

4.1 Research Questions

To conduct our research and to better address the problem of bias mitigation in multi-class classification, we formulate the following four research questions (RQ) that will help to highlight the fundamental findings and novel contributions presented in this chapter.

RQ_1 What are the strengths and limitations of current existing approaches addressing bias mitigation in multi-class classification problems?

We analyze three baselines designed to mitigate bias in multi-class classification problems, namely *Exponentiated Gradient* and *Grid Search* methods from [88], and the *Blackbox* method from [89]. To the best of our knowledge, these are the only methods implemented for the multi-class classification task. To

highlight their strengths and weaknesses, we apply each method to a heterogeneous set of binary and multi-class datasets that are widely used in research (extensively discussed in section 4.3.2).

RQ$_2$ How can we design a novel approach that goes beyond the existing baselines?

To overcome some of the limitations of the analyzed baseline, we present the *Debiaser for Multiple Variables (DEMV)*. DEMV is a generalization of the *Sampling* algorithm proposed by [86]. DEMV is model- and data-agnostic, allowing its introduction in already existing systems without particular effort and without introducing structural changes.

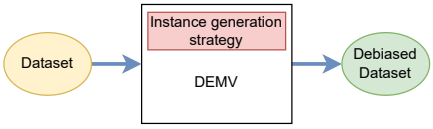


FIGURE 4.1: Application of DEMV

To the best of our knowledge, DEMV is the first proposed pre-processing method to mitigate bias caused by an unequal distribution of instances in the population (i.e., *unbalanced groups* bias [19]) in an agnostic way in both binary and multi-class classification considering multiple sensitive variables. As highlighted in Figure 4.1, DEMV takes as input a generic dataset and returns in output the debiased dataset without considering the classifier involved in the task. We implement DEMV with a plug-in approach where the user can select different *Instance generation strategies*. The source code is provided on GitHub.[1] In addition, DEMV is available as a package in PyPI repository[2] and as a method in the SoBigData RI [45].

- **RQ$_3$ How can DEMV keep a high level of accuracy while improving fairness?**

  Since DEMV is a pre-processing algorithm, the fairness and accuracy trade-off can be managed by better manipulating the instances of the dataset. Since our approach tackles the *unbalanced groups* bias, this means generating new instances that are coherent with the existing ones in terms of values and distribution. We plug-in in DEMV three different generating strategies, namely *Uniform*[3], *SMOTE* [140] and *ADASYN* [141]. To evaluate the influence of each strategy on DEMV's ability to enhance the fairness and effectiveness of the classifier, we extensively evaluate DEMV by employing nine datasets extensively used in literature (see Section 4.3.2). We perform this analysis both in binary and multi-class settings.

- **RQ$_4$ In which conditions does DEMV goes beyond the existing baselines?**

  To answer this question, we run a set of experiments aiming to evaluate the performance of DEMV in improving fairness while keeping a high level of accuracy. In particular, we evaluate it in binary and multi-class classification

[1] `https://github.com/giordanoDaloisio/demv2022`
[2] `https://pypi.org/project/demv/`
[3] Uniform strategy replicates the instances of the dataset with a uniform probability distribution.

problems and consider sensitive groups identified by up to three sensitive variables. To demonstrate the broad validity of DEMV, we employ in the experiments a heterogeneous set of ML classifiers, namely *Logistic Regression*, *Multi-Layer Perceptron*, *Gradient Boosting Classifier* and *Support Vector Classifier (SVC)*. As we show in Section 4.3.4, DEMV outperforms the baselines in the binary task when more than two sensitive variables are employed, while it remains competitive with one or two sensitive variables. In the case of the multi-class task, DEMV outperforms the baselines in every setting (i.e., number of sensitive variables) for all the considered datasets. Finally, DEMV improves the fairness of all the analyzed classification methods without affecting their behavior and maintaining a considerably high level of accuracy.

4.2 Debiaser for Multiple Variables (DEMV)

In this section, we describe in detail the *Debiaser for Multiple Variables (DEMV)* approach, a pre-processing bias mitigation method for multiple sensitive variables in the classification context.

The main idea behind the proposed method is that to enhance effectively the classifier's fairness during pre-processing is necessary to consider all possible combinations of the values of the sensitive variables and the label's values for the definition of the so-called *sensitive groups*. Under the definition of bias considered in this work (i.e., *unbalanced groups bias*), if a dataset is biased, we observe that the size of the sensitive group identified by the privileged value of the sensitive variable (e.g., *men*) and the positive label (e.g., *high income*) should be larger than expected. In comparison, the size of the sensitive group identified by the unprivileged value of the sensitive variable (e.g., *women*) and the positive label (e.g., *high income*) should be smaller than expected. In the same way, the size of the sensitive group identified by the unprivileged value of the sensitive variable and the negative label should be larger than expected, and the group size determined by the positive value of the sensitive variable and the negative label should be smaller than expected. For this reason, to enhance the fairness of the classifier, we have to perfectly balance the size of these groups by adding or removing items to remove disparity.

We approach the problem by recursively identifying all the possible groups given by combining all the values of the sensible variables with the belonging label (class). Next, for each group, we compute its expected (W_{exp}) and observed (W_{obs}) sizes[4] and look at the ratio among these two values. If $W_{exp} \backslash W_{obs} = 1$, it implies that the group is fully balanced. Otherwise, if the ratio is less than one, the group size is larger than expected, so we must remove an element from the considered group according to a chosen deletion strategy. Finally, if the ratio is greater than one, the group is smaller than expected, so we have to add another item accordingly to a generation strategy. For each group, we recursively repeat this balancing operation until $W_{exp} \backslash W_{obs}$ converge to one. It is worth noting that, in order to keep a high level of accuracy, the new items added to a group should be coherent in their values and distribution with the already existing ones.

Hence, DEMV can be defined as an algorithm made of two separate procedures: identification of the sensitive groups and balancing of them. In the following, we first illustrate the procedure used to identify the sensitive groups; then, we describe the balancing step.

[4] the formal definition of these values is given in Section 4.2.1

4.2.1 Sensitive Groups Identification

The identification and management of the sensitive groups are performed by the *DEMV* recursive function, whose pseudo-code is shown in Algorithm 2.[5]

Algorithm 2: Pseudo-code of DEMV

Input: (Dataset D, Sensitive variables $S_1, S_2, \ldots, S_n$, Label L, $i = 0$, $G = []$, condition=*true*)
Output: Sampled dataset D_S

$n = length(\{S_1, S_2, \ldots, S_n\})$
`/* base condition: check if all the sensitive variables have been explored for a given condition */`
if $i == n$ **then**
  foreach $l \in L$ **do**
    $g = \{X \in D |$ condition $\wedge\, L == l\}$
    $W_{exp} = \frac{|\{X \in D | \text{condition}\}|}{|D|} * \frac{|\{X \in D | L == l\}|}{|D|}$
    $W_{obs} = \frac{|g|}{|D|}$
    $g_b = \text{BALANCE}(g, W_{exp}, W_{obs})$
    add g_b to G
  return G
else
  `/* recursion point: select a new sensitive variable and call DEMV for each possible value of the variable */`
  $i = i + 1$
  foreach $s \in S_i$ **do**
    $G' = \text{DEMV}(D, S_1, \ldots, S_n, i, G, \text{condition} = condition \wedge S_i == s)$
    add G' to G
  `/* end condition: check if the number of explored sensitive groups is equal to the number of all possible combinations among values of the sensitive variables and values of the label */`
  if $length(G) == |L| * (\prod_{i=1}^{n} |S_i|)$ **then**
    $D_S =$ merge all $g \in G$
    return D_S
  else
    `/* if the end condition is not satisfied, simply return the set of explored sensitive groups */`
    return G

The main scope of this function is to identify and manage all the possible sensitive groups of a dataset. To this aim, this function takes as input the dataset D, the categorical sensitive variables $S_1, \ldots, S_n$, the label L and other parameters useful for the recursion: a counter i initially set to 0 (used to count the number of explored

[5] We recall that a recursive function is generally made of three main sections: a *base condition*, which defines the main return statement of the function, a *recursion point* in which the function calls itself, and an *end condition* representing the end of the process.

sensitive variables), an array G initially empty (used to collect the balanced, sensitive groups), and a boolean *condition* initially set to *true* (used to define the condition needed to identify the different sensitive groups). Lines from 2 to 9 define the base condition of the function. This condition checks if all the sensitive variables needed to identify a sensitive group have been explored (i.e., the counter i equals the number of sensitive variables). If so, the algorithm iterates the possible values of the label and creates, for each of them, a corresponding sensitive group g is defined as $\{X \in D | S_1 == s_1 \wedge S_2 == s_2 \wedge \cdots \wedge S_n == s_n \wedge L == l\}$, where $s_1, \ldots, s_n$ are possible values of the sensitive variables and l is a value of the label.

Then, for each group, the algorithm computes expected and observed sizes. These two values are defined respectively as:

$$W_{exp} = \frac{|\{X \in D | S = s\}|}{|D|} * \frac{|\{X \in D | L = l\}|}{|D|} \tag{4.1}$$

$$W_{obs} = \frac{|\{X \in D | S = s \wedge L = l\}|}{|D|} \tag{4.2}$$

where $S = s$ is a generic condition on the value of the sensitive variables[6] (i.e., *condition* variable in algorithm 2) and $L = l$ is a condition on the label's value. It is worth noting that $|\{X \in D | S = s \wedge L = l\}|$ is equal to the size of the sensitive group g identified by the conditions $S = s$ and $L = l$.

Next, the algorithm balances the group by invoking the BALANCE function (listing Algorithm 3). This function implements the balancing strategies described in section 4.2.2. Finally, the approach adds the balanced group g_b to the array G (used to collect all the balanced groups) and returns it.

Lines from 11 to 14 identify the recursion point of the function. The purpose of the recursion is to build the condition needed to determine the sensitive groups dynamically. In particular, if the algorithm has not explored all the sensitive variables (i.e., the value of i is not equal to the number of sensitive variables), the algorithm starts exploring a new one (variable S_i in the code). The exploration is done by iterating all the possible values of the current sensitive variable S_i. Each value of the sensitive variable corresponds to a new sensitive group that must be identified and balanced. Each identified sensitive group is collected inside a temporary set G'. The returning set of sensitive groups G' partially identified by the given sub-condition is then merged with the given set of sensitive groups G.

Finally, lines from 15 to 19 define the end condition of the function. In particular, the total number of sensitive groups obtainable from a dataset with n sensitive variables and a label L is equal to the product of all the possible values of the sensitive variable and the values of the label, that is:

$$|L| * \left(\prod_{i=1}^{n} |S_i| \right)$$

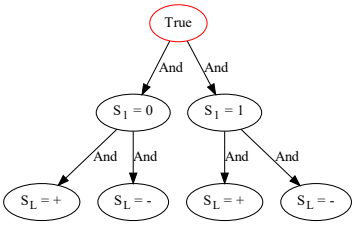
If the length of G is equal to this value, then the function has considered and balanced all the groups and returns the final sampled dataset D_S. Otherwise, the function, being in the middle of the recursion, returns G and will be again merged with the result of the previous recursive calls. The procedure shown in Algorithm 2 can also be applied to binary classification problems; in that case, the number of sensitive

[6] The variables can be binary, discrete or categorical ones

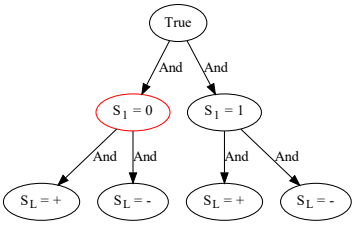
groups will be equal to

$$2 * \left(\prod_{i=1}^{n} |S_i| \right)$$

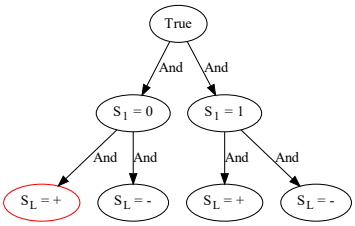
We like to note that, even if the number of sensitive groups grows exponentially with respect to the number of sensitive variables, this number is still manageable in the real-world case scenario where a small number of sensitive variables are typically considered (e.g., solely 32 groups are present in a multi-class classification task where four classes and three binary sensitive variables are considered).



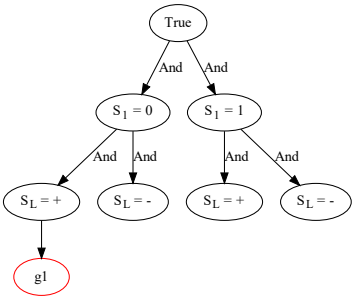
(A) Step 0: $i = 0$ and $condition = true$



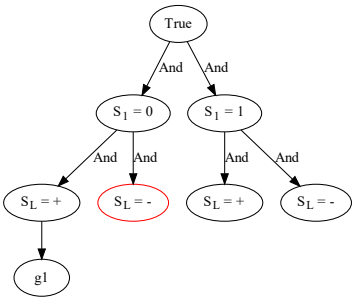
(B) Step 1: $i = 1$ and $condition = (true \wedge S_1 == 0)$



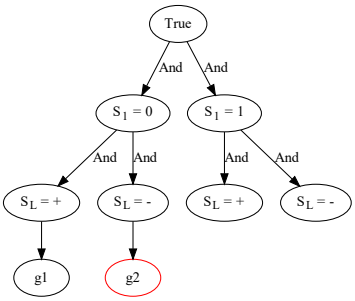
(C) Step 2: $i = 1$ and $condition = (true \wedge S_1 == 0 \wedge S_L == +)$



(D) Step 3: first sensitive group identified



(E) Step 4: $i = 1$ and $condition = (true \wedge S_1 == 0 \wedge S_L == -)$



(F) Step 5: second sensitive group identified

FIGURE 4.2: Example execution of the first steps of DEMV algorithm

To better clarify the behavior of DEMV, figure 4.2 shows an example execution of the first steps of the algorithm on a dataset with one binary sensitive variable and a binary label. For the sake of simplicity in this example we are using binary variables, but as highlighted in section 4.3, DEMV can also be applied to multi-class labels

and categorical sensitive variables. Figure 4.2a represents step 0 of the algorithm, in which the counter is set to 0, and the condition is set to *true*. Next, the algorithm starts a depth-first exploration of the depicted tree. In figure 4.2b, the algorithm adds the condition $S_1 == 0$ to the initial *true* condition, and in figure 4.2c the condition $S_L == +$ is also added. Figure 4.2d depicts the identification of the first sensitive group defined as $\{X \in D | S_1 == 0 \wedge S_L == +\}$. Finally, figures 4.2e and 4.2f show the identification of the second sensitive group, this time defined as $\{X \in D | S_1 == 0 \wedge S_L == -\}$. The algorithm then proceeds to balance the other sensitive groups. When all the groups have been balanced, they are merged to return a fully balanced dataset.

4.2.2 Balancing Strategies

The group-balancing operation is implemented by the `BALANCE` function, whose pseudo-code is depicted in Algorithm 3. This function takes as input the group g and the expected (W_{exp}) and observed size (W_{obs}). The core of this algorithm is a loop that checks if the value of $W_{exp} \backslash W_{obs}$ is different from 1. If the ratio is < 1, then it means that the size of the group is higher than expected. In this case, the algorithm selects an index in the range of $(0, size(g) - 1)$ according to the deletion strategy `REMOVE` to remove the corresponding item from the group. Otherwise, if the ratio is > 1, then the size of the observed group is lower than expected. In this case, the algorithm generates a new sample by using the generative strategy `GENERATE` and adds it to the group. Finally, the algorithm returns the balanced group when the `while` condition becomes `true` (i.e., $W_{exp} \backslash W_{obs} = 1$).

Algorithm 3: Pseudo-code of BALANCE

Input: (Group g, Expected size W_{exp}, Observed size W_{obs})
Output: Balanced group g

```
1 while W_exp \ W_obs ! = 1 do
      /* the group is not balanced                                  */
2     if W_exp \ W_obs < 1 then
          /* the size of the group is higher than expected, so we
             must remove an item from the group                     */
3         i = REMOVE(0, ..., size(g) − 1)
4         remove item i from g
5     else if W_exp \ W_obs > 1 then
          /* the size of the group is lower than expected, so we must
             add a new item to the group                            */
6         i = GENERATE()
7         add item i to g
8     recompute W_obs
9 return g
```

To better understand the overall process we need to also discuss the two underlying removal and generative strategies. The simplest of the two strategies is the removal one, where the removal candidate must be selected among the samples already present in the group. The removal strategy implemented in `REMOVE` function is typically based on a sampling function that follows a given distribution (e.g. uniform).

Conversely, the generative strategy implemented in the `GENERATE` function might be the most tricky since it is responsible for providing new samples used by the subsequent learning task. For instance, a simple approach might be to duplicate one of the samples already present in the group according to a sampling function that follows a certain distribution (e.g. uniform). It might be also possible to adopt other well-known generative approaches in the literature. In this work, we adopt a uniform sampling for the removal step while we will test and discuss three generative approaches (i.e. Uniform Sampling, SMOTE and ADASYN) in the experimental section 4.3.

4.3 Evaluation

This section describes the experiments we have conducted to evaluate DEMV: Section 4.3.1 reports the used experimental setting comprising the selected metrics and baselines; Section 4.3.2 describes the employed datasets and their characteristics; Section 4.3.3 reports on the analysis we have performed to select the best instance generation strategy to be plugged-in DEMV; Section 4.3.4 shows DEMV's evaluation results both in multi-class and binary classification; and finally, Section 4.3.5 reports a description on how to reproduce the performed experiments using the available code.

4.3.1 Experimental setting

We evaluate DEMV under heterogeneous conditions by applying a set of binary and multi-class datasets. As a base classifier, we used a Logistic Regression model [53] since it is very efficient from a computational point of view and natively supports multi-class classification. In addition, being a white-box method, it is comprehensible and promotes transparency. We also performed some specific experiments involving more sophisticated classifiers to analyze the impact of DEMV on these methods. The involved classifiers are Gradient Boosting [142], Support Vector Machine (SVM) [143], and Neural Network with *ReLU* activation function [144]. For all the experiments, we adopt the implementation from the `scikit-learn` library [145] with the default hyper-parameters.

For all the experiments, we compute the following metrics on the testing set:

- *Absolute Statistical Parity (SP)*, defined as the absolute value of the original Statistical Parity from [81]. We normalized this metric to reduce his variability and better evaluate each method's performance (i.e., avoid situations in which we measure values like 0.2 and -0.2 in two different runs, resulting in a mean of zero with a high standard deviation). The optimal value is **zero**.

- *Disparate Impact (DI)* [82], where the optimal value is **one**. To avoid the occurrence of reverse bias (i.e., metric value firmly higher than one), we adopt the formulation proposed by [146]:

$$\text{DI} = min\left(\frac{p(\hat{y}=1|s=1)}{p(\hat{y}=1|s=0)}, \frac{p(\hat{y}=1|s=0)}{p(\hat{y}=1|s=1)}\right) \tag{4.3}$$

  This metric computes the minimum among two formulations of DI: in one, the unprivileged group ($s = 0$) is at the numerator, and in the other is at the denominator. The metric value is between zero and one, where one means complete fairness.

- *Absolute Average Odds (AO)*, defined as the absolute value of the original Average Odds from [83]. We normalized this metric for the same reasons as SP. The optimal value is **zero**.
- *Zero-one Loss (ZO Loss)* [147], where the optimal value is **zero**.
- *Accuracy (Acc)* [148], where the optimal value is **one**.
- *Harmonic Mean (H-Mean)* [149] of the above metrics. In particular, concerning the metrics whose optimal value is zero (i.e., SP, AO, and ZO Loss), we beforehand perform a value's permutation to have the optimal value equal to one, and then we compute the H-Mean using these new values. Formally, H-Mean is computed as follows:

$$\text{H-Mean} = \frac{5}{\frac{1}{(1-|SP|)} + \frac{1}{(1-|AO|)} + \frac{1}{(1-|ZOLoss|)} + \frac{1}{DI} + \frac{1}{Acc}} \tag{4.4}$$

Table 4.1 shows the list of performed experiments. Specifically, we performed four main sets of experiments.

TABLE 4.1: Description of the performed experiments

Experiment	Reference section	Scope	Task	Involved classifier	Number of sensitive vars	Involved debiaser methods
1	4.3.3	Comparison of different implementations of DEMV embedding diverse generative strategies both in binary and multi-class classification	Binary and multi-class	Logistic Regression	2	DEMV Uniform DEMV Smote DEMV Adasyn
2	4.3.4	Analyze the behavior exposed by debiaser methods with sensitive groups identified by a different number of sensitive variables	Binary	Logistic Regression	1	No one EG Grid Blackbox DEMV
					2	No one EG Grid DEMV
					3	No one EG Grid DEMV
3	4.3.4	Analyze the behavior exposed by debiaser methods with sensitive groups identified by a different number of sensitive variables	Multi-class	Logistic Regression	1	No one EG Grid Blackbox DEMV
					2	No one EG Grid DEMV
					3	No one EG Grid DEMV
4	4.3.4	Analyze the behavior exposed by debiaser methods involving more sophisticated classifiers both in binary and multi-class classification	Binary and multi-class	Logistic Regression[a] Gradient Boosting Support Vector Machine Neural Network	2	No one EG Grid DEMV

[a] This classifier has not been directly employed in this experiment, but for clearness we report the results obtained in the previous experiments

The first is the comparison of different implementations of DEMV embedding diverse generative strategies (see Section 4.3.3). We consider the following three strategies:

- *Random sampling on Uniform Distribution (UNIFORM)*, where the algorithm duplicates an item present in the group with a uniform probability distribution;
- *Synthetic Minority Oversampling Technique (SMOTE)* from [140];

- *Adaptive Synthetic Sampling Approach (ADASYN)* from [141].

This analysis has been performed in binary and multi-class classification tasks on all the considered datasets considering two sensitive variables and using Logistic Regression as a classifier.

After identifying and settling on the best generation strategy, we compare DEMV with the selected baselines by performing three main sets of experiments (refer to Section 4.3.4). Experiments two and three in Table 4.1 are focused on analyzing the behavior exposed by debiaser methods with sensitive groups identified by one, two, and three sensitive variables. Experiment two focuses on binary classification task (see Subsection 4.3.4), while experiment three focuses on multi-class classification task (see Subsection 4.3.4). In both these experiments, we employed a Logistic Regression model as a classifier. To have a more concrete representation of the behavior of DEMV and the other baselines, at the end of Section 4.3.4 we also report a comparison of normalized confusion matrices for the privileged and unprivileged groups of a particular dataset.

Finally, experiment four in Table 4.1, is devoted to analyzing the behavior exposed by debiaser methods involving more sophisticated classifiers: Gradient Boosting [142], Support Vector Machine (SVM) [143], and Neural Network with *ReLU* activation function [144]. Since these models are more complex from a computational point of view, this last experiment has been performed in binary and multi-class classification tasks on a reduced but heterogeneous data set considering the two established sensitive variables (see Subsection 4.3.4).

In all the experiments (with the exception of experiment one) we compare with the following baselines:

- a biased classifier, where no debiasing method is applied, identified in the following by ***No one***;
- the *Exponentiated Gradient (**EG**)* and *Grid Search (**Grid**)* in-processing methods from [88];[7]
- the ***Blackbox*** post-processing method from [89].[8] This method has been employed only in analyses with one sensitive variable since, by the time of this paper, it does not support multiple sensitive variables.

Concerning Exponentiated Gradient and Grid Search, in agreement with the documentation available online [150], we used the Absolute Statistical Parity and the Zero-one Loss as constraints for binary and multi-class problems, respectively. Instead, Blackbox does not require a specific configuration of the hyperparameters.

For all the experiments shown in Table 4.1, we follow a *10-fold* cross-validation [151], repeated 30 times for those methods that expose a stochastic behavior (i.e., DEMV) as depicted in Figure 4.3. In particular, to better reproduce a production scenario, we apply DEMV only on the training set and train the Logistic Regression classifier using the balanced dataset. Then, we predict the labels using the original biased testing set and compute the metrics described above. In addition, since the balancing of the groups has a stochastic behavior, for each train-test fold, we repeat the aforementioned process 30 times so that we can investigate how the removal

[7]The adopted implementation of Exponentiated Gradient and Grid Search methods are available on the Fairlearn library [28]

[8]The considered Blackbox implementation is available at the following link: `https://github.com/scotthlee/fairness`

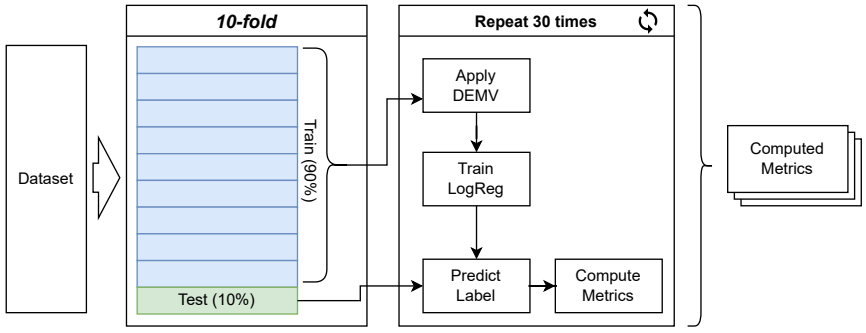


FIGURE 4.3: Evaluation procedure of DEMV for each train-test fold

or duplication of different items can influence the *accuracy* and the *fairness* of the classifier.

In all the performed experiments, we report the mean and standard deviation of all the metrics calculated over all the involved datasets. In the representation of such metrics, we use bar plots where larger bars depict the mean of the metrics, and thin bars show their standard deviation. In representing plots, we distinguish between metrics whose optimal value is 0 (shown on the left side of the figures) and metrics whose optimal value is 1 (reported on the right side of the figures).

4.3.2 Employed datasets

The experiments are conducted by employing nine well-known datasets (3 for the binary classification and 6 for the multi-class task) from the Bias and Fairness literature. For each dataset, we consider sensitive groups identified by three sensitive variables: two variables are the ones established as sensitive variables by the literature, while the third one is selected for each dataset, among the variables that could create discrimination, like age, education, and so on. Note that the PARK dataset has been excluded by the analysis with three sensitive variables because it does not have a third variable suitable for discrimination.

Table 4.2 depicts the descriptive statistics for the employed datasets. Concerning the sensitive variables, we highlight in bold the two ones established as sensitive by the literature. In the following, it is provided a brief description of the 9 considered datasets.

1. **Adult Income (ADULT)** [152]: This binary dataset comprises 30,940 items by 102 features (one-hot encoded). The goal is to predict if a person has an income higher than 50k a year. This information is represented by the `income` variable. The protected attributes are `sex`, and `race` and the unprivileged group is *black women* (items with `sex` and `race` equal to zero). In the analysis with three sensitive variables, we also introduced the `bachelor` variable, indicating if a person has a bachelor's degree or not. In this case, the sensitive group is *black women with no bachelor's degree*. The positive label is *high income*.

2. **ProPublica Recidivism (COMPAS)** [153]: This binary dataset is made of 6,167 samples by 399 attributes. The sensitive variables are `sex` and `race`. The goal is to predict if a person will recidivate in the next two years. The favorable label, in this case, is *no*, and the unprivileged group is *Non-Caucasian men* (items with `sex` and `race` equal to zero). In the test with three sensitive variables, we also introduced the `age` attribute. In this case, the sensitive group is `Non-Caucasian men with less than 50 years`.

3. **German Credit (GERMAN)** [154]: This binary dataset classifies people described by a set of attributes as good or bad credit risks (`credit` variable). The dataset consists of 1,000 instances by 59 features (one-hot encoded). The sensitive variables are `sex`, and `age` and the unprivileged group is *women with less than 25 years*. The positive label is *low credit risk*. In the experiment with three sensitive variables, we also introduced the `investment_as_income` variable, meaning if a person has more than the 30% of his income invested. In this case, the sensitive group is *women with less than 25 years and with less than 30% of their income invested*.

4. **Contraceptive Method Choice (CMC)** [155]: This multi-class dataset comprises 1,473 instances and ten columns about women's contraceptive method choice (*not-use*, *short-use*, and *long-use*). The sensitive variables are `religion` and `work`. The unprivileged group is *Islamic women who do not work* (both values equal one), and the positive label is *long-term use*. In the analysis with three sensitive variables, we introduced the education (`edu`) variable. The sensitive group, in this case, is *Islamic women who do not work and with no education*.

5. **Communities and Crime (CRIME)** [156]: This multi-class dataset is made of 1,994 instances by 100 attributes and contains information about the per-capita violent crimes in a community (variable `ViolentCrimesPerPop`). Since the label is continuous, we transformed it by grouping the values in 6 classes using equidistant quantiles. Following [157] the sensitive attribute is the percentage of the black population, but we also considered the ratio of the Hispanic population to have two sensitive variables. The unprivileged group is communities with a *high percentage of both black and Hispanic people* (both variables equal to 1), and the positive label is `100` (class of *low rate of crimes*). In the experiment with three sensitive variables, we also introduced the `MedRent` variable, showing the average price of rents in a community. In this case, the unprivileged group is *communities with a high percentage of black and Hispanic people and a low cost of the rent*.

6. **Drug Usage (DRUG)** [158]: This multi-class dataset has 1,885 instances and 15 attributes about the frequency of drugs consumption (variable `y`). The classes are *never used*, *not used last year*, and *used last year*. The sensitive variables are `race` and `gender` and the unprivileged group are *white women* (`race` equal to one and `gender` equal to zero). The positive label is *never used*. In the test with three sensitive variables, we also used the `age` variable. In this case, the sensitive group is *white women less than 50 years*.

7. **Law School Admission (LAW)** [74]: This multi-class dataset comprises 20,694 samples by 14 attributes and contains information about the bar passage data of Law School students. We grouped the continuous label (`GPA`) in 3 groups using equidistant quantiles. The sensitive variables are `race` and `gender` and the unprivileged group are *black women* (both variables equal to one), and the positive label is 2 (class of *high scores*). In the analyses with three sensitive variables, we also introduced the `age` variable. In the experiments with three sensitive variables, we also used the `age` variable. In this case, the unprivileged group is *black women with less than 61 years*.

8. **Parkinson's Telemonitoring (PARK)** [159]: This multi-class dataset comprises 5875 items and 19 features about Unified Parkinson's Disease Rating Scale (UPDRS) score classification (variable `score_cut`). The classes are *Mild*, *Moderate*

TABLE 4.2: Descriptive statistics for the employed Datasets (boldface are highlighted the protected variables established in the original dataset)

	Adult	**Compas**	**German**	**CMC**	**Crime**	**Drug**	**Law**	**Park**	**Wine**
Scope	Social	Justice	Social	Social	Justice	Social	Education	Health	Food
Instances	30,940	6,167	1,000	1473	1,994	1,885	20,427	5,875	6,438
Features	102	399	59	10	100	15	14	19	13
Classes	2	2	2	3	6	3	3	3	4
Positive label	*high income*	*no*	*low-credit risk*	*long-term use*	*100 (low percentage class)*	*never used*	*2 (high scores class)*	*mild*	*high quality*
Sensitive variables	**sex race** bachelors	**sex race** age	**sex age** investment	**religion work** edu	**black hisp** Medium Rent	**race gender** age	**gender race** age	**age sex**	**type alcohol** density
Percentage of sensitive group with two sensitive vars	5.02%	54.71%	10.50%	64.83%	23.62%	45.78%	8.42%	39.45%	11.40%

and *Severe*. The sensitive variables are `sex` and `age` and the unprivileged group are *males with more than 65 years* (`age` equal to one and `sex` equal to zero). Since this dataset does not have a third variable suited for identifying sensitive groups, we used it only in the experiments with one and two sensitive variables.

9. **Wine Quality (WINE)** [160]: This multi-class dataset comprises 6,438 instances and 13 attributes about wine quality (variable `quality`). The classes are four increasing values indicating quality (the higher, the better). The sensitive attributes are the wine's color (`type` variable) and the alcohol percentage lower or higher than 10 (`alcohol` variable). The unprivileged group is *white wine with an alcohol percentage* ≤ 10, and the positive label is 6 (*high quality*). In the experiment with three sensitive variables, we also introduced the `density` variable. In this case, the unprivileged group is *white wine with an alcohol percentage* ≤ 10 *and with a density less than 1.1%*.

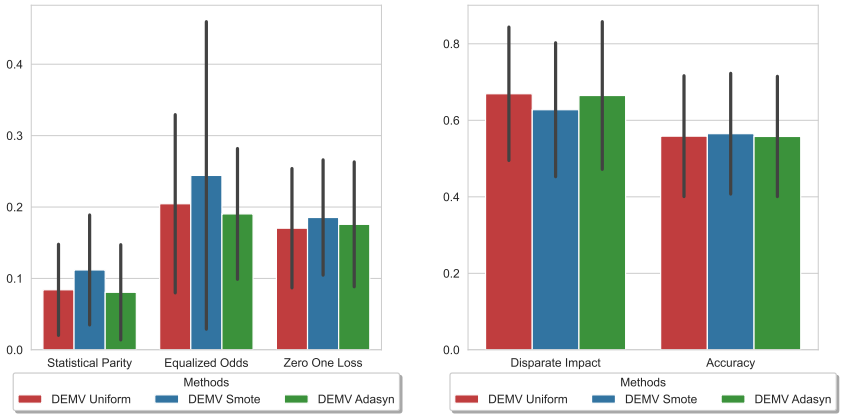
4.3.3 Selection of the best generative strategy

In this section, we show the experiments made to select the best instance generation strategy to plug-in in DEMV. As described in section 4.3.1, we consider the following generation strategies: Uniform sampling, SMOTE, and ADASYN. We perform the comparison with both binary and multi-class datasets using, for each dataset, sensitive groups identified by the two sensitive variables specified in literature. For the considered metrics (i.e., the ones introduced in section 4.3.1), we report in Appendix A.1 the tables showing the detailed values calculated. While in this section we show their mean and standard deviation calculated over all the datasets.

The aggregated metrics for multi-class datasets are shown in Figure 4.4. From this first analysis, *Uniform* sampling and *ADASYN* give similar results in fairness and effectiveness, while *SMOTE* behaves worse.

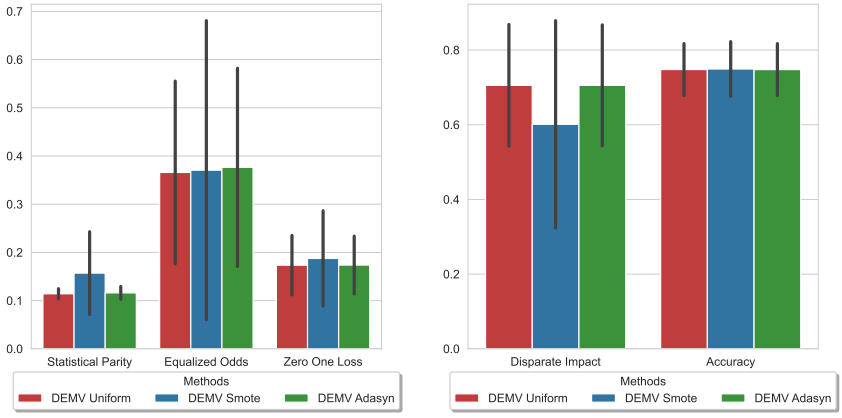
Figure 4.5 confirms that also in the case of binary classification, the *Uniform* sampling and *ADASYN* have comparable performances concerning effectiveness and fairness.

Since both the *Uniform* sampling and *ADASYN* generative strategies expose similar performance in terms of the classifier's fairness and effectiveness, we decide to analyze their computational performances in order to select the best and most efficient strategy to embed in DEMV. In particular, we focused on their execution time



(A) Metrics whose best value is zero (B) Metrics whose best value is one

FIGURE 4.4: Comparison of generation strategies of DEMV for multi-class classification with two sensitive variables



(A) Metrics whose optimal value is zero (B) Metrics whose optimal value is one

FIGURE 4.5: Comparison of generation strategies of DEMV for binary classification

(expressed in seconds) that we report in figure 4.6. This experiment has been conducted on a *MacBook Air M1 2020* with 16 GB of RAM.

The results show that DEMV implementing *ADASYN* takes much more time for completion, especially in larger datasets, while DEMV with *Uniform* always takes a reasonable execution time.

Considering all the analysis made, we adopt the *Uniform* sampling as the generation strategy to compare against the baselines because the obtained metrics are comparable to ADASYN and its execution time is lower.

4.3.4 DEMV evaluation in classification tasks

This section presents the quantitative results of the DEMV's evaluation. We compare the performance of DEMV with the selected baselines shown in Section 4.3.1.

Even if DEMV is a debiaser for the multi-class classification problem, we decided to evaluate it also in binary classification problems to identify its potentialities in this scenario (see Section 4.3.4). However, since the binary classification task is not the primary scope of our work, we decided for readability to show, for each method, the

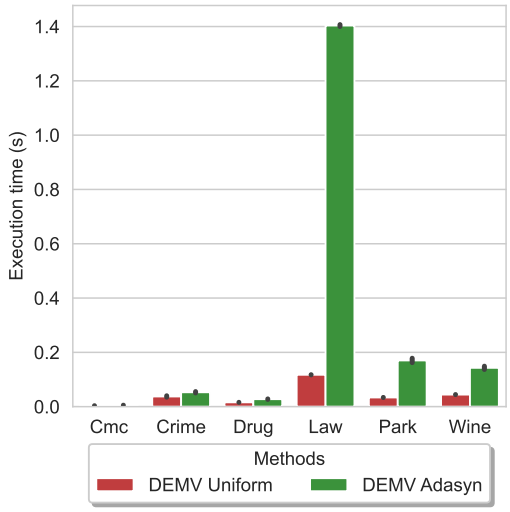


FIGURE 4.6: Execution time in seconds of DEMV Uniform and DEMV Adasyn in multi-class classifications tasks

variation of H-Means at the increasing of sensitive variables. For interested readers, detailed results are reported in the appendix A.2.

In Section 4.3.4, we present the DEMV evaluation with multi-class classification tasks. In this case, we report the mean and standard deviation of each measure described in Section 4.3 using the bar plots. Then, as an overall view, we show the variation of each method's H-Mean at increasing sensitive variables. Detailed metrics for each dataset are provided through tables in Appendix A.3.

Finally, in both binary and multi-class classification scenarios:

- For each dataset, the variation of H-Means, at the increasing number of sensitive variables is reported using line plots in which each line identifies one method. We recall that since the *Blackbox* algorithm does not support multiple sensitive variables, it has been applied only in the experiments involving sensitive groups identified by one sensitive variable, so it is represented as a point in such plots;

- Each dataset has two sensitive variables. To run the experiment with one sensitive variable, we averaged the results of two independent experiments, one for each sensitive variable;

- It is reported the statistical significance of all experiments computed using the non-parametric version of the ANOVA test in each analysis [161]. This test checks for the null hypothesis that all groups have the same mean; if the probability value (*p-value*) is less than 0.05, the test rejects the null hypothesis, which means that the groups have a different mean value. The ANOVA tables showing the test results are shown in Appendix A.4 as well.

Comparison in the binary classification task

In this section, we compare DEMV with the other baselines in a binary classification context. It is worth noting that, in the context of binary classification with one sensitive variable, DEMV coincides with the original *Sampling* method [86] it derives from.

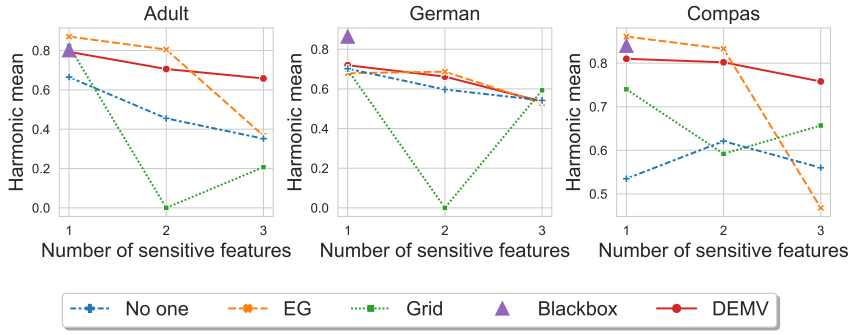


FIGURE 4.7: Comparison of overall H-Mean at different number of sensitive variables for binary classification datasets

The results of the experiments are reported in Figure 4.7. As reported in the figure, DEMV (represented with the red line) better mitigates the bias with an arbitrary number of sensitive variables, producing results that are generally competitive and even better when more than two variables are considered. A closer analysis lets us notice that EG outperforms the other methods when the number of sensitive variables is one or two. At the same time, it dramatically fails when three sensitive variables are considered. Blackbox method (reported by a single triangle in correspondence to one variable) is a good performer only when one sensitive variable is needed. In contrast, its adoption will not be applicable in cases where more sensitive variables must be considered.

The conducted ANOVA test, whose detailed results are reported in the Appendix A.4, confirms the statistical significance of all the experiments made in case of the binary classification task.

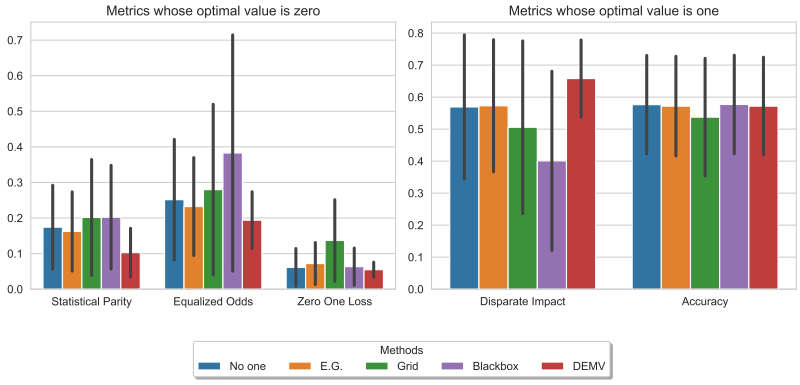
TABLE 4.3: Overall H-Mean of all methods with different sensitive variables in the binary classification context

Sensitive variables	**Methods**				
	No one	**Blackbox**	**EG**	**Grid**	**DEMV**
1	0.648 ± 0.034	$\mathbf{0.835 \pm 0.031}$	$\mathbf{0.835 \pm 0.048}$	0.761 ± 0.056	0.777 ± 0.036
2	0.558 ± 0.09	-	$\mathbf{0.775 \pm 0.077}$	0.197 ± 0.342	0.723 ± 0.072
3	0.485 ± 0.115	-	0.454 ± 0.081	0.486 ± 0.243	$\mathbf{0.651 \pm 0.111}$

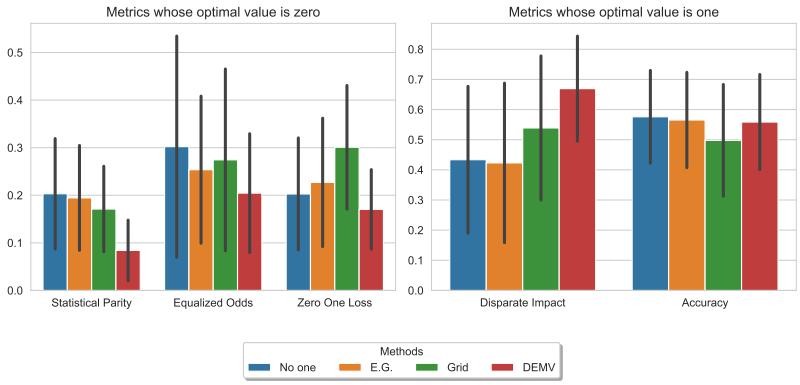
To give an overall view of the performances of each method, we provide a synthetic version of the above results in Table 4.3 where, for each method, we report the average of the H-Mean computed overall for the considered datasets. This summary confirms what we observed in the details above; that is, in binary classification with one sensitive variable, Blackbox and EG perform similarly. EG also behaves well in case of two variables. Finally, DEMV produces competitive results with one or two sensitive variables while outperforming the other baselines when three variables are needed. However, no clear winner can be picked out of the shelf, and more evaluation should be provided to determine which method to apply in different settings, including dataset characteristics. In addition, a quality that can be decisive in selecting the best method is the computational complexity, which we will consider in the future for a better evaluation of DEMV.

Comparison in the multi-class classification task

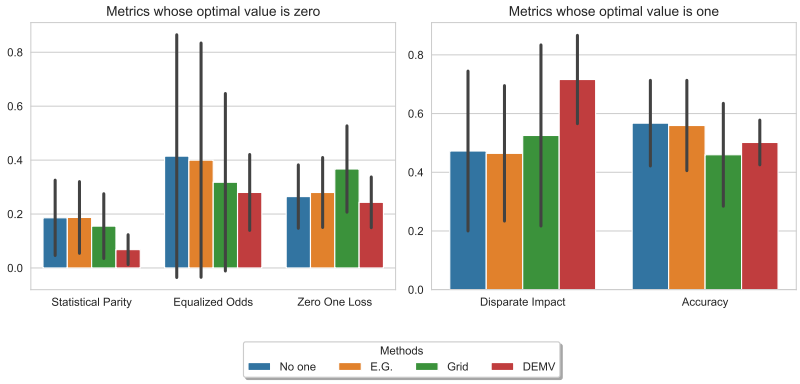
In this subsection, we report the results of the experiments conducted in the context of multi-class classification. The experiment's results are reported in Figure 4.8 and



(A) Application with one sensitive variable



(B) Application with two sensitive variables



(C) Application with three sensitive variables

FIGURE 4.8: Comparison of DEMV with the baselines in multi-class classification

Figure 4.9. The former reports the mean and the standard deviation of the metrics computed by each method on overall datasets, distinguishing among the usage of one (a), two (b), and three (c) sensitive variables. The latter instead, provides a different view, and for each dataset, it reports the values of H-Mean for each method at the increasing of sensitive variables.

In particular, Figure 4.8a focuses on the experiments involving one sensitive variable. The high standard deviation of all metrics is explained by the fact that the

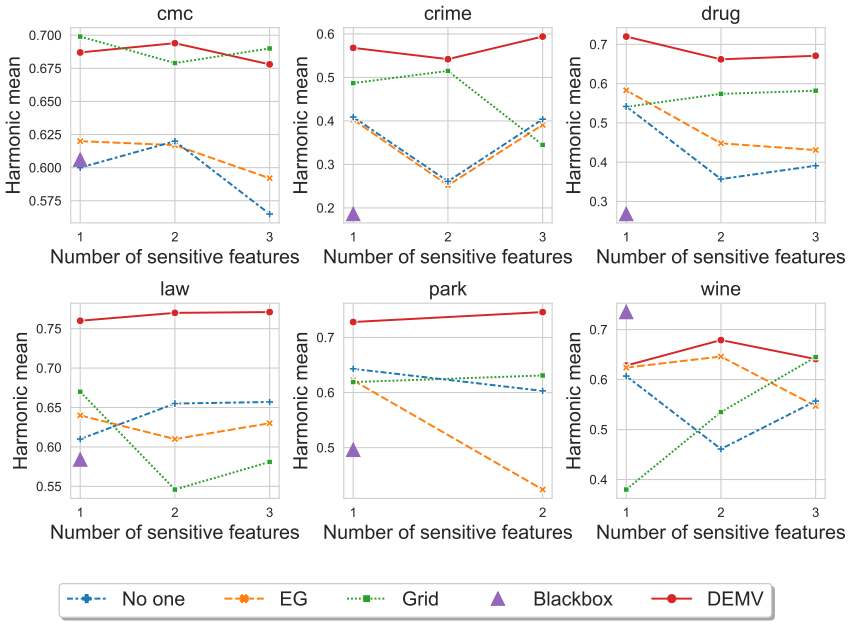


FIGURE 4.9: Comparison of overall H-Mean at different number of sensitive variables for multi-class classification datasets

metrics are here calculated putting together the results of two separate experiments. From the figure, we can see that, on average, DEMV overcomes all the baselines. In addition, we observe that DEMV is the method performing in a more stable and coherent way. This is highlighted by an overall lower standard deviation for all metrics.

The performances of DEMV in case of one variable are confirmed by figure 4.9, where it can be seen that DEMV overcomes the baselines in all datasets with the only exception of CMC (in which the best method is Grid), and Wine (in which the best method is Blackbox).

More detailed results are reported in table A.9 in the appendix A.3.

Finally, the ANOVA test (whose detailed results are reported in table A.16.a of the appendix A.4) confirms the statistical significance of the experiments with the exception of the metrics AO, which has a p-value of 0.262. The fact that the observations of AO are not statistically significant can be explained by the high standard deviation of such metric in Grid and especially in Blackbox.

Figure 4.8b reports the results of the experiments with sensitive groups identified by two sensitive variables. As before, DEMV overcomes all the other baselines in all the involved datasets, and its stability is confirmed by an overall lower standard deviation. Figure 4.9 shows that also in this context, DEMV outperforms the baselines in all datasets. Detailed results in the case of two sensitive variables are reported in table A.10 of the appendix A.3.

The ANOVA test confirms the statistical relevance of all the results (see table A.16.b in the appendix A.16).

The above considerations are also confirmed in the case of three involved sensitive variables. The results are reported in figure 4.8c and in table A.11 of the appendix A.3. We recall that the Park dataset has not been used in this experiment since it does not have a third variable suitable to be treated as sensitive.

In this case, from figure 4.9 it can be seen that Grid performs slightly better in CMC (with a delta of H-Mean of about 0.01 points) and Wine (with a delta of 0.005).

As for the other two experiments, DEMV performs more consistently with an overall standard deviation lower than the different baselines. Again, the ANOVA test confirms the statistical significance of the experiments (see table A.16.c in the appendix A.16), with the only exception of AO metrics (with a p-value of 0.27) which has a high variability especially with EG and with the biased classifier (identified by *No one* label) in the figure.

TABLE 4.4: Overall H-Mean of all methods with different sensitive variables in the multi-class classification context

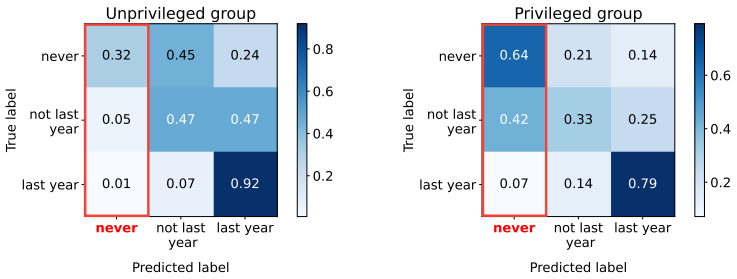
Sensitive variables	**Methods**				
	No one	**Blackbox**	**EG**	**Grid**	**DEMV**
1	0.568 ± 0.085	0.479 ± 0.211	0.582 ± 0.09	0.566 ± 0.121	$\mathbf{0.682 \pm 0.072}$
2	0.493 ± 0.16	-	0.505 ± 0.16	0.58 ± 0.063	$\mathbf{0.677 \pm 0.081}$
3	0.486 ± 0.135	-	0.49 ± 0.128	0.529 ± 0.182	$\mathbf{0.646 \pm 0.08}$

As for the experiments involving binary classification datasets, in order to have a complete, concise overview, in table 4.4, we report the overall H-Mean of all the methods in the three performed experiments over all the datasets. Note that DEMV generally overcomes the other baselines in all the explored contexts, increasing the H-Mean by up to 0.2 points with respect to the biased classifier (i.e., *No one* in the table) in the experiments with two and three sensitive variables.

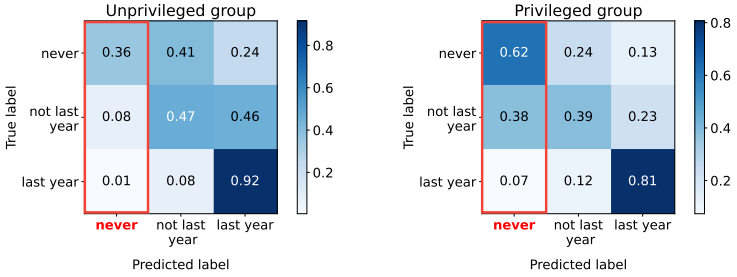
Finally, to have a more concrete representation of the behavior of all the analyzed methods, in figure 4.10 we report a comparison of the normalized confusion matrices [162] for the privileged and unprivileged (i.e., biased) groups of the Drug dataset with two sensitive variables. We decided to choose the Drug dataset for this experiment since it is among the ones having a high bias and showing better the inequality among the privileged and unprivileged groups (confirmed also by the values of the fairness metrics for the biased classifier shown in table A.10 of appendix A.3). In all the matrices, we highlight in red and in boldface the predicted positive label (i.e., *never*), which identifies the column of the matrix affected by bias (highlighted in red as well). In particular, figure 4.10a shows the confusion matrices of the biased classifier. From the picture, it can be seen how the probability of the privileged group having a predicted positive label (i.e., column corresponding to *never*) is much higher than the unprivileged group. The confusion matrices related to EG and Grid (figures 4.10b and 4.10c respectively) do not differ much from the ones of the biased classifier, meaning that these two methods are not able to improve the fairness of the classifier. Instead, in figure 4.10d, it can be seen how DEMV is able to balance these two matrices, and the probability of having the positive label predicted is almost the same for the two groups, meaning that the fairness of the classifier has increased.

Comparison using more sophisticated classifiers

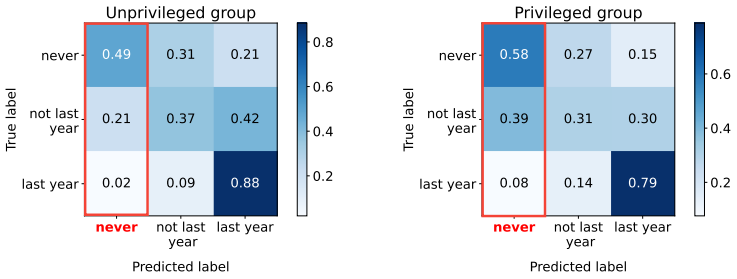
In this subsection, we report the results of the experiments conducted in binary and multi-class classification contexts using more complex classifiers. As already described in section 4.3.1, the employed classifiers are: Gradient Boosting, Support Vector Machine (SVM), and Neural Network with *ReLU* activation function. Since these models are more complex from a computational point of view, we performed these experiments on a reduced, but heterogeneous set of data using sensitive groups identified by two sensitive variables. The selected datasets are Adult (binary large



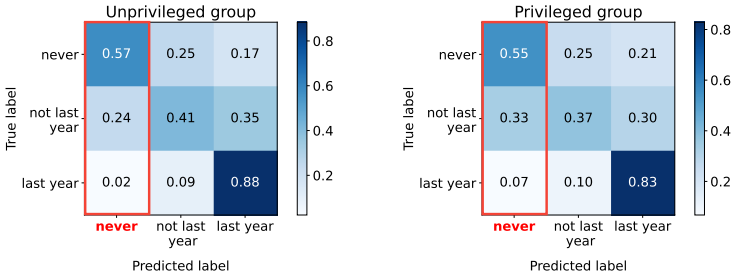
(A) Biased classifier



(B) EG



(C) Grid



(D) DEMV

FIGURE 4.10: Normalized confusion matrices of privileged and unprivileged groups for each baseline on Drug dataset

dataset), COMPAS (binary small dataset), CMC (multi-class small dataset), and Law (multi-class large dataset).

Finally, considering the debaiser approaches, it is worth noting that EG and Grid can not be applied when a Neural Network model is used as a classifier. In fact, EG and Grid apply arbitrary weights to the instances in order to remove bias, but Neural Networks, by their nature, do not allow the specification of weights to the instances. For this reason, in the experiments involving Neural Networks, we only compared the performance of the original classifier with the performance of the classifiers after the application of DEMV.

TABLE 4.5: Overall H-Mean of all methods with different classifiers in the binary classification context

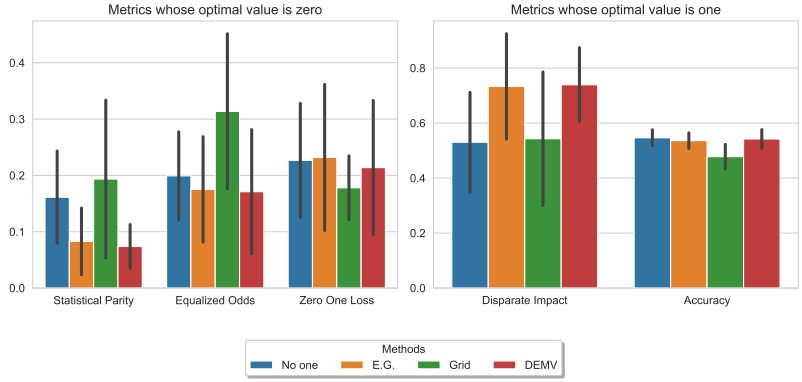
Classifier	**Methods**			
	No one	**EG**	**Grid**	**DEMV**
Logistic Regression	0.558 ± 0.09	$\mathbf{0.775 \pm 0.077}$	0.197 ± 0.342	0.723 ± 0.072
Gradient Boosting	0.588 ± 0.19	0.476 ± 0.383	0.582 ± 0.228	$\mathbf{0.724 \pm 0.057}$
SVM	0.57 ± 0.201	0.554 ± 0.238	0.59 ± 0.205	$\mathbf{0.721 \pm 0.066}$
Neural Network	0.584 ± 0.202	-	-	$\mathbf{0.69 \pm 0.127}$

Concerning binary classification, table 4.5 reports the overall H-Mean of all the baselines for each involved classifier overall the involved datasets[9], detailed results are reported in the appendix A.2 as well. Note how, differently from the experiments with a Logistic Regression classifier, DEMV overcomes the other baselines in all of the performed analyses, with a delta up to around 0.2 points in the case of EG with a Gradient Boosting classifier. The ANOVA test, whose results are reported in the appendix A.4, confirms the statistical significance of the results.

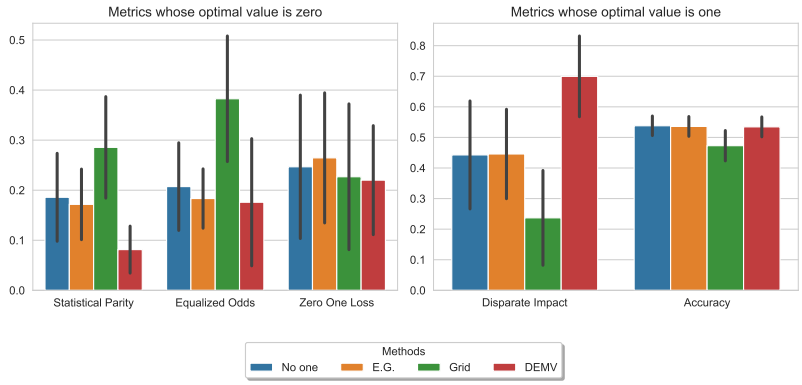
Concerning multi-class classification, figure 4.11 reports the mean and the standard deviation of all the metrics computed over all datasets. In particular, figure 4.11a shows the results of the experiments involving the Gradient Boosting classifier. In this context, DEMV outperforms the baselines under the SP and AO definitions of fairness, while it almost equals EG under the DI definition of fairness. More detailed results are reported in the table A.12 of the appendix A.3. The ANOVA test confirms the statistical significance of this experiment, with the only exception of Zero One Loss which has a p-value of 0.732 (see table A.18.a of the appendix A.4). Figure 4.11b, reports instead the results of the experiments involving Support Vector Machines. In this context, it can be seen how DEMV overcomes the baselines under all the considered definitions of fairness, keeping an accuracy level almost equal to the original biased classifier. Detailed results are reported in table A.13 of the appendix A.3 as well. Also in this case, the ANOVA test confirms the statistical significance of the results (see table A.18.b of the appendix A.4). Finally, figure 4.11c details the results of the experiments performed with Neural Networks. We recall that in this case EG and Grid can not be applied, hence we compared DEMV only with the biased classifier. Also, in this case, DEMV is able to improve the fairness of the classifiers, keeping an almost unchanged level of accuracy, and more detailed results are reported in table A.14 of the appendix A.3. The ANOVA test confirms the statistical significance of the results as well (see table A.18.c of the appendix A.4).

Table 4.6 reports the overall H-Means of all the methods for each classifier overall the selected datasets. It can be seen how DEMV generally overcomes the other baselines with all the selected classifiers with a delta up to around 0.3 points concerning SVM with Grid method.

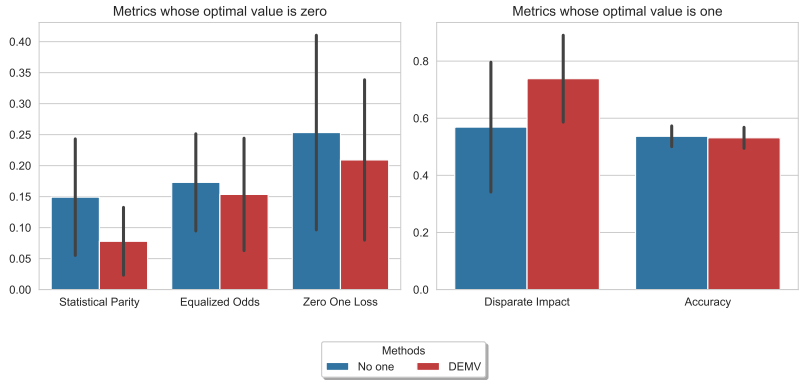
[9]To have a better overall view, we also reported the measurements computed with a Logistic Regression classifiers, which corresponds to the measures of table 4.3 with two sensitive variables.



(A) Application with Gradient Boosting



(B) Application with Support Vector Machines



(C) Application with Neural Networks

FIGURE 4.11: Comparison of DEMV with the baselines in multi-class classification using other classifiers

4.3.5 Reproducibility of the experiments

Nowadays, ensuring that the proposed methods and their corresponding results are sound and reliable is one of the challenges for research in machine learning. To ensure that the findings are valid, it is essential for the experiments to be repeatable and to yield results and conclusions comparable or identical to the originally reported ones [163]. For this reason, we choose to release the full code of the DEMV algorithm along with the replication package of all the performed experiments. This section is dedicated to describing how to use such code in order to reproduce the experiments described in the previous sections. The repository also includes a specification of all the Python dependencies required for a correct execution of the code,

TABLE 4.6: Overall H-Mean of all methods with different classifiers in the multi-class classification context

Classifier	**Methods**			
	No one	**EG**	**Grid**	**DEMV**
Logistic Regression	0.493 ± 0.16	0.505 ± 0.16	0.58 ± 0.063	$\mathbf{0.677 \pm 0.081}$
Gradient Boosting	0.653 ± 0.061	0.72 ± 0.049	0.607 ± 0.121	$\mathbf{0.729 \pm 0.018}$
SVM	0.603 ± 0.069	0.613 ± 0.054	0.392 ± 0.127	$\mathbf{0.716 \pm 0.012}$
Neural Network	0.656 ± 0.038	-	-	$\mathbf{0.728 \pm 0.016}$

which can be installed using `anaconda`[10] or `pip`[11].

The program that must be called to replicate the experiments is `generatemetrics.py`, which is responsible for generating the measures computed and aggregated in all the experiments of section 4.3. Noticing that the code to reproduce the plots presented in this paper has not been included in the replication package, but can be easily implemented using the metrics generated from the given program. The code `generatemetrics.py` can be invoked from the command line through the Python interpreter in the following way:

```
$ python generatemetrics.py <DATASET> <METHOD> <NUMBER_OF_FEATURES>
    --sensitivefeature <SENSITIVE FEATURE?> --classifier <
   CLASSIFIER?> --cm <CM?>
```

and accepts the following parameters (please refer also to the `README` of the GitHub repository for a more precise description of these parameters):

- **DATASET**: the dataset on which apply the experiments. Can be one of the datasets described in section 4.3.2.
- **METHOD**: debiaser method to use. Can be one of the debiaser methods employed in this paper or `biased` in case of no methods.
- **NUMBER OF FEATURES**: number of sensitive variables to identify the sensitive groups. Can be an integer up to 3.
- **SENSITIVE FEATURE**: optional parameter to specify the sensitive variables for the identification of the sensitive groups in case `NUMBER OF FEATURES` is equal to one or two. To ensure that the selected variables are truly sensitive, they must be among the three sensitive variables defined for each dataset in section 4.3.2.
- **CLASSIFIER**: optional parameter to specify a classification method. Can be one of the classifiers employed in the experiments of this paper. The default is the Logistic Regression classifier.
- **CM**: optional boolean value to plot the confusion matrices of the two sensitive groups. Default is `false`.

[10] `https://www.anaconda.com/`

[11] `https://pypi.org/project/pip/`

The following command can also be executed to receive help and information on the requested parameters:

```
$ python generatemetrics.py -h
```

The execution of this script will produce a `.csv` file containing all the measures described in section 4.3.1 for each train-test fold. We like to remark that, in case of DEMV, the number of measures will be equal to 30 times the number of train-test folds (see the description of the experiment setting in section 4.3.1).

To reproduce the experiments of section 4.3.3, the script can be called passing as input any of the involved dataset, a number of sensitive variables equal to 2, and `UNIFORM`, `SMOTE` or `ADASYN` as debiaser method. For instance:

```
$ python generatemetrics.py cmc uniform 2 --sensitivevariable
    religion,work
```

will produce the metrics of the CMC dataset using the DEMV Uniform debiaser strategy. Aggregating the results of the execution of this script for all of the involved datasets and all of the analysed generation strategies makes possible to reproduce the experiments and charts shown in the section 4.3.3.

Similarly, it is possible to reproduce the experiments of section 4.3.4 by running `generatemetrics.py` on all the combinations of datasets, methods, number of sensitive variables and classification methods and then aggregating the results. For instance, the following command:

```
$ python generatemetrics.py adult eg 3 --classifier gradient
```

will generate the metrics for the Adult dataset with three sensitive variables using the EG debiaser method and the Gradient Boosting classifier.

Finally, confusion matrices for the privileged and unprivileged groups can also be created using this script. For instance, the command:

```
$ python generatemetrics.py crime uniform 3 --cm
```

will generate the confusion matrices of the Crime dataset for the privileged and unprivileged groups.

It is also possible to refer to the `README` file on the GitHub repository for a more complete description of the method and the parameters.

In addition to the source code, DEMV is also freely available as a ready-to-use package in the PyPI repository[12] and it is provided as a ready-to-use method into the SoBigData RI [45].

4.4 Discussion

In this section, we discuss the results of the experiments conducted in section 4.3 by referring to the research questions highlighted at the beginning of this chapter.

4.4.1 RQ_1: Evaluation of existing approaches

From the experiments conducted in section 4.3, we have seen that almost all the baselines can improve fairness in the binary classification context and classification problems involving one sensitive variable. However, no baselines can consistently handle bias in the multi-class classification domain with multiple sensitive variables

[12] `https://pypi.org/project/demv/`

either because they do not support it (like in the case of *Blackbox*) or because they perform very poorly (like in the case of *EG* and *Grid*). More specifically, the strengths and weaknesses we have observed for each baseline in the performed experiments are as follows:

- **EG.** It can improve fairness in the context of binary classification with very relevant results. It has much more difficulty in improving fairness in multi-class classification. This weakness might be imputed to the constraint metric suggested by the authors to be used in the case of multi-class classification, i.e., ZO Loss. In addition, this method is highly influenced by the involved classification algorithm and can not be applied if the employed classifier is a Neural Network.
- **Grid.** His performances are strictly related to the dataset and the search space size. Since, in our experiments, we always used a grid size of 20 (the default value of the adopted implementation), this method performed well with some datasets and worse with others in which a larger search space was needed. This results in a very high variability of the overall obtained metrics. In particular, our experiments observed that Grid performs well with the CMC and Wine multi-class datasets. At the same time, in the binary classification task, the Grid method exposes a higher variability even in the same dataset but among a different number of sensitive variables. Finally, as for EG, this method is strongly influenced by the classification algorithm and can not used if the employed classifier is a Neural Network.
- **Blackbox.** This method performs well in mitigating bias in binary and multi-class classification. However, it does not support multiple sensitive variables. In addition, we observed high variability in the overall metrics that let the method be considered unstable.

4.4.2 RQ_2: Overcoming existing limitations

In this work, we have presented the *Debiaser for Multiple Variables (DEMV)*, which is, to the best of our knowledge, the first *pre-processing* approach able to improve fairness both in binary and multi-class classification with multiple sensitive variables. DEMV generally overcomes the other baselines in binary and multi-class classification tasks with one, two, and three sensitive variables. In addition, being a pre-processing method, DEMV can be applied to a heterogeneous set of classification methods without impacting or being influenced by their behavior. DEMV is also the method that performs more consistently in all the experiments, resulting in less variability of the overall metrics.

4.4.3 RQ_3: DEMV instance generation strategy

The generative strategy that must be adopted to rebalance groups in DEMV is Uniform sampling. The Uniform generating strategy is preferable from two points of view: *i)* it is the best performer (among the other generative strategies) in terms of fairness and accuracy; *ii)* its computational complexity is negligible. This approach has been adopted in DEMV and compared with the other baselines, obtaining excellent results.

4.4.4 RQ_4: Comparison with baseline approaches

The performed experiments showed that DEMV can improve the fairness in binary and multi-class classification contexts with any number of involved sensitive variables, keeping a high level of accuracy. In particular, our method overcomes the other baselines in multi-class classification problems with any number of sensitive variables. In contrast, as expected, other specifically designed methods may perform better in binary classification with one or two sensitive variables. Instead, we have noticed that when the sensitive variables are more than two, DEMV overcomes the baselines also in the case of binary classification. In addition, we have shown how DEMV can consistently improve the fairness of several classification methods without impacting their behavior, while other debiaser methods behave differently according to the involved classification method or can not be applied at all (like EG and Grid with a Neural Network). Finally, we observed that when the size of the sensitive group is tiny, DEMV has more difficulty improving fairness and finding the optimal group size. This issue can be explained by the fact that when the group size is small, the addition or removal of a single item impacts more on the expected and observed size ratio, so the optimal balancing is more complex or impossible to achieve.

4.5 Conclusion

We addressed the problem of bias mitigation in the multi-class classification context by proposing the *Debiaser for Multiple Variables*, a novel approach extending the work of [86] to the multi-class classification domain with multiple sensitive variables. To the best of our knowledge, DEMV is the first pre-processing method able to handle bias in both binary and multi-class classification problems with any number of sensitive variables. Thus, it can be applied during the *feature engineering* phase to mitigate the bias exposed by the dataset.

We have exhaustively evaluated our algorithm by comparing it with three established baselines using a heterogeneous set of binary and multi-class datasets, with a different number of sensitive variables, and by employing a heterogeneous set of classification methods. In addition, we have also evaluated how different instances generation strategies can influence the ability of DEMV in improving fairness. The conducted experiments show that our method is the better choice to adopt in the multi-class classification context with one, two, or three sensitive variables. Instead, we noticed how other specifically designed methods might perform better in binary classification with one and two sensitive variables. However, our method is still the better solution in binary classification problems with three sensitive variables and, being a pre-processing method, it can be successfully applied even with classifiers not supported by other baselines (e.g., Neural Networks).

Finally, we have seen that the Uniform sampling of existing instances is the best strategy to manipulate groups in terms of accuracy and fairness.

In the future, we want to overcome the current main weakness of DEMV, highlighted when the sensitive groups are very small. We will address this issue by investigating if there are situations that lead to optimal fairness before a complete balance within the groups. If so, we want to identify further which are the characteristics that lead to these situations. In addition, we want to improve our analysis by studying the impact of the size of the dataset and the number of attributes that are not the sensitive variables on DEMV and his behavior. Next, we also want to assess the computational complexity with respect to the other baselines, and we want

to study the impact that different removal strategies may have during the balancing procedure and, thus, on the capacity of DEMV to mitigate bias and improve fairness. Finally, we will study the impact that DEMV has on the fairness of the full pipeline of recommender systems that embed a multi-class classifier.

Chapter 5

Modelling Fairness Concepts and Metrics

This chapter presents the formal representation of the workflows introduced in Chapter 2 for fairness assessment and to identify the best combination of ML models and fairness-enhancing methods. We demonstrate how to model the workflow to select the best ML model and fairness-enhancing method as a Software Product Line (SPL). In addition, we propose a metamodel to represent the entire fairness assessment process. These formalisms underpin the development of two low-code approaches, MANILA [46] and MODNESS [47], which are discussed in Chapter 6.

Referring to the challenges and contributions described in Chapter 1, this chapter presents the theoretical foundations behind contribution CN_2 proposed to address the challenge CH_2.

The chapter is structured as follows: Section 5.1 provides background knowledge on Model Driven Engineering (MDE) and Feature-Oriented Software Development (FOSD). Section 5.2 details the modeling of the workflow to evaluate different ML models and fairness-enhancing method combinations and select the best one. Section 5.3 instead introduces the metamodel for fairness assessment. Finally, Section 5.4 wraps up this chapter.

5.1 Background on Software Modeling

5.1.1 Model Driven Engineering

Model Driven Engineering (MDE) is a software development methodology that uses models as primary artifacts in the software development process [164]. MDE is based on the principle of separation of concerns, where different aspects of a system are modeled independently and then integrated to form a complete system. MDE is widely used in software development to address the complexity of modern software systems and to improve software quality and maintainability [164].

MDE is based on the concept of *models*, which are abstract representations of a system that capture different aspects of that system. Models can be used to describe the structure, behavior, and properties of a system, and can be used to generate code, documentation, and other artifacts [164]. Models are typically created using modeling languages, which provide a set of concepts and notations for describing the system.

In MDE, models adhere to a hierarchical structure where each model conforms to a specific meta-model. A meta-model acts as a blueprint, defining the concepts and relationships that can be utilized across all models derived from it. Specifically, a meta-model encapsulates all the entities that can be instantiated within a model and

specifies the relationships that can exist among them, such as references, compositions, and extensions [165]. Similar to how objects in Object-Oriented Programming are instances of a class, models in MDE are instances of a meta-model. These models can represent various facets of a system, including its structure, behavior, and properties.

Different technologies can be used to implement MDE, and provide tools and frameworks for creating, editing, and transforming models. Among those, the Eclipse Modeling Framework (EMF) is a widely used technology in MDE [165]. The popularity of this technology is enforced by a wide ecosystem of tools and plugins that support developers in activities such as the creation of Domain Specific Languages (DSLs) [166] or code generation [167].

The high flexibility and expressiveness of MDE make it a suitable approach for modeling complex software processes, such as the fairness assessment workflow discussed in Chapter 2. MDE is particularly well-suited for scenarios involving multiple stakeholders operating at various levels of abstraction, enabling clear representation and management of intricate workflows.

A previous work employing MDE for fairness development is FairML, developed by Yohannis and Kolovos [112]. The authors have first defined a meta-model to represent scripts for bias mitigation and assessment. Then, they defined a Domain Specific Language (DSL) to specify such scripts. Finally, from a model defined using such a DSL, the proposed tool automatically generates a Python script implementation. However, the proposed approach does not guide the data scientist in selecting components always leading to an executable experiment. Moreover, it does not allow the specification of custom fairness definitions and metrics. With our proposed approaches, we overcome those limitations.

5.1.2 Feature Oriented Software Development

Feature-Oriented Software Development (FOSD) is a software development methodology that focuses on the reuse of software artifacts across multiple software products [48]. FOSD is based on the concept of *software product lines (SPL)*, which are families of related software products that share a common set of features.

Feature Models are a traditional formalism in FOSD to represent SPL [48]. Indeed, Feature Models allow the definition of a template for families of software products with standard features (i.e., components of the final system) and a set of variability points that differentiate the final systems [48].

Features in the model are organized in a hierarchical, tree-like structure based on parent-child relationships. These features can be categorized as either mandatory (i.e., they must be included in the final system) or optional (i.e., their inclusion in the final system is discretionary) [168]. Sibling features may form an Or-relationship (i.e., zero or more features can be included) or an Alternative-relationship (i.e., at most one feature can be included) [168]. Additionally, Cross-tree relationships may exist between features in different branches of the tree. These relationships are typically defined using logical propositions [168].

One limitation of traditional Feature Models is that they do not allow associating attributes with features. This limitation is particularly relevant in the contexts like the workflow for the evaluation of fairness-enhancing methods, where information such as the name of the column encoding the label in the dataset or the number of rounds to perform for cross-validation [151] are essential to generate the code implementing the experiment workflow. To address this limitation, the Extended Feature Models (ExtFM) formalism has been proposed by the literature [169], [170]. ExtFM

extends the Feature Model formalism by allowing the association of attributes with features, enabling a more detailed representation of software product lines.

Regarding the adoption of Feature Models to model learning-based systems, Di Sipio *et al.* presented LEV4REC, a Feature Model-based application to define pipelines for the recommender systems domain [171]. The variation points are identified by all the components needed to implement a recommender system (e.g., the ML algorithm to use or the Python libraries for the implementation). From a given specification, the application generates a Python implementation of the modeled recommender system.

5.2 An Extended Feature Model to Support the Development of Fair and Effective Learning-Based Systems

In this section, we introduce an Extended Feature Model (ExtFM) that models the general fairness benchmarking workflow presented in Chapter 2 [46]. The ExtFM has been implemented using *FeatureIDE*, an open-source graphical editor which allows the definition of ExtFMs [172].

Figure 5.1 presents a condensed version of the implemented ExtFM.[1] The ExtFM is designed to assist data scientists during the *model requirements* phase (refer to Figure 1.1) by guiding the selection of all necessary features for evaluating and identifying the optimal combination of ML models and fairness-enhancing methods.

The ExtFM comprises seven macro features, which are then detailed by children's features. The first mandatory feature is the Dataset. The Dataset has a file extension (e.g., CSV, EXCEL, JSON, and others) and a Label, which can be Binary or Multi-Class. The Label feature has two attributes specifying his name and the positive value (used to compute fairness metrics). The Dataset could also have one or more sensitive variables that identify sensitive groups subject to unfairness [19]. The sensitive variables have a set of attributes to specify their name and the privileged and unprivileged groups [19]. Finally, there is a feature to specify if the dataset only has positive attributes. This feature has been included to define a cross-tree constraint with a scaler technique that requires only positive attributes (see Table 5.1). All these features are modeled as abstract since they do not have a concrete implementation in the final experiment.

The next feature is a Data Scaler algorithm, which is not mandatory and can be included in the experiment to scale and normalize the data before training the ML model [173]. Different scaler algorithms from the *scikit-learn* library [145] are listed as concrete children of this feature.

Furthermore, the model includes a macro feature representing the ML Task to be performed. This feature is not modeled as mandatory because two fairness-enhancing methods — *Gerry Fair* and *Meta Fair* [174], [175] — integrate a built-in fair classification algorithm. As a result, if either of these methods is selected, specifying the ML Task becomes unnecessary. However, to maintain consistency, a cross-tree constraint has been included to enforce the selection of the ML Task unless one of these two methods is chosen: $\neg$ Gerry Fair $\wedge$ $\neg$ Meta Fair $\implies$ ML Task. An ML Task could be Supervised or Unsupervised. A Supervised task could be a Classification task or a Regression task and has an attribute to specify the size of the training set. These two abstract features are then detailed by a set of concrete implementations of ML methods selected from the *scikit-learn* library [145]. The Unsupervised learning

[1] The complete ExtFM is accessible online at `https://github.com/giordanoDaloisio/manila-web/blob/main/fm-full.png`

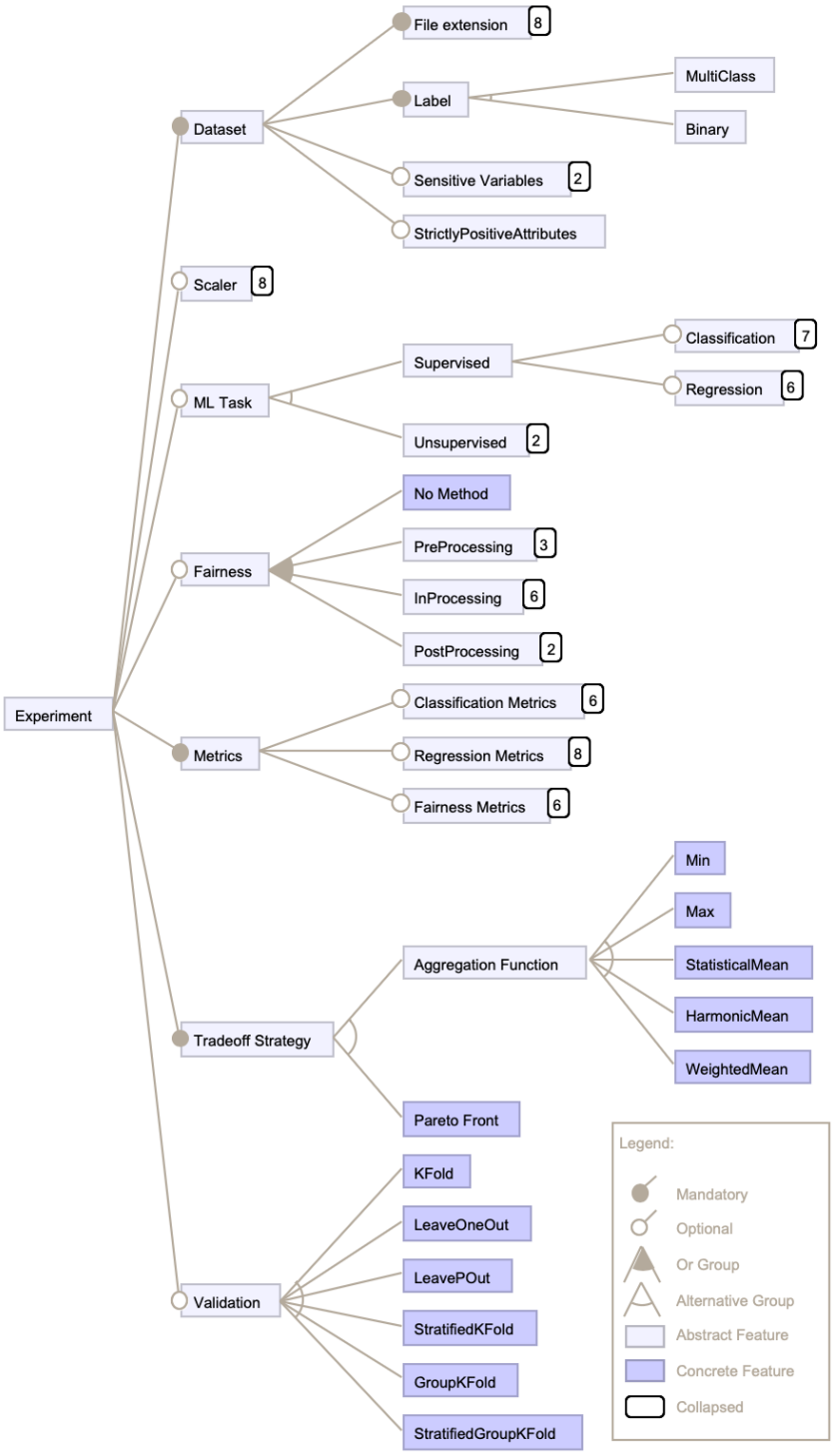


FIGURE 5.1: Short version of the implemented Extended Feature Model

task could be a Clustering or an Aggregation task. At this stage of the work, these two features have not been detailed and will be explored in future works.

Next is the macro feature representing the methods to enhance the system's fairness. This feature is mandatory since a data scientist could simply focus on the model's effectiveness without accounting for fairness. Fairness methods can be *Pre-Processing*, *In-Processing*, and *Post-Processing*. These three features are detailed by

several concrete features representing fairness-enhancing methods. In selecting such algorithms, we selected methods with a solid implementation, i.e., algorithms integrated into libraries such as *AIF360* [27] or *Fairlearn* [28]. In addition, we included the *DEMV* algorithm presented in Chapter 4 [43] for bias mitigation in multi-class tasks. All these methods have been modeled with an *Or-group* relationship since multiple methods could be tested on the same ML model.

The next macro feature represents the metrics used in the experiment. Metrics are divided among metrics to assess the effectiveness in classification or regression tasks and metrics for fairness assessment. Each metric category has a set of concrete metrics selected from the *scikit-learn* library [145] and the *AIF360* library [27]. If the data scientist is evaluating classification methods, then at least one metric for classification effectiveness must selected. The same applies to regression tasks. In addition, if the data scientist is evaluating the model's fairness, then at least one fairness metric must be selected. These constraints are formalized by cross-tree relationships among features (see Table 5.1).

The final mandatory feature pertains to the trade-off strategy used to identify the optimal ML model and any chosen fairness-enhancing methods. There are two types of trade-off strategies. The first includes aggregation functions that summarize metrics using specific calculations, such as median, minimum, or maximum values. The second type is the Pareto front function, which yields the set of non-dominated solutions [176].

Finally, there is the optional macro feature identifying the Validation function. Validation functions are different strategies to evaluate the effectiveness and fairness of an ML model [151]. Several Validation functions are available as children features, and there is an attribute to specify the number of groups in case of cross-validation [151].

Table 5.1 lists the cross-tree constraints among features. These constraints guide the data scientist through the definition of a complete and correct (i.e., not leading to execution errors) experimental workflow. They have been derived from an empirical analysis of methods and metrics provided by the *sklearn* and *aif360* libraries and by testing all possible combinations of ML models and fairness-enhancing methods.

5.3 A Metamodel for Fairness Assessment

In this section, we introduce a metamodel for fairness assessment based on the key concepts provided in Chapter 2 [47]. The proposed metamodel allows data scientists and domain experts to define the bias related to a particular domain and to specify fairness analyses starting from that definition.

The ground idea behind the proposed metamodel is that a fairness assessment process can be depicted as a two-layered workflow. First, there is a high-level definition of bias for a given domain. Next, starting from a high-level definition of bias, multiple concrete fairness analyses can be defined on a specific dataset using standard or even custom metrics. Thus, the metamodel can be divided into three related packages providing the modeling constructs for bias definitions (see Fig. 5.2), fairness analyses specification (see Fig. 5.3), and for the definition of metrics (see Fig. 5.4).[2]

[2]A full picture of the metamodel is available online at this link `https://github.com/giordanoDaloisio/MODNESS/blob/main/assets/metamodel.jpg`

TABLE 5.1: Extended Feature Model cross-tree constraints

Cross-tree constraints	**Description**
Fairness ⇒ Sensitive Variables	If the fairness QA is selected, then the sensitive variables must be specified
Fairness ⇒ Fairness Metrics	If the fairness QA is selected, then at least a fairness metric must be selected
Multiple Sensitive Var ⇒ ¬ Sampling ∧ ¬ Blackbox ∧ ¬ DIR	If multiple sensitive variables are selected, then disable the fairness enhancing methods not supporting more than one sensitive variable
MultiClass ⇒ ¬ Reweighing ∧ ¬ DIR ∧ ¬ Optimized Preprocessing ∧ ¬ LFR ∧ ¬ Adversarial Debiasing ∧ ¬ Gerry Fair ∧ ¬ Meta Fair ∧ ¬ Prejudice Remover ∧ ¬ Calibrated EO ∧ ¬ Reject Option	If a multi-class label is selected, then disable the fairness methods not supporting multi-class classification
Regression ⇒ ¬ PostProcessing ∧ ¬ Reweighing ∧ ¬ DIR ∧ ¬ DEMV ∧ ¬ Optimized Preprocessing ∧ ¬ LFR ∧ ¬ Adversarial Debiasing ∧ ¬ Gerry Fair ∧ ¬ Meta Fair ∧ ¬ Prejudice Remover	If the regression task is selected, then disable all the fairness methods not supporting this task
Exponentiated Gradient ∨ Grid Search ⇒ ¬ MLP Classifier ∧¬ MLP Regressor	Exponentiated Gradient and Grid Search fairness methods do no work with MLP Classifier and MLP Regressor ML methods
¬ GerryFair ∧¬ MetaFair ⇒ ML Task	If not GerryFair and MetaFair fairness methods are selected, then an ML Task must be selected
Classification ⟺ Classification Metrics ∧¬ Regression Metrics	If the classification ML task is selected, then at least one classification and no regression metrics must be selected, and vice versa
Classification Metrics ⟺ ¬ Regression Metrics	If a classification metric is selected, then a regression metric must not be selected and vice versa
Regression ⟺ Regression Metrics ∧¬ Classification Metrics	If the regression ML task is selected, then at least one regression and no classification metrics must be selected, and vice versa
Box-Cox Method ⇒ Strictly Positive Attributes	If the Box-Cox scaler method is selected, then the dataset must have strictly positive attributes

5.3.1 Bias Definition

The root of the metamodel in Figure 5.2 is the abstract class `Bias` representing the concept of discrimination at a higher level of abstraction. Bias has a domain and one or more sources (i.e., what generated bias). The possible sources of bias have been selected from [19] e.g., *human discrimination*, *wrong sampling of data*, among others and have been listed in the dedicated `BiasSource` enumeration. Then, bias is composed of a `PositiveOutcome` and one or more `SensitiveVariable` instances, which have one or more `SensitiveVarValue` each. Finally, both the privileged and unprivileged groups must be specified in defining bias. The `SensitiveGroup` metaclass models these groups. In particular, each `SensitiveGroup` is identified by one or more `SensitiveVarValue`. The `Bias` metaclass is then specialized by two submetaclasses representing `GroupBias` and `IndividualBias`. Both group and individual biases are composed of one or more `Analysis` with a particular `Scope`. In particular, `GroupBias` has one or more `GroupAnalysis`, while `IndividualBias` has one or

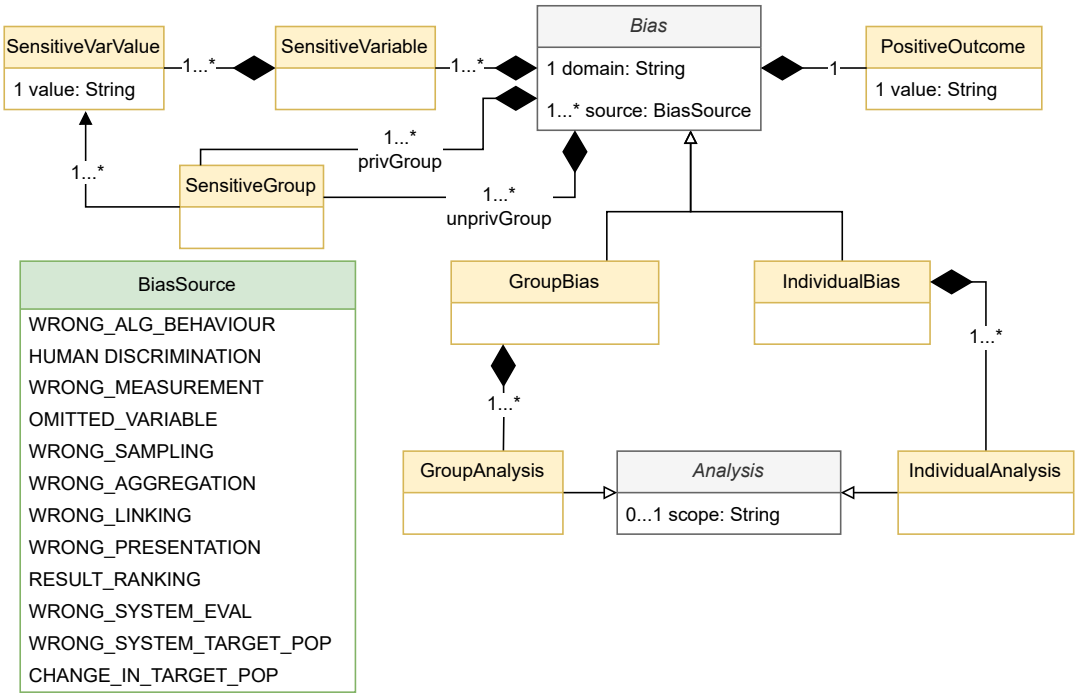


FIGURE 5.2: Bias Definition.

more `IndividualAnalysis`.

It is worth noticing how, being high-level and not related to a specific dataset or analysis, this portion of the model can be defined by the domain and legal experts only, without assistance from data scientists.

5.3.2 Fairness Analysis

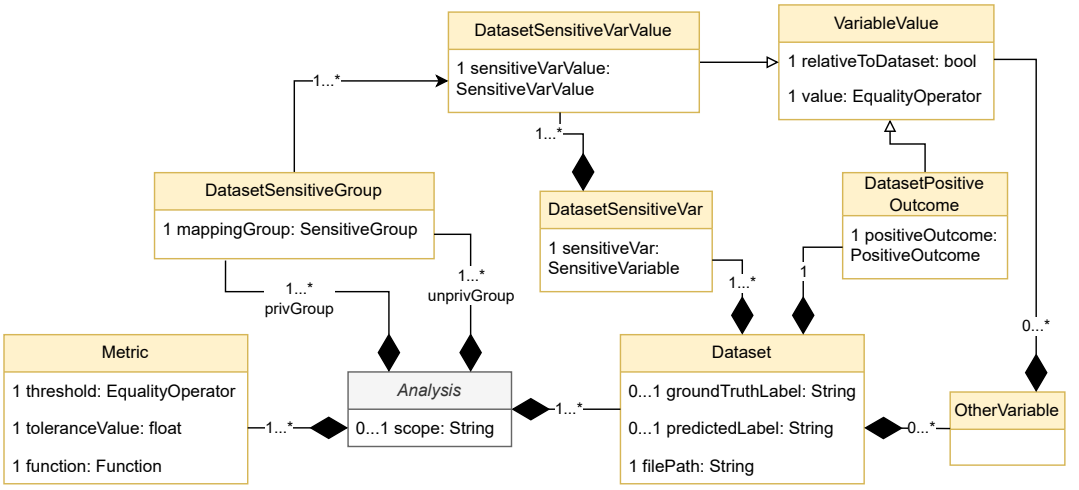


FIGURE 5.3: Fairness Analysis.

Figure 5.3 depicts the metamodel portion dedicated to the fairness analysis specification, represented by the `Analysis` abstract metaclass. An analysis may have a scope, i.e., a textual description of the analysis, and is composed by one or more `Datasets`. A `Dataset` has an attribute to specify the name of the column containing

the `groundTruthLabel` (if any), an attribute to specify the name of the column containing the `predictedLabel` (if available), and an attribute to specify its `filePath`. Then, a mapping of each general concept defined in the bias definition must be identified in the dataset. In particular, a `Dataset` is composed of one `DatasetPositiveOutcome` metaclass mapping the value of the positive outcome in the dataset, one or more `DatasetSensitiveVar` metaclasses (which are in turn composed of one or more `DatasetSensitiveVarValue` metaclasses) mapping the sensitive variables, and, if needed, one or more `OtherVariable` metaclasses representing other values encoded in the dataset. Then the sensitive groups must also be mapped in the dataset through the `DatasetSensitiveGroup` metaclass. All the metaclasses representing values extend a `VariableValue` metaclass. It is worth noting that all the values can be absolute or relative to the dataset. Finally, an analysis comprises one or more `Metric`.

5.3.3 Metric Definition

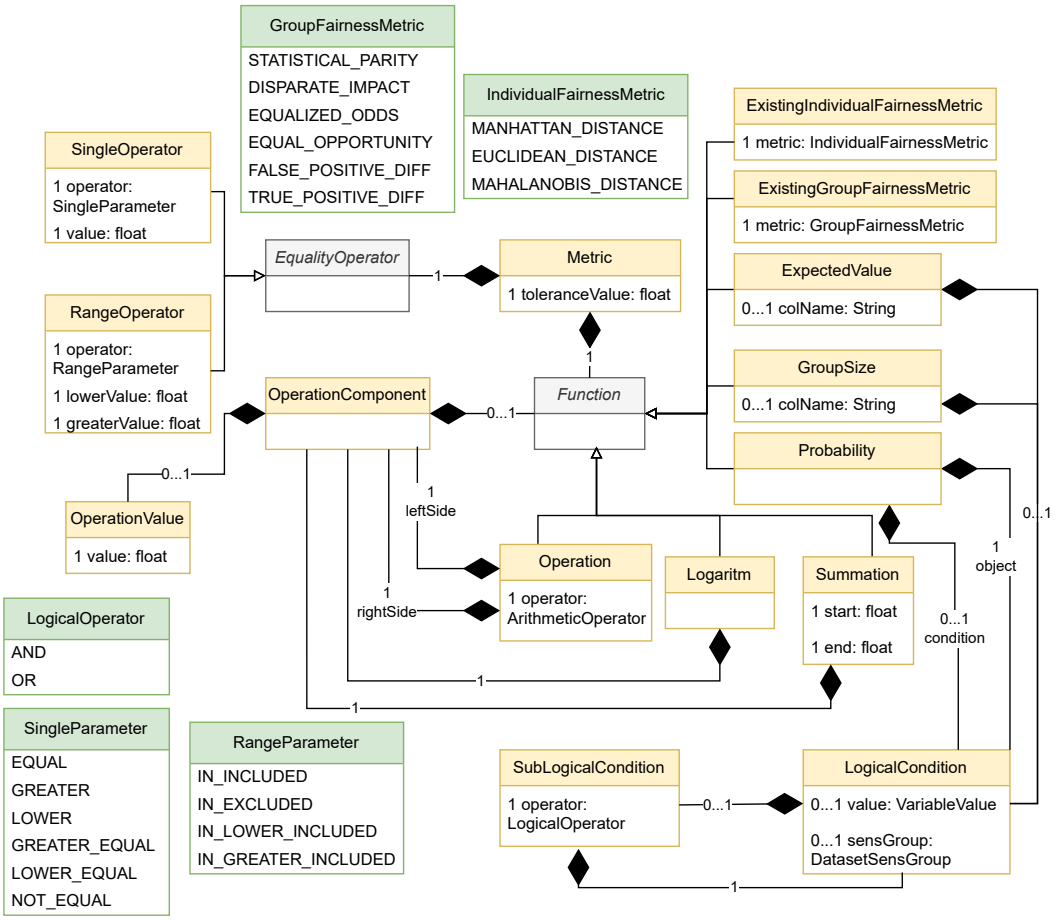


FIGURE 5.4: Metric Definition.

Figure 5.4 reports the portion of the metamodel dedicated to the metric definition. A `Metric` is composed of an `EqualityOperator` representing the threshold and has an attribute representing the `toleranceValue` (i.e., the level of bias tolerated by the system). The `EqualityOperator` can be a `SingleOperator` (e.g., "$= 0$" or "≤ 1") or a `RangeOperator` (e.g., "the metric must be < 1 and > 0"). Next, a `Metric` has a `Function` representing the actual metric implementation. Currently, the following functions are available:

`Operation` (representing a generic arithmetical operation), `Logarithm`, `Summation`, `ExpectedValue`, `GroupSize`, and `Probability`. A metric can also be based on `ExistingGroupFairnessMetric` and `ExistingIndividualFairnessMetric`. Such metaclasses represent the set of fairness metrics known in the literature and already implemented for group and individual fairness definitions, respectively.[3] For each function, we included a set of attributes needed for their implementation. In the future, new functions can be added by extending the `Function` metaclass.

5.4 Conclusion

In this chapter, we presented a formal modeling of the two workflows introduced in Chapter 2 to perform fairness assessments and identify the best combinations of ML models and fairness-enhancing methods. These formal models are the foundation for two low-code approaches discussed in Chapter 6.

Although we have presented them separately, those formalisms could be integrated in the future to further automate the development of fair learning-based systems. Specifically, the metamodel introduced in Section 5.3 allows for a high-level definition of bias for a given domain, as well as the creation of custom metrics. In contrast, the ExtFM outlined in Section 5.2 assists data scientists in selecting features that contribute to the generation of a comprehensive and correct experiment. By integrating these two formal models, we could effectively guide data scientists in designing experiments that help select the best ML models while also accommodating custom definitions of bias and custom fairness metrics.

[3] To select the set of metrics, we referred to the ones implemented in the AIF360 library [27].

Chapter 6

Low-Code Approaches for Software Fairness

In this chapter, we introduce two low-code approaches derived from the formal models discussed in Chapter 5. The first approach is MANILA, a web-based application designed to define and execute experiments aimed at identifying the optimal combination of ML models and fairness-enhancing methods (i.e., *fairness benchmarking workflow*) [46]. It is grounded on the ExtFM presented in Section 5.2.

The second approach is MODNESS, an MDE-based approach to define and execute fairness assessment workflows at different levels of abstraction [47]. MODNESS complements MANILA by allowing domain experts and data scientists to specify custom bias definitions and metrics. It is based on the bias and fairness metamodel presented in Section 5.3.

In the future, those two approaches can be combined to guide the data scientist through the definition and execution of a fairness benchmarking workflow, while providing high expressiveness in the fairness definition.

The tools presented in this chapter constitute the contribution CN_2 proposed to address the challenge CH_2.

This chapter is structured as follows: Section 6.1 is entirely devoted to present MANILA and the empirical evaluation we conducted. Section 6.2 focuses on MODNESS and its evaluation. Finally, Section 6.3 discusses the proposed approaches and concludes the chapter.

6.1 MANILA

In this section, we present MANILA, a low-code framework designed to formalize and execute the workflow outlined in Section 2.2. This framework automates the evaluation of various ML models and fairness-enhancing methods, helping to select the one that offers the best trade-off.

Our approach aims to automate and ease the fairness benchmarking workflow, making it accessible also to data scientists who are less expert in fairness.

As outlined in Section 5.2, each fairness benchmarking workflow comprises a set of features acting as *variation points*, which differentiate them from one another. For this reason, we can think of this family of experiments as a Software Product Line (SPL) specified by an Extended Feature Model [48].

Figure 6.1 details a high-level picture of MANILA, where each rounded box represents a step in the fairness benchmarking workflow, while square boxes represent artifacts. MANILA has been implemented as a low-code web-based application through which all the steps of the fairness benchmarking workflow can be performed. Near each artifact, we report the tools involved in its implementation.

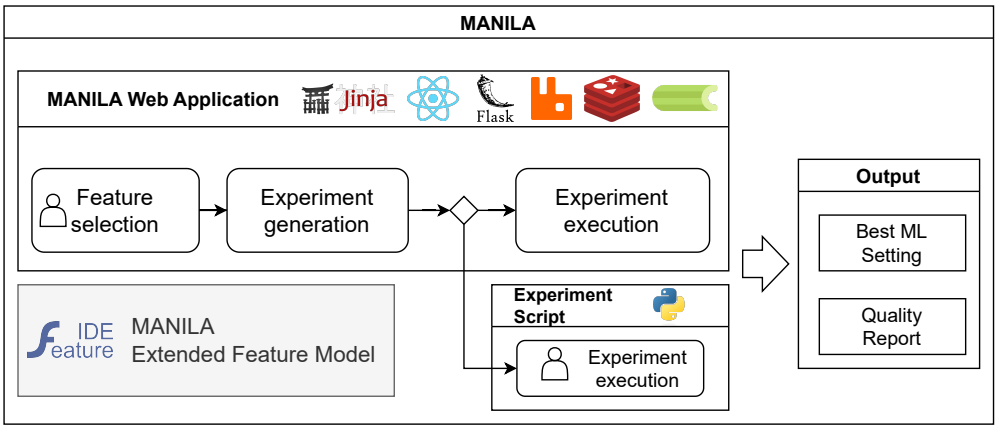


FIGURE 6.1: MANILA high-level overview

The first step in the development process is feature selection, in which the data scientist selects all the components of the fairness benchmarking workflow through a dedicated web form. This process may occur during the *model requirements* process of the workflow depicted in Figure 1.1. Next, a Python script implementing the experiment is automatically generated from the selected features. The generated experiment can be executed directly in the web application, or it can be downloaded and executed locally or embedded in other pipelines. After its execution, the experiment returns:

1. A quality report reporting for each fairness-enhancing method and ML algorithm the related metrics;

2. The best ML model with the applied fairness-enhancing method identified with the given trade-off strategy, trained with the full input dataset.

The architecture of MANILA makes it easy to extend. In fact, adding a new method or metric to MANILA translates to adding a new feature to the web form and adding the proper code implementing it.

The basis of MANILA is the Extended Feature Model (ExtFM) presented in Section 5.2. The ExtFM is the template of all possible experiments a data scientist can perform and guides them through the fairness benchmarking workflow. We used the ExtFM as a formalism to reason about the different relationships and constraints among features before implementing them in the web application.

MANILA is publicly available in the SoBigData research infrastructure (RI) [45] (after registration)[1], and its source code is available on GitHub[2].

In the following, we first describe in Section 6.1.1 the web application and how each workflow step has been implemented. Next, we present the empirical evaluation of MANILA in Section 6.1.2. Finally, we discuss possible threats to the validity of our proposed approach in Section 6.1.3

6.1.1 Web Application

The features and the constraints defined in the ExtFM presented in Section 5.2 have been implemented into a low-code web application, which is the core of MANILA.

[1] `https://sobigdata.d4science.org/group/sobigdata.it/manila-univaq`
[2] `https://github.com/giordanoDaloisio/manila-web`

Through the web application, it is possible to perform all the steps of the fairness benchmarking workflow described in Section 2.2. The web application has been implemented using the React Javascript library [177] for the front end and the Flask Python library [178] for the back end. Moreover, we employ the Celery Python library [179], with the RabbitMQ message broker [180] and the Redis database [181], to run the experiments asynchronously on the server. This way, we avoid overloading the server in case of multiple experiment runs. In the following, we detail how each workflow step has been implemented in the application.

Feature Selection

The feature selection step has been implemented in MANILA through a web form that includes all the features and the constraints defined in the ExtFM.

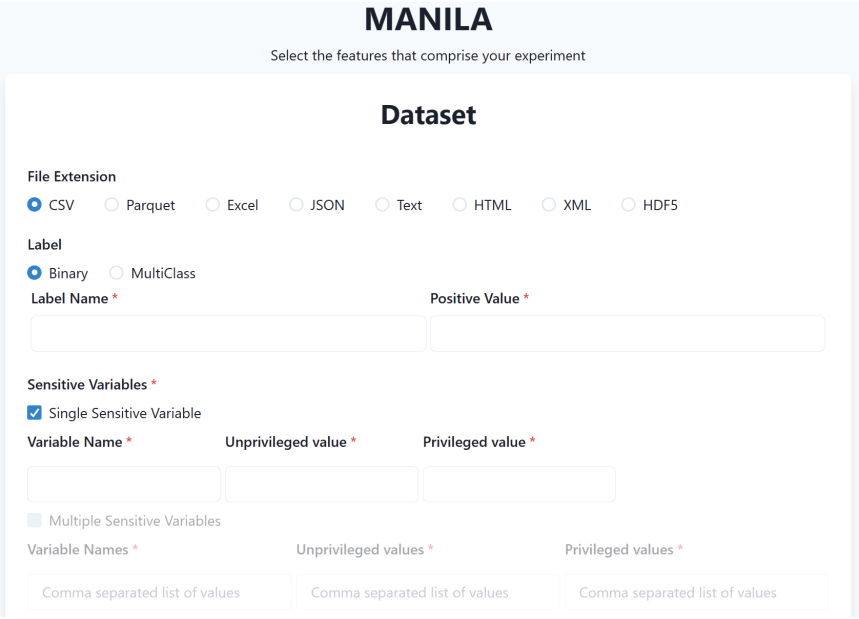


FIGURE 6.2: MANILA Web Form

Figure 6.2 shows a portion of the web form. The form comprises seven sub-forms (one for each macro feature defined in the ExtFM, see Section 5.2). Each sub-form includes all the *concrete* (i.e., non-abstract) children features of the relative macro feature in the ExtFM. For instance, Figure 6.2 shows the sub-form relative to the *Dataset* macro feature, which includes all children features such as *File Extension*, *Label*, and *Sensitive Variables*. Children with an *alternative* relationship in the ExtFM are implemented in the form either through a radio group (like the *File Extension* or *Label* fields in Figure 6.2) or by a logical condition among fields (for instance, in Figure 6.2 the *Multiple Sensitive Variables* field has been disabled because the *Single Sensitive Variable* field has been selected). In all other cases, features in the ExtFM have been implemented as checkbox fields in the form. Additional attributes related to the features (like the *Label Name* or *Positive Value* fields) have been implemented as text fields, which may be mandatory or not, depending on the case.

The cross-tree constraints defined in the ExtFM have been implemented as logical constraints among the different form fields. Figure 6.3 shows an example of such constraints. In the figure, it can be seen how the *Regression* field has been disabled. This is due to two reasons: first, the *Classification* ML task has already been selected, and second, the *Regression* task is incompatible with the fairness methods included

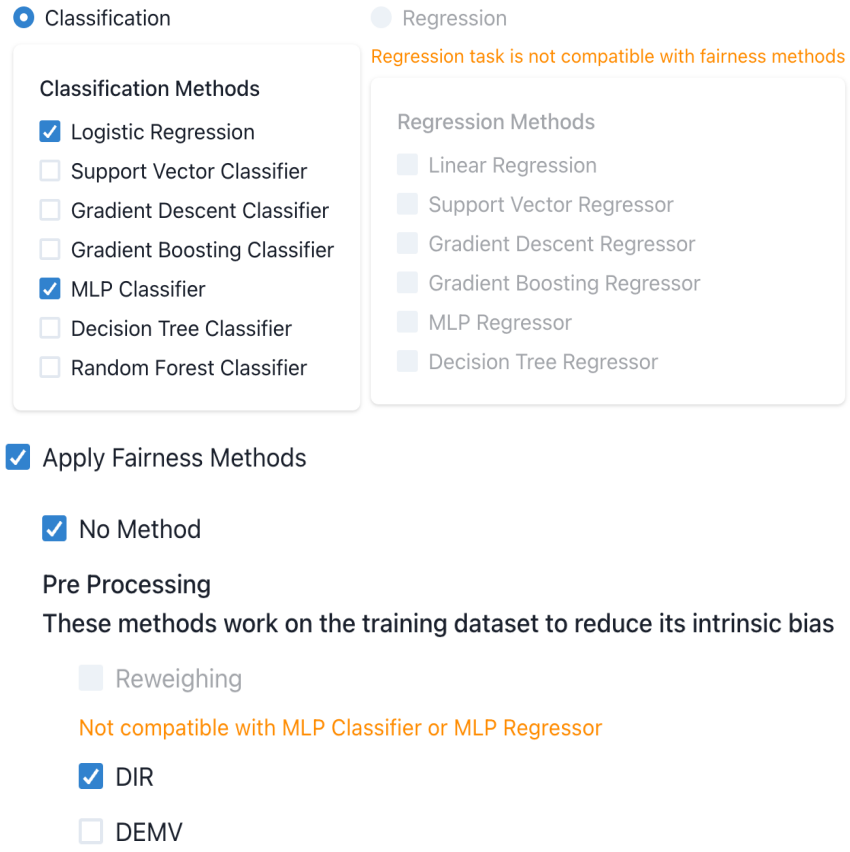


FIGURE 6.3: Example of web form cross-tree constraints

so far. Hence, since the *Fairness* quality property has been selected, regression ML methods can not be selected; otherwise, they would lead to a non-executable experiment. This constraint is also shown to the user through a message saying that "*Regression task is incompatible with fairness methods*". In addition, note how the *Reweighing* fairness method has been disabled as well. This is because the *Reweighing* method is not compatible with the *MLP Classifier* ML method that has already been selected. This constraint is also reported to the user, saying that Reweighing is "*not compatible with MLP Classifier or MLP Regressor*".

Concerning the selection of fairness metrics, we included a set of questions (inspired by [112]) to help the data scientist select the proper ones. Figure 6.4 reports the set of questions. The first question aims at identifying if the fairness definition to assess is an *individual* (i.e., similar individuals should be treated similarly) or *group* (i.e., individuals of a specific group should not be discriminated) fairness definition [19]. If the data scientist selects *individual*, then the three individual fairness metrics implemented in the `aif360` library (i.e., *Euclidean Distance*[182], *Manhattan Distance*[183] and *Mahalanobis Distance*[184]) are shown. Instead, if the data scientist chooses the *group* fairness definition, another question aims to identify the specific category of group fairness definitions. In particular, the data scientist has to specify if they are interested in *equal* fairness (i.e., everyone should have the same probability of receiving the positive label predicted [19]), *proportional* fairness (i.e., everyone should have the positive label predicted only if the other variables, different from the sensitive ones, tell that [19]), or *other* fairness definitions. Based on the selected definition, then a set of metrics is shown. Following previous work [112], for the *equal* category, we have included *Statistical Parity* and *Disparate Impact* fairness metrics

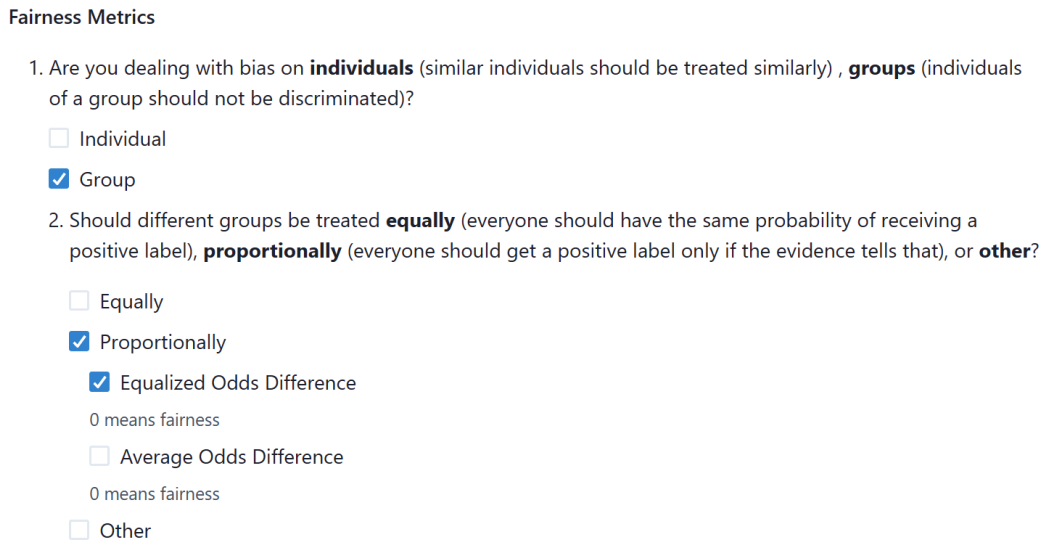


FIGURE 6.4: Fairness metric selection

[81]. For the proportional category, we have included the *Equalized Odds Difference* [83] and *Average Odds Difference* [116]. Finally, for the *other* category, we included the *True Positive Difference* and *False Positive Difference* [185].

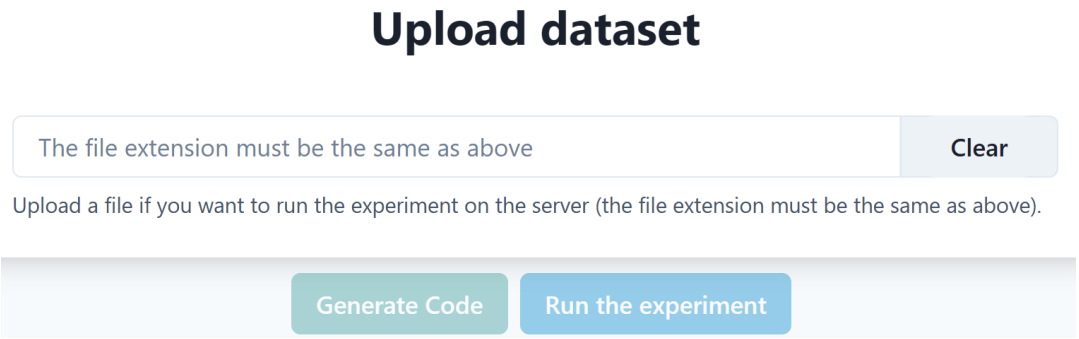


FIGURE 6.5: File upload field and execution buttons

Finally, Figure 6.5 shows the last field in the form that enables users to upload their dataset for running the generated experiment on the server. Below this field, two buttons are available: one for downloading the generated code and the other for executing the experiment on the server. It is important to note that, as shown in Figure 6.5, the buttons are disabled because not all the constraints outlined in the form have been met.

Experiment Generation and Execution

After selecting a set of features that meet all the form's constraints the experiment is generated and can be executed.

The generation process starts by either clicking on the *Generate Code* or *Run the experiment* buttons shown in Figure 6.5. The first button starts the code generation process and makes the generated code available for download. The second button executes the experiment directly on the server.

The experiment performs a grid search across all combinations of ML algorithms and fairness-enhancing methods, calculating the selected metrics for each combination. It is worth noticing how the space of the search (i.e., the possible combinations of ML models and fairness-enhancing methods) will always be relatively small, given the constraints imposed by the ExtFM. This allows to perform a complete search in a reasonable amount of time (depending on the size of input data).

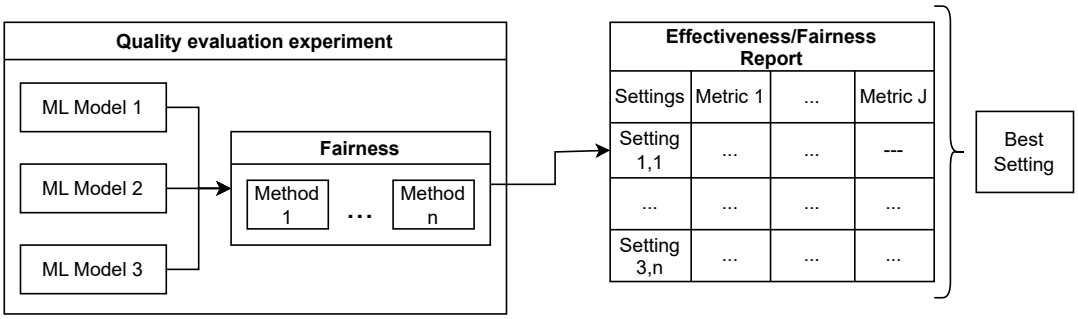


FIGURE 6.6: Example of benchmarking process

Figure 6.6 reports an example of how the fairness benchmarking process is done in MANILA. In this example, the data scientist has selected three ML models and n fairness-enhancing methods. In addition, they selected j metrics to evaluate the fairness and effectiveness of the ML methods. Thus, the testing process applies the n fairness methods to each ML algorithm and computes the j fairness metrics. Finally, the process returns a report synthesizing the results for fairness and effectiveness, along with the best setting selected using the chosen trade-off strategy.

```
from utils import cross_val
from methods import FairnessMethods
...

ml_methods = {
    'logreg': LogisticRegression(),
    'gradient': GradientBoostingClassifier(),
}
fairness_methods = {
    'no_method': FairnessMethods.NO_ONE,
    'preprocessing': [
        FairnessMethods.DEMV,
    ],
    'inprocessing': [
        FairnessMethods.EG,
        FairnessMethods.GRID,
    ],
    'postprocessing': []
}
base_metrics = {
    'stat_par': [],
    'eq_odds': [],
    'zero_one_loss': [],
    'disp_imp': [],
    'acc': [],
    'hmean': [],
}
for m in ml_methods.keys():
    model = Pipeline([
        ('scaler', StandardScaler()),
        ('classifier', ml_methods[m])
    ])
```

```
33      for f in fairness_methods.keys():
34          model = deepcopy(model)
35          data = data.copy()
36          if f == 'preprocessing':
37              for method in fairness_methods[f]:
38                  model_fair, ris_metrics = cross_val(..., model=
                        model, metrics=base_metrics, preprocessor=method
                        )
39                  ...
40          elif f == 'inprocessing':
41              for method in fairness_methods[f]:
42                  model_fair, ris_metrics = cross_val(..., model=
                        model, metrics=base_metrics, inprocessor=method)
43                  ...
44          elif f == 'postprocessing':
45              for method in fairness_methods[f]:
46                  model_fair, ris_metrics = cross_val(..., model=
                        model, metrics=base_metrics, postprocessor=
                        method)
47                  ...
```

LISTING 6.1: Portion of the generated experiment code

To have a more concrete visualization of how the experimental evaluation is conducted, Listing 6.1 reports a small portion of the generated Python code. In the code, there are three dictionaries containing the list of ML methods, fairness methods, and metrics to use - namely `ml_methods`, `fairness_methods`, and `base_metrics`, respectively (lines from 5 to 27 in Listing 6.1). Next, lines from 28 to 32 show a *for* loop exploring the list of ML methods, and, for each method, the function creates a pipeline including a preprocessing method (in this example, the `StandardScaler` preprocessing approach). Finally, a nested *for* loop explores the list of fairness methods, and, for each of them, a function performing the evaluation is called according to the fact that the fairness method is a *preprocessing*, *inprocessing*, or *postprocessing* one (lines from 33 to 47 in Listing 6.1). It is worth noting how the depicted generated experiment may vary based on the features selected. However, the overall structure is general and unrelated to the selected features.

If the user chooses to download the experiment, MANILA generates the corresponding Python implementation relying on the Jinja template engine [186]. Additionally, it generates the `environment.yml` file needed to create the *conda* environment with all the required libraries [187]. The experiment can be executed locally by running the following command:

```
$ python main.py -d <DATASET PATH>
```

Otherwise, it can be called through a REST API or any other interface such as a desktop application or a Scientific Workflow Management System like *KNIME* [188], [189]. This generality of our experimental workflow makes it very flexible and suitable for many use cases. After the execution, the code returns the quality report in CSV format. In addition, the best ML model is trained with the full input dataset and by applying the best fairness-enhancing method if selected. The ML model returned by the experiment is saved as a *pickle* file [190]. We have chosen this format since it is a standard format to store serialized objects in Python and can be easily imported into other scripts.

On the contrary, if the user chooses to run the experiment online, it is executed asynchronously using the Celery task queue system [179]. Figure 6.7 shows the online execution process. First, the frontend makes a call to the app controller through its REST API (step ① in Figure 6.7). The controller sends the task to a RabbitMQ

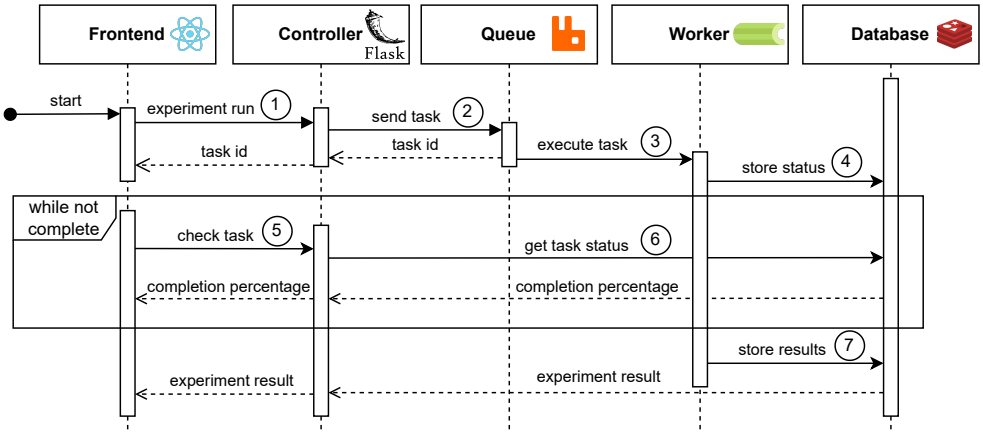


FIGURE 6.7: Experiment execution on the server

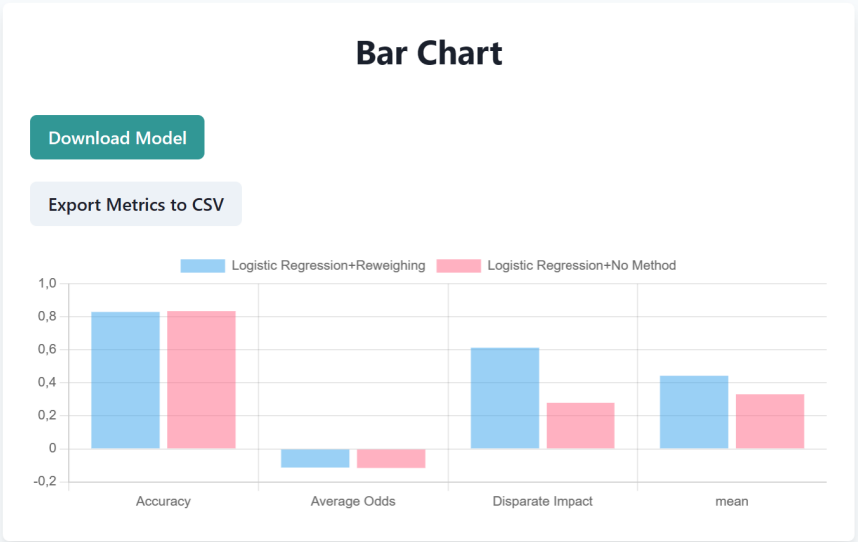
queue and returns the task ID to the frontend (step ② in Figure 6.7). Next, the task is executed by a Celery worker, and its status is stored in a Redis database (steps ③ and ④ in Figure 6.7). At the same time, until the experiment execution is not completed, the frontend periodically asks the controller about the task status (steps ⑤ and ⑥ in Figure 6.7). If the experiment is still in progress, a completion percentage is returned. Otherwise, when the execution is completed, the Celery worker saves the results on Redis (step ⑦ in Figure 6.7), and the results are returned to the frontend. Figure 6.8 shows the different elements of the results page. In this particular example, we used MANILA to evaluate the fairness and effectiveness of a *Logistic Regression* model alone and a *Logistic Regression* model with the application of the *Reweighing* preprocessing method [86]. We used the *Accuracy* metric to evaluate the effectiveness and the *Average (Equality) Odds* [83] and *Disparate Impact* [82] metrics to evaluate the fairness of the two settings. Finally, we adopted the *Statistical Mean* as the aggregation function to evaluate the fairness-effectiveness trade-off. The result page shows the metrics in a bar chart and in a tabular way (i.e., raw results). Figure 6.8a shows the bar chart along with two buttons to download the fully trained best ML model in *pickle* format and the computed metrics as a CSV file, respectively. From the bar chart, it can be seen how the Logistic Regression plus Reweighing combination (the blue bar in Figure 6.8a) performs better because it achieves a higher value of Disparate Impact (the closer this metric is to one, the more fair is the model) and a higher value of Statistical Mean. The same results are shown in a tabular format as displayed in figure 6.8b. In this case, the best combination is highlighted in green in the table. Finally, figure 6.8c shows a short description of how to read and interpret the different metrics.

6.1.2 Evaluation

In this section, we describe the evaluation we performed on MANILA. To this aim, we formulate the following research questions (RQ):

RQ_1 Can MANILA effectively assist in conducting real-world fairness evaluations?

RQ_2 To what extent the results returned by MANILA are in line with base-lines?



(A) Metrics bar chart

Raw results

FAIRNESS METHOD	MACHINE LEARNING MODEL	ACCURACY	AVERAGE ODDS	DISPARATE IMPACT	MEAN
Reweighing	Logistic Regression	0.83	-0.11	0.61	0.44
No Method	Logistic Regression	0.83	-0.12	0.28	0.33

(B) Raw results

How to read the metrics

- **Classification Metrics**
  These metrics are used to evaluate the performance of a classification and have an optimal value equal to **one**.
  The only exception is **Zero One Loss** which has an optimal value equal to **zero**.
- **Regression Metrics**
  These metrics measure the error of the predictions of the model and have an optimal value equal to **zero**.
- **Fairness Metrics**
  These metrics measure the fairness of the predictions of the model and have an optimal value equal to **zero**.
  The only exception is **Disparate Impact** which has an optimal value equal to **one**.

(C) Metrics description

FIGURE 6.8: MANILA result page

Following a previous work [112], we evaluate MANILA both in terms of *expressiveness* (RQ_1) and *correctness* (RQ_2) by reproducing the experimental evaluation described in Chapter 4 for the DEMV algorithm. More in detail, the *expressiveness* is assessed by proving that MANILA can replicate the selected fairness evaluation. In contrast, *correctness* is assessed by showing that the results obtained with the experiments generated by MANILA are comparable to the original ones.

TABLE 6.1: Replicated experiments

Experiment	ML Settings	Datasets	Metrics
1	LogReg LogReg + EG LogReg + Grid LogReg + DEMV	CMC Crime Drug Law Park Wine	SP AO DI ZO Loss Acc H-Mean
2	Gradient Gradient + EG Gradient + Grid Gradient + DEMV	CMC Law	SP AO DI ZO Loss Acc H-Mean
3	SVM SVM + EG SVM + Grid SVM + DEMV	CMC Law	SP AO DI ZO Loss Acc H-Mean

We replicate three specific experiments performed on DEMV (i.e., the ones shown in Tables A.10, A.12, and A.13), which are synthesized in Table 6.1. We have chosen to replicate these experiments among all the ones conducted because they provide the highest combination of ML methods, fairness methods, and metrics.

The first replicated experiment is the evaluation of four different ML settings (i.e., *Logistic Regression (LogReg)* alone, *Logistic Regression* plus *Exponentiated Gradient (EG)*, *Logistic Regression* plus *Grid Search (GRID)*, and *Logistic Regression* plus *DEMV*) on six different datasets (i.e., *Contraceptive Method Choice (CMC)* [155], *Communities and Crime (Crime)* [156], *Drug Usage (Drug)* [158], *Law School Admission (Law)* [74], *Parkinson Telemonitoring (Park)* [159], and *Wine Quality (Wine)* [160]). The other two replicated experiments involve the same fairness methods as the first one but employ two different ML classifiers (i.e.,, *Gradient Boosting (Gradient)* and *Support Vector Machines (SVM)*, respectively). In this case, the evaluations are applied only to the *CMC* and *Law* datasets. In all experiments, we consider two sensitive variables for each dataset. The metrics involved are: *Statistical Parity (SP)*[81], *Equalized Odds (EO)*[83], *Zero One Loss (ZO Loss)*[147], *Disparate Impact (DI)*[82], *Accuracy (Acc)*[155], and *Harmonic Mean (H-Mean)*[149] as aggregation function.

The *expressiveness* is evaluated by assessing if MANILA can reproduce the described experiments correctly. The *correctness* is evaluated by assessing if the results of the experiments generated by MANILA are close to the ones reported in the original experiments. More in detail, following previous work [112], the results of the generated experiments should be within the standard deviation range of the results reported in Tables A.10, A.12, and A.13, and there should not be a statistically significant difference between the results.

RQ_1: Expressiveness Evaluation

Concerning the *expressiveness* of MANILA, we were able to correctly reproduce all the experiments reported in Table 4.1.[3] In particular, following the steps described in Section 6.1.1, we first specified the features of the experiments listed in Table 4.1 from the graphical interface of MANILA. Next, we downloaded the generated codes to execute them locally. Finally, we ran the experiments to obtain the results. In total, we generated and executed ten different experiments, one for each dataset reported in Table 4.1.

Being a low-code platform, MANILA does not require to write any line of code to implement the given experiments. In contrast, the original experiments required almost 200 lines of code, as seen in the repository linked in the original paper.[4].

Answer to RQ_1: MANILA presents a high level of *expressiveness* that allows it to implement real-world fairness evaluations involving real-world datasets and to replicate previous experiments.

RQ_2: Correctness Evaluation

Concerning the *correctness* of the generated experiments, Table 6.2 reports, for each metric of each experiment, the *p-values* of the non-parametric Kruskal-Wallis H test performed between the results of the original experiments and the ones obtained by executing the code generated by MANILA.

TABLE 6.2: *p-values* of the Kruskal-Wallis H test for each experiment

	SP	**AO**	**ZO Loss**	**DI**	**Acc**	**H-Mean**
Exp. 1	0.84	0.08	0.84	0.45	0.38	0.53
Exp. 2	0.10	0.71	0.31	0.43	0.27	0.71
Exp. 3	1.00	0.57	0.16	0.96	0.72	0.75

From the table, it can be seen that all the *p-values* are > 0.05, meaning that all the metrics obtained by running the code generated by MANILA are not statistically different from the original ones. In addition, Figure 6.9 reports a comparison of the aggregated h-means of the three experiments.[5] As shown, on average, the results of the three experiments generated by MANILA are very close to the original ones and are within the standard deviation range.

Answer to RQ_2: The *correctness* of the code generated by MANILA allows to correctly replicate baseline fairness evaluations by obtaining results that are statistically equal to the original ones.

[3]The full replication package of the experiment is available at the following link `https://github.com/giordanoDaloisio/manila/tree/main/replication_package`

[4]The original code of the reproduced experiment is available here `https://github.com/giordanoDaloisio/demv/blob/main/replication_package/src/generatemetrics.py`

[5]We visualize only the h-mean because, being an aggregated value, it can be considered as a synthesis of the other selected metrics

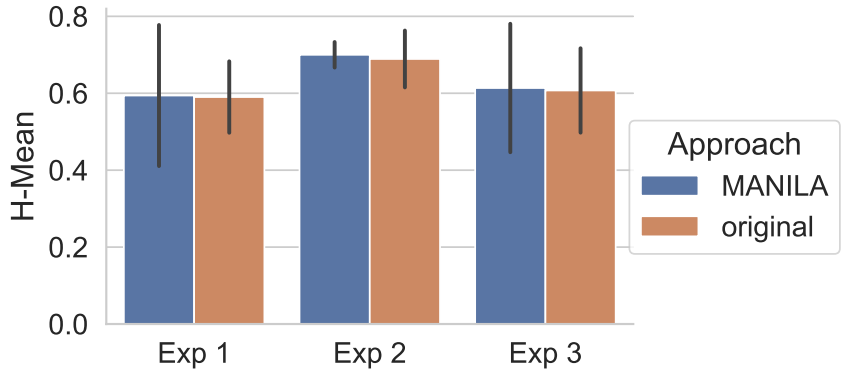


FIGURE 6.9: Aggregated H-Means of the original experiments and MANILA's ones

6.1.3 Threats to Validity

This section discusses possible threats that can hamper the results of the performed evaluation.

Internal validity concerns internal factors that can influence the results of our evaluation. The code generated by MANILA could contain implementation errors. To mitigate this threat, we referred to the most adopted fairness library (i.e., AIF360) for selecting the fairness methods and metrics to include in the tool. In addition, we have shown how the results returned by MANILA are statistically comparable with the original ones. Another threat concerns the fact that there could be other methods or metrics currently not included in MANILA. To address this threat, we have shown how MANILA has a level of *expressiveness* able to model different real-world use cases. In addition, MANILA can be easily extended to include other methods or metrics by adding a new entry in the feature model and its actual implementation in the code generation template.

External validity threats concern the generalizability of our approach. In this respect, there could be some real-world use cases that can not be implemented in MANILA. In particular, at this stage of work, we are not considering *individual* fairness definitions, but only *group* fairness definitions [19]. However, we have re-implemented an extensive evaluation involving different ML methods, fairness methods, metrics, and datasets to show how MANILA can manage different *group* fairness evaluations, and, in our future works, we will extend MANILA to include also *individual* fairness definitions.

6.1.4 Limitations

While we have demonstrated that MANILA offers a high level of expressiveness in replicating fairness development workflows, it also has some limitations. Specifically, modeling the workflow as an SPL provides valuable guidance to data scientists and promotes the reuse of existing features. However, it may also restrict the ability to model fairness development workflows in contexts that differ from traditional ones. This is, for instance, the case for the *TPL* use case presented in Chapter 2, where the authors of the related paper adapted the *coverage* metric from the recommender systems domain to measure popularity bias. This metric is not a traditional metric from the fairness domain. Thus, it has not been included in the ExtFM and in the subsequent web implementation of MANILA.

We aimed to address this limitation with MODNESS, a model-driven framework designed to perform more extensive and comprehensive bias assessment workflows.

6.2 MODNESS

In this section, we introduce MODNESS, a model-driven framework designed to conceptualize, design, implement, and execute the fairness assessment workflow presented in Chapter 2 and illustrated in Figure 2.1. MODNESS complements MANILA by allowing a more extensive and comprehensive definition of bias. In particular, it allows the specification of high-level definitions of bias for a given domain and the modeling of custom metrics for fairness assessment.

MODNESS can be characterized as a two-layered framework. At its core lies an abstract bias definition upon which multiple fairness analyses can be defined and built.

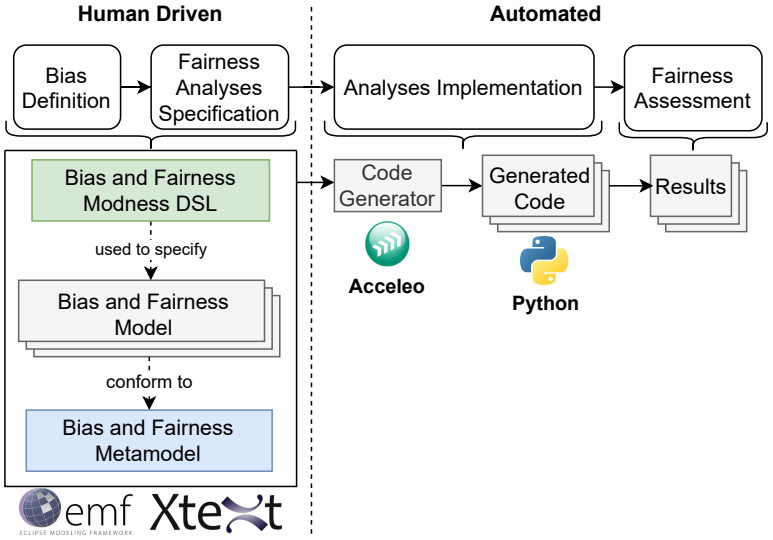


FIGURE 6.10: MODNESS high-level view.

Figure 6.10 provides a high-level overview of MODNESS: round boxes represent the four primary steps of the fairness assessment workflow outlined in Section 2.1, while square boxes depict the artifacts that are either developed or automatically generated. Adjacent to each artifact, we also indicate the technologies employed for its implementation. The fairness assessment workflow within MODNESS can be divided into two main phases: a *Human Driven* phase, which involves user interaction (as described in Section 2.1), and an *Automated* phase, which operates without direct user involvement.

To initiate the fairness assessment process with MODNESS, the initial step involves defining bias by specifying the *sensitive variables*, *privileged and unprivileged groups*, and *positive outcome*. Recalling the *University* and *TPL* use cases defined in Chapter 2, the bias definition for *University* could include *gender* as the sensitive variable, *men and women* as the privileged and unprivileged groups respectively, and *positive admission* as the positive outcome. For the *TPL* use case, a bias definition can comprise *popularity* as the sensitive variable, *popular and unpopular* libraries as the privileged and unprivileged groups, and *recommendation* as the positive outcome.

Note how these bias definitions are generic in this phase and unrelated to any specific dataset.

Subsequently, multiple analyses can be constructed based on a definition of bias. A *fairness analysis* tailors a specific bias definition to a particular dataset with a defined scope and associated fairness metrics. These fairness metrics can be established in existing literature or custom-defined by the user. Note how the dataset may also include the predictions of an ML model in one of its columns also allowing the fairness assessment of the model's predictions (similarly to other fairness toolkits like Aequitas or Themis [95], [191]). Recalling our use cases, a fairness analysis for *University* can be made of a dataset containing information about the gender of each student (mapped, for instance, in a column named *sex*)[6] and the admission outcome (for instance, in a column named *admission*), the scope will be *equal probability for men and women to be admitted*, and the metric is *Statistical Parity*. Concerning the *TPL* use case, a fairness analysis can comprise a dataset with the popularity of each library (for instance, in a column named *frequency*) and a recommendation score (for instance, in a column named *recommendation*). The recommendation score is eventually used by the recommender system to identify the items more suited for a recommendation. For instance, the system may recommend only the items with a score higher than 80% of all other libraries. The scope of the analysis will be that *each library must be recommended despite its popularity*, and the metric adopted is the custom *coverage* metric from the recommender systems domain [192]. The traditional *coverage* metric measures the number of items being recommended over the total amount of items [84]. This variation measures the amount of *unpopular* libraries recommended over the whole recommendations. It is defined as: $|L_{unpop}|/|L|$ where $|L_{unpop}|$ is the number of *unpopular* (i.e., *non-frequent*) libraries recommended and $|L|$ is the whole set of recommendations [75]. The closer this metric is to 1, the more the system is free from popularity bias.

These specification steps are implemented in MODNESS as a model-driven approach, utilizing the EMF ecosystem [165]. In this phase, the output is a model (referred to as the *Bias and Fairness Model* in Figure 6.10) that includes both the bias and the corresponding fairness analysis definitions. This model adheres to the *Bias and Fairness Metamodel* presented in Section 5.3, which serves as the foundational structure of MODNESS.

Starting from a model describing the bias and the related fairness analyses definitions, MODNESS automatically generates an implementation of them through a code generator based on Acceleo [167]. In particular, MODNESS generates a Python code that automatically checks the fairness of a given dataset using the information provided in the fairness analysis specification. The analyses implementation and the fairness assessment steps comprise the automated phase of MODNESS and do not require direct human intervention.

To support the specification of bias and fairness models, we developed a domain-specific language and its corresponding environment utilizing Xtext technology [166]. This way, users can rely on a textual concrete syntax to specify all the concepts needed in the traditional fairness workflow. In the forthcoming section, we present the textual MODNESS specification for two explanatory use cases, i.e., *University* and *TPL*. The Xtext grammar of the MODNESS DSL is available online at [49].

[6]Note how we refer to the original column names of the dataset related to this use case

In the following, we detail MODNESS by first describing the implemented Domain Specific Language (DSL) in Section 6.2.1 and, next, describing the code generation and fairness assessment processes in Section 6.2.2.[7]

6.2.1 Domain Specific Language

MODNESS DSL has been generated starting from the metamodel presented in Section 5.3. Thus, following the metamodel structure, it can be divided into three macro components. In the following, we present them by showing two implementations for the *University* and *TPL* use cases.

Bias Definition

```
1   GroupBias "university"{
2     Definition: {
3       domain: "education";
4       source: WRONG_SAMPLING;
5       sensitiveVariables: {
6         SensitiveVariable{
7           name: "gender";
8           values: "male","female";
9         }
10      };
11      positiveOutcome: "positive admission";
12      unprivilegedGroup: {
13        SensitiveGroup{
14          name: "women";
15          sensitiveValue: "gender.female";
16        };
17      };
18      privilegedGroup: {
19        SensitiveGroup{
20          name: "men";
21          sensitiveValue: "gender.male";
22        };
23      };
24    };
25
```

LISTING 6.2: Bias definition example for the *University* use case.

Listing 6.2 shows an example of MODNESS bias definition related to the *University* use case. Since this use case is about group bias, the model's root is an instance of the `GroupBias` metaclass. Next, the first portion of the model consists of a *definition*, which specifies all the high-level components of a bias definition. For this use-case, the domain can be *education*, and the source can be *wrong sampling* (e.g., wrong data have been used to train the ML model). Recalling the general workflow described in Section 2.1, to give a high-level definition of group bias, we must provide the sensitive variables, the positive outcome, and the privileged and unprivileged groups. For this scenario, we have only one sensitive variable representing the *gender*, which has two values, i.e., *male* and *female*.[8] Next, the *positive outcome* is represented by a *positive admission*. Finally, the *unprivileged group* is *women* and has a reference to the *female* sensitive value (indicating that this group is identified by that specific value of the sensitive variable *gender*). On the contrary, the *privileged group* is *men* and has a reference to the *male* sensitive value.

[7] The source code of MODNESS is available in our replication package [49]

[8] Note how the values *male* and *female* are instances of the `SensitiveVarValue` metaclass. However, in the DSL, they are represented as `values` attributes of the `SensitiveVariable` metaclass so as not to burden the overall syntax.

```
GroupBias "TPL"{
  definition: {
    domain: "recommender systems";
    source: WRONG_ALGORITHM_BEHAVIOUR;
    sensitiveVariables: {
      SensitiveVariable{
        name: "popularity";
        values: "popular","unpopular";
      }
    };
    positiveOutcome: "recommendation";
    unprivilegedGroup: {
      SensitiveGroup{
        name: "unpopular libraries";
        sensitiveValue: "popularity.unpopular";
      };
    };
    privilegedGroup: {
      SensitiveGroup{
        name: "popular libraries";
        sensitiveValue: "popularity.popular";
      };
    };
  };

```

LISTING 6.3: Bias definition example for the *TPL* use case.

Listing 6.3 reports instead an example of bias definition for the *TPL* use case. Note how the main components of a bias definition are always the same regardless of the domain (i.e., *group* or *individual* bias, *sensitive variables*, *positive outcome*, and *sensitive groups* if the definition is a group bias). The root of the model is a `GroupBias` class where the domain is *recommender systems*, and the source of bias could be *wrong algorithm behaviour*. The *sensitive variable*, in this case, is represented by *popularity* its possible values are *popular* and *unpopular*. The positive outcome is a *recommendation* from the system. Finally, the *unprivileged* group is represented by unpopular libraries, whereas the *privileged* group is represented by popular libraries.

Fairness Analysis

```
analysis: {
  GroupAnalysis{
    scope: "all people must have
        same admission
        probability despite gender";
    dataset: {
      Dataset {
        id: 'admissions';
        predictedLabelName: 'admitted';
        filePath: 'admissions.csv';
        positiveOutcome: {
          id: "admission";
          mappingOutcome: "positive admission";
          value: {
            operator: EQUAL;
            value: 1.0;
          };
        };
      datasetSensitiveVariable: {
        DatasetSensitiveVariable{
        name: "sex";
        mappingSensitiveVariable: gender;
        values: {
          SensitiveVariableValue{
            id: "female";
            mappingValue: "gender.female";
```

```
27                value: {
28                  operator: EQUAL;
29                  value: 0.0;
30                };
31              },
32              SensitiveVariableValue{
33                id: "male";
34                mappingValue: "gender.male";
35                value: {
36                  operator: EQUAL;
37                  value: 1.0;
38                };
39              }
40              }
41            }
42          };
43        }
44      };
45      datasetUnprivilegedGroup: {
46          id: 'women';
47          mappingGroup: women;
48          sensitiveVariables:
49          ("admissions.sex.female");
50      };
51      datasetPrivilegedGroup: {
52        id: 'men';
53        mappingGroup: men;
54        sensitiveVariables:
55        ("admissions.sex.male");
56      };
57
```

LISTING 6.4: Fairness analysis example for the *University* use case.

Listing 6.4 shows the fragment of the model related to the analysis definition for the *University* scenario. We recall that an analysis implements a high-level bias definition and maps each abstract component into a real feature of a dataset. In this example, since we start from a `GroupBias` definition, the analysis is modelled as an instance of the `GroupAnalysis` metaclass. The scope of the analysis is that *"all people must have the same admission probability despite their gender"*. The analysis comprises a `Dataset` class, which models the dataset that will be used in the analysis. In particular, the model's predictions (i.e., admission of students) are stored in a column named *admitted* (see line 9 of Listing 6.4). As specified in the bias definition, a positive outcome for this use case is represented by a *positive admission*, which is encoded with a value of 1.0 in the *admitted* column of the dataset. This information is represented in the model by the `positiveOutcome` attribute of the Dataset class, which says that the *positive admission* positive outcome is represented with a *value* equal to 1.0 (lines from 11 to 18).

Next, we need to associate each *sensitive variable* with specific columns and values in the dataset. More specifically, the model defines the `"sex"` column as representing the *gender* sensitive variable, where *female* is coded as a value of 0.0, and *male* is coded as a value of 1.0 (referenced in lines 19 to 44).

Furthermore, the model identifies the unprivileged group as *women* within the dataset, which corresponds to instances with a value of 0.0 in the `"sex"` column (this concept is captured within the model through a linkage to the `SensitiveVariableValue` class, marked by the ID `"admissions.sex.female"`). Similarly, the privileged group is represented by instances that have a value of 1.0 in the dataset (covered in lines 45 to 56).

```
1     analysis: {
2       GroupAnalysis{
3         scope: "relevant libraries must
4             be recommended despite
```

```
5             their popularity";
6         dataset: {
7           Dataset {
8             id: 'recommendations';
9             predictedLabelName: 'ranking';
10            filePath: 'recommendations.csv';
11            positiveOutcome: {
12              id: "high-ranking";
13              mappingOutcome: recommendation;
14              value: {
15                operator: GREATER_EQUAL;
16                value: 0.8;
17              };
18              relativeToDatasetSize;
19            };
20          datasetSensitiveVariable: {
21            DatasetSensitiveVariable{
22              name: "frequency";
23              mappingSensitiveVariable:
24              popularity;
25              values: {
26                SensitiveVariableValue{
27                  id: "non-frequent";
28                  mappingValue:
29                  "popularity.unpopular";
30                  value: {
31                    operator: MINOR_EQUAL;
32                    value: 0.8;
33                  };
34                  relativeToDatasetSize;
35                },
36                SensitiveVariableValue{
37                  id: "frequent";
38                  mappingValue:
39                  "popularity.popular";
40                  value: {
41                    operator: GREATER;
42                    value: 0.8;
43                  };
44                  relativeToDatasetSize;
45                }
46              }
47            }
48          };
49        }
50      };
51      datasetUnprivilegedGroup: {
52          id: 'non-frequent libraries';
53          mappingGroup: "unpopular libraries";
54          sensitiveVariables: ("recommendations.frequency.non-frequent");
55      };
56      datasetPrivilegedGroup: {
57        id: 'frequent libraries';
58        mappingGroup: "popular libraries";
59        sensitiveVariables: ("recommendations.frequency.frequent");
60      };
61
```

LISTING 6.5: Fairness analysis example for the *TPL* use case.

Listing 6.5 shows the analysis definition for the *TPL* use case. Similar to the *University* scenario, this analysis maps each component of the bias definition into concrete features of the dataset under analysis. We start with an instance of the `GroupAnalysis` metaclass. An instance of the `Dataset` metaclass models the dataset used for the analysis, with the predicted rank stored in the *ranking* column. This information is thus reported as an attribute of the `Dataset` class (line 9). Additionally, based on inputs from the domain expert, we know that the system recommends a library if its ranking exceeds 80% of the predicted ranks (i.e., if the libraries are ordered in descending rank order, only the top 20% are recommended) [75]. This can

be modelled in MODNESS by specifying in the `positiveOutcome` attribute that a *recommendation* positive outcome is encoded with a value greater than or equal to 0.8 for the *ranking* class. The `relativeToDatasetSize` keyword indicates that we are using relative values (i.e., percentage) rather than absolute ones (lines 11-19).

Similarly, assume that the domain expert specifies that a library is *popular* if it appears in many projects [75]. In particular, the dataset has a column named *frequency* containing the number of projects in which it appears, and a library is *popular* if its frequency is higher than 80% of all libraries. Hence, first, an instance of the `DatasetSensitiveVariable` metaclass maps the *popularity* sensitive variable to the *frequency* column in the dataset (lines 21-24). Next, as done for the ranking, the model maps *popular* libraries to values greater than 0.8 using the `relativeToDatasetSize` keyword and *unpopular* libraries to values lower or equal to 0.8 of the entire dataset (lines 25-45). Finally, the model reports how the unprivileged group is identified in the dataset with the *non-frequent* sensitive variable value, while the privileged group is identified in the dataset with the *frequent* sensitive variable value (lines 51-60).

Metric Definition

```
1    metric: {
2        Metric{
3          name: "StatisticalParity";
4          toleranceValue: 0.2;
5          function: ExistingGroupFairnessMetric {
6            metric: STATISTICAL_PARITY;
7          };
8          optimalValue: {
9            operator: EQUAL;
10           value: 0.0;
11         };
12       };
13         };
14
```

LISTING 6.6: Metric definition example for the *University* use case.

Listing 6.6 shows the fragment of the model devoted to the metric definition for the *University* use case. In this scenario, based on the scope of the analysis, the data scientist suggests using the *Statistical Parity (SP)* metric to assess fairness. SP is a widely adopted metric in the fairness literature that measures the probability of an item receiving a positive prediction, whether it is in the privileged group or not [81]. A value of 0 means fairness. This metric is included among the possible values for the `ExistingGroupFairnessMetric` metaclass. Hence, to model this information in MODNESS, a user first has to define an instance of the `Metric` metaclass and set a tolerance value, e.g., 0.2 [82]. Next, they have just to define an instance of the `ExistingGroupFairnessMetric` metaclass and set its value to `STATISTICAL_PARITY`. Finally, this metric's optimal value is reported as equal to 0.0.

```
1    Metric{
2        name: "coverage";
3        toleranceValue: 0.2;
4        function: Operation{
5          arithmeticOperator: RATIO;
6          leftSide: {
7            function: GroupSize{
8              groupCondition: {
9                sensitiveGroup: "non-frequent libraries"
10               AND value:"recommendations.high-ranking"
11             };
12           };
13         };
```

```
14          rightSide: {
15            function: GroupSize{
16              groupCondition: {
17                value: "recommendations.high-ranking"
18              };
19            };
20          };
21        }
22        optimalValue: {
23          operator: EQUAL;
24          value: 1.0;
25        };
26      };
27
```

LISTING 6.7: Metric definition example for the *TPL* use case.

Listing 6.7 reports instead an implementation of a metric for the *TPL* use case. In this case, the data scientist suggests using a custom metric to assess the amount of bias in the system. In particular, they suggest using an adaptation of the *coverage* metric from the recommender systems domain to measure popularity bias [192]. Recall, from the definition given at the beginning of this section, that this metric is defined as the ratio of *unpopular* items recommended over the whole recommendations, i.e., $|L_{unpop}|/|L|$. Since this metric is not a traditional metric from the fairness literature, it is not included among the possible values for the `GroupFairnessMetric` enumeration. However, it can be modeled in MODNESS as follows: first, create an instance of the `Metric` metaclass, which will contain the custom metric and set its tolerance value, for instance, 0.2 (lines 2-3 in Listing 6.7). Next, recalling the definition given above, this metric is a ratio between two values. Hence, we create an instance of the `Operation` metaclass with the `arithmeticOperator` attribute set to `RATIO` (line 5). Next, we have to model the left and right sides of the ratio (i.e., its numerator and its denominator). The numerator is the number of *non-frequent* libraries being recommended. This information can be modeled in MODNESS by first creating an instance of the `GroupSize` metaclass, which represents a function to count the number of items in a group (line 7). Next, we have to define the set of items that have to be counted by the `GroupSize` function (i.e., the *non-frequent* items being recommended). So, we define a set of boolean conditions that can be used to select a set of items from the dataset. In this case, we have two logical conditions connected with an `AND`. The first logical condition selects items from the *non-frequent* sensitive group, while the second logical condition selects items having the *high-ranking* positive outcome (lines 9-10). The denominator is instead the number of items having a *high-ranking*. This information can be modeled similarly to the numerator by creating an instance of `GroupSize` metaclass, this time filtering only items with a *high-ranking* (lines 15-19). Finally, it is reported that the threshold for this metric is equal to 1.0 (lines 22-25).

6.2.2 Code generation and fairness assessment

After defining fairness and its corresponding metrics, MODNESS generates the fairness assessment implementation using a source code generator developed with Acceleo. This generator exploits specific templates to create static and dynamic parts of the target code by incorporating queries on the source models. Acceleo templates leverage a defined syntax to specify conditions or iterations over elements in the input models. Listing 6.8 depicts an excerpt of the developed Acceleo template.[9]

[9]The developed Acceleo-based code generator is available on the MODNESS replication package [49]

```
file_path = '[biasModel.dataset.filePath/]'
predicted_label_name = '[biasModel.dataset.predictedLabelName/]'
ground_truth_label_name = '[biasModel.dataset.groundTruthLabelName/]'
...
[if biasModel.metric -> size()>0]
[for (metric: Metric | biasModel.metric)]
[if (metric.operator.oclIsTypeOf(SingleOperator))]
threshold = [metric.operator.oclAsType(SingleOperator).value/]
[/if]
[/for]
[/if]
```

LISTING 6.8: Fragment of an explanatory MODNESS Acceleo template.

To support the generation phase, we rely on Pandas [193] and AI360 Python [27] libraries to preprocess and support fairness analysis, respectively. In particular, MODNESS exploits three different Acceleo templates developed to support each phase of the process, i.e., bias definition, fairness analysis specification, and metric definition. To cover the first phase, the generated code supports the high-level specification of the analyzed fairness scenario, including the sensitive variables, the expected positive outcome, and the parameters needed to feed the selected metric. In such a way, the configuration is compliant with the user-defined specification. Afterward, the fairness analysis phase could be conducted by means of a set of predefined implementations of the state-of-the-art fairness metrics (taken from the AIF360 library [27]). Alternatively, MODNESS can generate operator specifications that define and compose different metrics.

```
from fairnessMetric import FairnessMetric
import pandas as pd

# INPUT DATA
file_path = "data/admissions.csv"
predicted_label_name = "admission"
data = pd.read_csv(file_path)
indexes = ["sex"]
dataset_unprivileged_group = {"sex": 0}
dataset_privileged_group = {"sex": 1}

# PARAMETERS
dataset_positive_outcome = 1
threshold = 0.0
tolerance_value = 0.2

# FAIRNESS ASSESSMENT
metrics = FairnessMetric(data,dataset_unprivileged_group,
    dataset_privileged_group,ground_truth_label_name,
    predicted_label_name,dataset_positive_outcome)
print(metrics.statistical_parity_difference())
if abs(metrics.statistical_parity_difference()) > (threshold +
    tolerance_value):
     print("Biased")
else:
     print("Fair")
```

LISTING 6.9: Generated code for the *University* use case.

Listing 6.9 reports the code generated by MODNESS for the *University* use case. All the metrics and operations defined in the metamodel (i.e., all the metaclasses extending the `Function` metaclass in Fig. 5.4) have been implemented as functions in the `FairnessMetric` Python class, which is imported at the beginning of the script.

In addition, the `pandas` Python library is imported to read and process the dataset. Next, lines from 4 to 15 define a set of variables implementing attributes specified in the model during the *fairness analysis* definition phase (i.e., file path, predicted label name, privileged and unprivileged groups, positive outcome, threshold, and tolerance value). Finally, the `FairnessMetric` class is instantiated on line 18. As said above, this class provides both implementations of existing fairness metrics and operations to create new ones. Since, in this use case, we are using a metric already defined among the possible values of the `GroupFairnessMetric` enumeration (i.e., *statistical parity*), the generated code calls a method from the `FairnessMetric` class implementing it (line 20). Finally, lines from 21 to 24 check if the value returned by the metric is within the defined threshold. If so, a `"Fair"` message is printed, `"Biased"` otherwise.

```
1  from FairnessMetric import FairnessMetric, binarize
2  import pandas as pd
3  from operators import SingleOperator
4
5  # INPUT DATA
6  file_path = "data/popbias.csv"
7  predicted_label_name = "ranking"
8  data = pd.read_csv(file_path)
9
10 # PREPROCESSING
11 operator_value = 0.8
12 operator = SingleOperator.GREATER_EQUAL
13 binarize(data, "frequency", operator, operator_value, True)
14 operator_value = 0.8
15 operator = SingleOperator.GREATER_EQUAL
16 binarize(data, "ranking", operator, operator_value, True)
17
18 # PARAMETERS
19 dataset_unprivileged_group = {"frequency": 0}
20 dataset_privileged_group = {"frequency": 1}
21 dataset_positive_outcome = 1
22 threshold = 1.0
23 tolerance_value = 0.2
24
25 # FAIRNESS ASSESSMENT
26 metrics = FairnessMetric(
27     data,
28     dataset_unprivileged_group,
29     dataset_privileged_group,
30     ground_truth_label_name,
31     predicted_label_name,
32     dataset_positive_outcome,
33 )
34 coverage = metrics.group_size("frequency == 0 and ranking == 1") /
       metrics.group_size("ranking == 1")
35 print(coverage)
36 if abs(coverage) < threshold + tolerance_value:
37     print("Biased")
38 else:
39     print("Fair")
```

LISTING 6.10: Generated code for the *TPL* use case.

Listing 6.10 reports instead the generated code for the *TPL* use case. The code follows the same structure of Listing 6.9, with some additional changes. The first difference is shown on lines from 12 to 17. In particular, differently from the *University* use case where all the values were binary, here both *frequency* and *ranking*

columns contain continuous values. Hence, based on the definition of *positive label* and *sensitive groups* specified in the model, those lines of code map values greater or equal than 0.8 to 1 and values lower than 0.8 to 0. This binary mapping is needed because all the fairness metrics available are defined on binary data [19], [194]. The second main difference is about the adopted metric. As stated in Section 5.3.3, in this use case, we use a custom fairness metric named `coverage`. Since this is a custom metric, differently from the *University* use case, it is not directly defined as a function in the `FairnessMetric` class. Instead, line 35 shows how this metric is implemented in the code. In particular, it is implemented as the ratio between the values returned by two calls of the `group_size` function. The input of the first function (i.e., the numerator) is a string selecting libraries with `frequency` equal to 0 (i.e., unpopular libraries) and `ranking` equal to 1 (i.e., recommended libraries). The input of the function in the denominator is instead a string selecting only recommended libraries (i.e., `ranking` equal to 1).

```
1  from aif360.metrics import ClassificationMetric
2  from aif360.datasets import BinaryLabelDataset
3  import pandas as pd
4  import math
5
6  class FairnessMetric(ClassificationMetric):
7      def __init__(
8              self,
9              df: pd.DataFrame,
10             unprivileged_groups: dict,
11             privileged_group: dict,
12             true_label_name: str,
13             predicted_label_name: str,
14             positive_value: int
15         ):
16             ...
17             super().__init__(
18                 self.dataset_true,
19                 self.dataset_pred,
20                 unprivileged_groups=[unprivileged_groups],
21                 privileged_groups=[privileged_group])
22
23      def probability(
24             self,
25             object: str,
26             condition: str = ""
27     ) -> float:
28             probability = self.df.query(object).shape[0] / self.df.
                   shape[0]
29             if condition == "":
30                 return probability
31             else:
32                 return (
33                     self.df.query(condition + " and " + object).
                           shape[0]
34                     / self.df.shape[0]
35                 ) / probability
36     ...
```

LISTING 6.11: Portion of the `FairnessMetric` class.

Finally, Listing 6.11 reports a portion of the `FairnessMetric` class implementation. As shown in line 6, this class extends the `ClassificationMetric` class from the `aif360` library. Thus, it inherits all the standard fairness metric implementations (i.e.,

the ones defined in the `GroupFairnessMetric` and `IndividualFairnessMetric` enumerations reported in Figure 5.4) from this library. In addition, this class provides implementations of the functions reported in Figure 5.4 to define custom metrics. For instance, in Listing 6.11 it is reported the implementation of the `probability` function.

6.2.3 Evaluation

In this section, we present the evaluation of MODNESS by referring to the following research questions (RQs):

RQ_1 How do state-of-the-art toolkits allow users to explicitly specify customized fairness definitions?

RQ_2 Can MODNESS support the whole fairness assessment process, including the automated phase?

RQ_3 Is MODNESS able to overcome the limitations of current MDE-based approaches for fairness assessment?

In particular, we rigorously assess the *expressiveness* and *correctness* of MODNESS, demonstrating how it effectively addresses the limitations of contemporary state-of-the-art methods in assessing fairness.

In the following, we first present the set of use cases examined to answer the RQs. Then, we address and answer each RQ.

Examined use cases

By carefully analyzing the approaches described in Table 3.2, we have identified pertinent use cases utilized across the literature to evaluate fairness within various domains. Given the vast array of potential scenarios, it is impractical to examine all of them within this section. Therefore, we focus on widely embraced case studies, specifically those covered by at least three distinct approaches. Moreover, to emphasize the expressiveness of MODNESS, we introduce two additional case studies into our comparative analysis. These case studies pertaining to the evaluation of popularity bias in recommender systems encompass a curated dataset of third-party Java libraries and a curated dataset of Arduino hardware and software components, respectively. Table 6.3 reports a short description of these use cases as well as references to the approaches that have addressed them. Note how two of these use cases have been used as running examples throughout this paper(i.e., *University* and *TPL*) and are highlighted in the table. In the following, we provide detailed descriptions of the other use cases and their associated datasets.

It is important to notice how, in this evaluation, we follow the related literature to identify the *sensitive group(s)* and the *positive outcome*. In a normal use case, the sensitive groups are instead identified based on the outcome of the fairness assessment, starting from a set of possible sensitive variables (e.g., variables that are protected by regulations [82]) and an outcome considered *positive* for the specific use case.

➤ **ProPublica Recidivism (COMPAS) [153] -** The *Correctional Offender Management Profiling for Alternative Sanctions* (COMPAS) was an ML system used by judges in the US to predict if a condemned person would have been a repeating offender in the two years after their release. An investigation of this software showed that this system had a bias against *non-white women*. In this case, the sensitive variables are

TABLE 6.3: The examined use cases. Use cases adopted as running examples in the paper are highlighted in bold.

Name	Domain	Sensitive attribute(s)	Positive outcome	Existing approaches
COMPAS	Social	gender, race	Non-recidiv	[64],[104],[109], [102], [34], [105], [112], [35], [110], [98]
ADULT	Social	race, gender	Income > $50.000	[97], [76], [106], [100], [104], [109] , [34], [105], [102], [112], [35],[110],[98], [96],[100],[95]
GERMAN	Financial	gender	Credit granted	[97],[100],[106], [104],[109], [102], [102],[112], [34], [105], [35], [110], [34], [98],[96], [95]
BANK	Financial	age	Client subscribed	[106], [104],[109], [34], [105], [102], [97], [110]
RESYDUO	IoT	view, respect	Item recommended	[196]
UNIVERSITY	**Education**	**gender**	**Positive admission**	[74]
TPL	**RSSE**	**frequency**	**Library recommended**	[75]

race and *gender*, the favourable outcome is *non-recidiv*, and the unprivileged group is *non-white men*.

➤ **Adult Census Income (ADULT) [152] -** This use case is about predicting whether a person's income is above $50,000 a year based on their personal information. This system was biased against *non-white women*. Hence, the sensitive variables are *gender* and *race*, the positive outcome is *income higher than 50,000$ a year*, and the unprivileged group is *non-white women*.

➤ **German Credit (GERMAN) [154] -** This use case is about the adoption by a German bank of an ML system to predict the granting of credit. This system has been proven to be biased against women, i.e., women have a lower probability of getting credit from the bank. In this case, the sensitive variable is *gender*, and the unprivileged group is *women*. The positive outcome is *having a credit granted*.

➤ **Bank Marketing (BANK) [195] -** The Bank Marketing system was an ML system developed for a phone call company to predict whether the client would subscribe to a term deposit. This system was shown to be biased against people more than 25 years old. Hence, in this case, the sensitive variable is *age*, and the unprivileged group is *people with more than 25 years*. The positive outcome is *will subscribe*.

➤ **Resyduo dataset (RESYDUO) [196] -** This dataset comprises all the Arduino projects collected from the ProjectHub[10] open-source repository. In particular, it includes 5,547 projects, 3,137 tags, 11,645 hardware components, and 1,802 libraries. It is worth mentioning that this data has been used to feed a collaborative filtering-based recommender system supporting Arduino project development [196]. Similar to the TPL dataset, the scope of the fairness analysis is to measure how popularity impacts the recommended items by considering two different sensitive variables for each project, i.e., `views` and `respects` (i.e., the number of appreciations from users). The former quantifies project popularity based on the number of users who view it. The latter represents explicit feedback on project quality. Consequently, fairness assessment can be conducted on two distinct disadvantaged groups, i.e., *low viewed* and *low respected* projects. Hence, in this use case, the sensitive variables are *views* and *respect*, and the positive outcome is *recommendation*. The unprivileged groups are items with *low views* and *low respect*, respectively.

RQ_1: State of the Art

To answer RQ_1, we evaluate the existing approaches in terms of the elicited features introduced in Section 3.1 (i.e., **F1 - Bias definition**, **F2 - Abstract bias definition**, **F3 -**

[10] `https://projecthub.arduino.cc/`

Custom metric definition, **F4 - Metric composition**, **F5 - Automated fairness assessment**, **F6 - Tool availability**). Furthermore, we discuss MODNESS by highlighting the contribution compared to the examined works shown in Table 3.1.

F1 support - Based on our investigations, it is evident that only five of the analyzed approaches can address both individual and group fairness. These approaches are Fair-SMOTE, FairKit-learn, Astraea, FairML, and MANILA. In contrast, the other tools are designed to focus exclusively on either individual or group fairness. Notably, MODNESS stands out by offering modeling constructs that comprehensively cover both individual and group bias definitions. This flexibility is achieved by drawing upon essential concepts extracted from surveys and empirical studies [19], [31], [197].

F2 support - While tools like Fair-SMOTE, Fairkit-learn, FairML, or MANILA provide methods to assess several canonical individual and group bias definitions automatically, only ASTRAEA offers a dedicated grammar that allows the user to define any biases besides the traditional ones. MODNESS goes a step forward compared to ASTRAEA by providing a tailored metamodel conceived to define fairness at two different levels of abstractions, i.e., domain-level and dataset-level (e.g., refer to the evaluation of the RESYDUO use case in Section 6.2.3, where, starting from the same domain, two different fairness analyses are depicted selecting two different sensitive attributes of the dataset).

F3 support - It is worth noting that only two approaches within our analysis allow for the customization of metric definitions: Themis and Fairway. MODNESS provides the `Function` metaclass (see Section 5.3) to specify a dedicated operation on sensitive variables defined in the specification phase. Additionally, the metamodel incorporates a selection of significant metrics that have already been established in the literature, serving as valuable tools for assessing fairness in various contexts beyond the original ones. In essence, MODNESS can be used to evaluate the statistical parity of two sensitive groups that are not strictly bounded by the social domain. Furthermore, it empowers users to define novel bias metrics (e.g., *coverage* [84]) necessary for emerging domains, such as recommender systems, thereby adapting to evolving research needs and applications.

F4 support - Among the examined strategies, only MANILA supports this feature by modeling each metric as a feature and allowing their compositions by means of aggregation functions. Meanwhile, MODNESS relies on the metamodel to compose the defined metrics, thus pursuing the generalizability of the whole process in terms of entities and their combinations.

F5 support - Although all the investigated tools offer automatic fairness assessment, these approaches are generally confined to established use cases and state-of-the-art metrics. In contrast, MODNESS is extendible and very flexible since it allows the conceptualization of fairness in domains known in the literature and domains not yet covered (i.e., recommender systems and novel use cases). Furthermore, it allows for the creation of novel fairness metrics without sacrificing automation, making it a powerful and adaptable tool for addressing the evolving landscape of fairness assessment.

F6 support - Regarding the tool availability, it is important to note that all the scrutinized approaches offer a replication package, with the exceptions being TILE, AITEST, and ASTRAEA. The majority of these approaches utilize Python libraries and frameworks, primarily due to the fact that the tested models are machine learning-based.

Furthermore, both FairML and MANILA go a step further by offering code generation functionalities that automate the deployment and testing of the system as defined during the setup phase. MODNESS adopts a comparable approach by utilizing dedicated Acceleo templates, which are fed with models adhering to the specified metamodel.

Answer to RQ_1: Although offering a good degree of automation, existing approaches lack in supporting the customization of bias and fairness definitions. MODNESS fills this gap by covering all the elicited features for bias definition, fairness analysis specification, analysis implementation, and fairness assessment.

RQ_2: Use Case Coverage

To address RQ_2, we have selected five distinct use cases from those previously discussed in Section 6.2.3, encompassing various application domains, including social, financial, education, recommender systems in software engineering (RSSE), and the Internet of Things (IoT). The details of the five use cases, implemented using MODNESS, are provided in Table 8.6. Specifically, for each use case, we specify the chosen metric for assessment, the number of sensitive variables considered, and the MODNESS outcome.

Note how two of these use cases (i.e., *University* and *TPL*) have already been implemented throughout the paper to show the main capabilities of MODNESS and are not reported in this Section.

TABLE 6.4: MODNESS implementation of the use cases. Use cases adopted as examples throughout the paper are highlighted in bold.

Use case	Domain	Metric	Number of sensitive vars	Outcome (Expected)	MODNESS Assessment result
COMPAS	Social	AO	2 (`sex`,`race`)	0.3 ($\leq$ |0.2|)	**Biased**
GERMAN (BIASED)	Financial	EO	1 (`sex`)	-0.25 ($\leq$ |0.2|)	**Biased**
GERMAN (DEBIASED)				-0.05 ($\leq$ |0.2|)	*Fair*
RESYDUO	IoT	GEI	1 (`views`	0.31 ($\geq$ |0.8|)	**Biased**
			1 (`respects`)	0.28 ($\geq$ |0.8|)	**Biased**
UNIVERSITY	Education	SP	1 (`sex`)	-0.15 ($\leq$ |0.2|)	*Fair*
TPL	RSSE	COV	1 (`frequency`)	0.29 ($=$ |1.2|)	**Biased**

Moreover, for the GERMAN use case, we conduct two separate analyses: the first utilizing the original biased dataset and the second involving the same dataset after applying a preprocessing algorithm designed to mitigate bias (specifically, the Debiaser for Multiple Variables algorithm presented in Chapter 4). This analysis scenario exemplifies a typical scenario for MODNESS, where a user initially assesses the fairness of the original dataset and subsequently verifies if the bias has been reduced after employing a debiasing method.[11] Finally, for the RESYDUO use case, we perform two distinct analyses, one considering the *views* sensitive variable and the other focusing on the *respects* sensitive variable. This approach showcases MODNESS's

[11]It is important to clarify that the mitigation of bias is beyond the scope of this approach, as MODNESS primarily focuses on designing and implementing the fairness assessment workflow, as outlined in

versatility and ability to handle different sensitive variables, further highlighting its capabilities.

The implemented models and generated code for each use case are reported in our replication package [49].

```
GroupBias "compas"{
    definition: {
        domain: "justice";
        source:HUMAN_DISCRIMINATION;
        sensitiveVariables: {
            SensitiveVariable{
                name: "gender";
                values: "male","female";
            },
            SensitiveVariable{
                name: "race";
                values: "white","non-white";
            }
        };
        positiveOutcome: "Non Recidiv";
        unprivilegedGroup: {
            SensitiveGroup{
                name: "non-white men";
                sensitiveValue: "race.non-white",
                                "gender.male";
            };
        };
        privilegedGroup: {
            SensitiveGroup{
                name: "white women";
                sensitiveValue: "race.white",
                                "gender.female";
            };
        };
  };
```

LISTING 6.12: Bias definition for the COMPAS use case.

```
Dataset {
  id: 'compas';
  groundTruthLabelName: 'two-year-recid';
  predictedLabelName: 'prediction';
  filePath: 'compas.csv';
  positiveOutcome: {
    id: "non-recidiv";
    mappingOutcome: "Non Recidiv";
    value: { operator: EQUAL; value: 0.0; };
  };
  datasetSensitiveVariable: {
    DatasetSensitiveVariable{
      name: "sex";
      mappingSensitiveVariable: gender;
      values: {
        SensitiveVariableValue{
          id: "female";
          mappingValue: "gender.female";
          value: { operator: EQUAL; value: 0.0; };
        },
        SensitiveVariableValue{
          id: "male";
          mappingValue: "gender.male";
          value: { operator: EQUAL; value: 1.0; };
        }
      }
    },
    DatasetSensitiveVariable{
      name: "race";
      mappingSensitiveVariable: race;
      values: {
        SensitiveVariableValue{
          id: "white";
```

```
34            mappingValue: "race.white";
35            value: { operator: EQUAL; value: 1.0; };
36          },
37          SensitiveVariableValue{
38            id: "non-white";
39            mappingValue: "race.non-white";
40            value:{ operator: EQUAL; value: 0.0; };
41          }
42        }
43      }
44    };
45  };
46  datasetUnprivilegedGroup: {
47      id: "non-white-men";
48      mappingGroup: "non-white men";
49      sensitiveVariables: ("compas.sex.female",
50      "compas.sex.male");
51  };
52  datasetPrivilegedGroup: {
53    id: "white-women";
54    mappingGroup: "white women";
55    sensitiveVariables: ("compas.sex.male",
56     "compas.race.white");
57  };
```

LISTING 6.13: Excerpt analysis definition for the COMPAS use case.

The COMPAS use case. Concerning the Social domain, we replicated the COMPAS use case. Listings 6.12 and 6.13 shows the bias definition and an excerpt of the fairness analysis definition, highlighting how multiple sensitive variables and intersectional sensitive groups (i.e., sensitive groups identified by more than one sensitive variable [80]) can be defined in MODNESS. As previously described, this use case is about discrimination of *non-white men* in the prediction of *recidivism*. Hence, we modeled the bias definition in MODNESS, specifying *gender* and *race* as sensitive variables, *non-recidiv* as the positive outcome, *non-white men* as the unprivileged group, and *white women* as the privileged group (see Listing 6.12).

Next, we defined our fairness analysis by specifying the dataset containing all the related information (see Listing 6.13). In particular, we specified in the attributes of the `Dataset` class that the ground truth labels are encoded in the `two_years_recid` column. In contrast, the model predictions are encoded in the `prediction` column. Then, we modeled that the positive outcome equals `1.0`. Finally, we specified that the sensitive variables are encoded in the `race` and `sex`[12] columns where *non-white women* have a value of `1` for both columns. After defining the dataset, we specified the metrics for analysis. For this use case, we adopted the *Average Odds (AO)* fairness metric, which is included in the `ExistingFairnessMetric` class. Finally, we specified that this metric should be equal to 0 to have fairness, with a tolerance value of 0.2.

From such a model, MODNESS generates the code implementing the analysis. The code follows the same structure of Listing 6.9 and is reported in our replication package.

The GERMAN use case. Concerning the Financial domain, we implemented the GERMAN use case, which is about the discrimination of women in credit granting. Similarly to the COMPAS use case, we first specified in the bias definition the sensitive variable (*gender*), the positive outcome (*credit grant*) and the privileged (*men*) and unprivileged (*women*) groups. Next, we specified two different fairness analyses, one involving the original biased dataset and another involving the debiased

[12]We refer to the original column names of the dataset reported in [153]

one. In both analyses, we modeled that the `sex` column encodes the *gender* sensitive variable where *women* have a value equal to `1`. In contrast, the positive outcome is encoded in the `credit` column with a value equal to `1`. In both analyses, we selected the *Equal Opportunity (EO)* fairness definitions, specifying a threshold of 0 and a tolerance value of 0.2. The generated code follows the same structure of Listing 6.9 and is reported in the replication package as well as the MODNESS implementation for this use case.

The RESYDUO use case. Finally, for the IoT domain, we implemented the RESYDUO use case, which is about popularity bias in recommending software and hardware Arduino components [196]. In the bias definition, we specified *views* and *respect* as sensitive variables and *high ranking* as the positive outcome. Next, we defined two sensitive groups: one identified by the *views* sensitive variable (i.e., the privileged group is *high-viewed* items, while the unprivileged group is *low-viewed* items), and one identified by the *respect* sensitive variable (i.e., the privileged group is *high-respected* items, while the unprivileged group is *low-respected* items). Further, we defined two different fairness analyses. The first one aims at assessing the amount of popularity bias with respect to the number of *views*, while the second aims at assessing the popularity bias with respect to the level of *respects*. In both analyses, we specified that the predicted rank is encoded in the `tot_recommendations` column and that we consider a rank *high* if it is greater than the 80% of the predicted ranks (like done for the TPL use case). Next, in the first analysis, we specified that the number of views is encoded in the `views` column and that an item is *highly viewed* if its number of views is higher than 80% of the other items. Instead, in the second analysis, we specified that the level of respect is encoded in the `respects` column and that an item is *highly respects* if it has a respect level higher than 80% of the other items. In both analyses, we modeled a custom metric used in the RecSys literature named *Generalized Cross Entropy (GEI)*[198]. This metric measures how the probability distribution of having an item of the privileged group recommended is different from the probability of having an item of the unprivileged group recommended. Following the metric definition, we specified that this metric should be equal to or greater than 0.8 to have fairness. As for the other cases, the generated code follows the same structure of Listing 6.9 and is reported in our replication package as well as the MODNESS DSL implementation.

Altogether, the performed fairness assessment confirms that the bias is correctly detected in all the considered use cases, meaning that MODNESS is capable of detecting the biases defined at the model level.

> **Answer to RQ_2:** MODNESS has a level of *expressiveness* and *correctness* able to model and successfully evaluate use cases from various domains, including social, financial, RSSE, and IoT. Our experiments demonstrate the extensive range of MODNESS's ability to define bias and fairness in different domains and its capability to automatically generate the relative experiments and hence assess fairness in the considered use cases.

RQ_3: Baselines Comparison

To address RQ_3, we conducted a comparative analysis between MODNESS and two MDE-based baselines for fairness assessment, namely FairML [112] and the MANILA framework presented in Section 6.1. Assessing the quality of MDE-based

tools is daunting since they usually rely on tailored metamodels conceived for a specific application domain. Prior works have defined a set of quality metrics that investigate several aspects, such as expressiveness, completeness, or portability. Within the scope of our paper, we follow the criteria proposed in [199] to establish two dimensions for facilitating comparison: *expressiveness* and *automation*. We frame these aspects by referencing the set of features detailed in Section 3.1.2

- **Expressiveness:** This dimension measures the extent to which the tool enables the modelling of bias definitions and relative fairness analysis (encompassing features **F2-F3-F4**).
- **Automation:** This dimension evaluates the degree to which the tool streamlines the entire fairness assessment process (encompassing features **F5-F6** plus an additional feature describing the level of guidance for the user provided by the tool in the fairness analysis specification).

These dimensions are assessed on a scale ranging from 1 to 3, based on the number of features provided by the tools for both **expressiveness** (i.e., F2, F3 and F4) and **automation** (i.e., F5, F6 and guidance in the specification).

We model with the two baselines the use cases used to answer RQ_2, and we use them to evaluate these features. The comparison results, focusing on **Expressiveness** and **Automation**, are presented in Table 6.5.

TABLE 6.5: Baseline comparison. For each baseline, we evaluate the expressiveness and automation scores based on the features they provide.

	Expressiveness				Automation			
	Abstract bias def.	Custom metric def.	Metric comp.	Expr. Score	Tool available	Code generation	Spec. guidance	Automation Score
FairML	✗	✗	✗	0	✔	✔	✔	3
MANILA	✗	✗	✔	1	✔	✔	✔	3
MODNESS	✔	✔	✔	3	✔	✔	✗	2

Expressiveness pertains to the extent to which the tools offer abstraction capabilities to model a variety of heterogeneous use cases. To assess this aspect, we implemented each of the use cases detailed in Section 6.2.3 with every baseline tool and assessed their ability to define high-level custom bias definitions and custom metric definitions and to compose different existing metrics. In the following, we describe each tool in detail and explain how they provide or do not provide these features.

Similarly to MODNESS, FairML relies on an MDE-based infrastructure to define fairness assessments using a dedicated DSL. In particular, the tool provides abstractions to specify standard metrics from the AIF360 library. However, the tool does not provide abstractions to define high-level custom bias definitions and custom metrics or to compose existing ones. Hence, FairML got `0` as score for `Expressiveness`.

As described in Section 6.1, MANILA relies on the ExtFM formalism to model a fairness evaluation workflow as an SPL. However, it does not provide features to compose or define custom metrics. Moreover, the tool does not provide features to define high-level custom bias definitions. Hence, neither custom bias definition nor custom metric features are supported. Instead, the tool provides a set of aggregation functions to combine different metrics, providing the metric composition feature. Hence, MANILA got `1` as score for *Expressiveness*.

MODNESS instead has been developed to address the limitations of current baselines in defining and executing custom bias assessments. Hence, it provides

abstractions to define high-level custom bias definitions and custom metrics and to compose existing ones. Moreover, like the two baselines, it provides abstractions to use existing metrics from the AIF360 library. Hence, MODNESS got 3 as expressiveness score.

Regarding the degree of **automation**, both FairML and MANILA offer user guidance when defining fairness assessments. FairML, for instance, employs a decision tree to assist users in selecting the appropriate metric based on the analysis scope they intend to pursue. On the other hand, as described in Section 6.1, MANILA employs ExtFM constraints to guide users in selecting a set of features that invariably results in a correct (i.e., executable) experiment. As of the current development stage, MODNESS does not provide this level of user guidance. It is important to note that we plan to integrate MODNESS capabilities with MANILA to guide users through the fair development process while providing extensive expressiveness capabilities (see Section 5.4). Consequently, we assigned a score of 3/3 for the automation level of FairML and MANILA, while MODNESS received a score of 2/3 for its current automation capabilities.

TABLE 6.6: List of implemented use cases and assessment result.

	FairML	**MANILA**	**MODNESS**
COMPAS	0.28	0.29	0.3
GERMAN	-0.2	-0.23	-0.25
GERMAN FAIR	-0.05	-0.1	-0.05
UNIVERSITY	-0.15	-0.12	-0.15
TPL	✗	✗	0.29
RESYDUO	✗	✗	0.31 (`views`) 0.28 (`respect`)

Finally, the list of assessment results for each use case implemented is presented in Table 6.6. As can be seen, all the tools report comparable results for all the involved use cases. The small variability among the results can be explained by the different training-testing splits.

From this analysis, we have seen how all the selected baselines provide a high level of *automation* in the definition and implementation of a fairness evaluation. However, both baselines do not have a level of *expressiveness* fairness analyses definition in terms of domains (e.g., RSSE or IoT) and metrics (e.g., *coverage* or *GEI*).

Answer to RQ$_3$: While all the examined baseline tools exhibit a high degree of automation throughout the fairness assessment process, both of them share a common limitation, i.e., they lack the ability to express fairness in different domains. In contrast, MODNESS overcomes these limitations by offering a versatile framework for modeling high-level bias definitions and specifying and implementing custom fairness metrics tailored to specific application domains.

6.2.4 Threats to Validity

This section discusses possible threats that can hamper the results of the performed evaluation.

Internal validity Concerning the proposed approach, it is possible that the metamodel we have created and the supporting tools are not extensive enough to cover all fairness assessment scenarios. However, we purposely considered different application domains, including the one related to the popularity bias of recommender systems in software engineering. Another potential threat of our study concerns the macro-sources of bias we cover. In particular, we address *algorithmic bias* and *unbalanced group bias* [19], [43], despite various other macro-sources of bias have been identified over the years, such as *confounding variables bias* [43]. However, we believe that our approach covers most of the bias case studies documented in the literature, as they originate from macro sources of bias that we address. In addition, our proposed metamodel is also designed to be extendable to model sources of bias that are not currently addressed. We acknowledge that the code generation may not be accurate due to some errors while running the Acceleo transformation. To mitigate this, we linked the metamodel to the project programmatically, to avoid any possible issue in running the code generation.

External validity In this respect, the results obtained in this paper may be valid only for the considered datasets. To mitigate this threat, we diversified the datasets, which have been collected from different sources and domains. Furthermore, we demonstrated that the proposed approach could cover fairness conceptualization and assessment also in the software engineering domain by considering popularity bias in recommender systems. Another threat that may hamper the obtained results is the choice of the baselines, i.e., we cannot conduct a quantitative comparison in terms of metrics. To mitigate this, we conduct a qualitative analysis by reimplementing the examined use cases using the selected approaches.

6.3 Conclusion

In this chapter, we introduced two low-code approaches designed to facilitate the development of fair learning-based systems. The first approach is MANILA, a web-based application based on the ExtFM for modeling fairness benchmarking workflow. We demonstrated how MANILA offers a degree of *expressiveness* and *correctness* that enables the reproduction of the DEMV experimental evaluation detailed in Chapter 4. However, despite its high expressiveness, MANILA does not support modeling use cases, such as the *TPL* example presented in Chapter 2, where custom fairness metrics are utilized. To mitigate this threat, we proposed MODNESS, a model-driven framework designed to conceptualize, design, implement, and execute custom fairness assessment workflows. We demonstrated how MODNESS provides a degree of *expressiveness* that overcomes existing model-driven baselines, including MANILA. However, it provides a lower degree of *automation*.

The evaluations described demonstrate how these approaches complement one another. By relying on the ExtFM formalism, MANILA guides data scientists in establishing complete and accurate fairness development workflows. In contrast, MODNESS, which is based on the MDE formalism, offers a significant level of expressiveness for defining and conducting fairness evaluations. In the future, we plan to integrate these two approaches to create a system that assists data scientists and domain experts in defining and executing fairness development workflows, even for more unconventional use cases.

Chapter 7

Towards Early Detection of Algorithmic Bias from Dataset Bias Symptoms

In Chapters 5 and 6, we presented the modeling and the low-code implementations of the two standard workflows for fairness assessment and to identify the best ML model and fairness-enhancing method combination presented in Chapter 2. As highlighted in Chapter 2, those workflows are nowadays standards for the development of fair learning-based systems, and a plethora of approaches have been proposed following them, like [34]–[36] to mention a few. However, one limitation of all those approaches (including MANILA and MODNESS) is that they require the predictions of the underlying ML model as input to assess its bias. Hence, they can only be performed after the *model training* or *model deployment* phases, which are late steps of the learning-based systems development workflow shown in Figure 1.1.

For this reason, research interest has recently started to grow towards *early bias detection* - i.e., having a glimpse of algorithmic bias at earlier stages of the learning-based system development workflow. Shome *et al.* were the first authors to analyze the relationship existing between Data Fairness Metric (DFM), i.e., metrics to assess the bias in the training data, and their corresponding adaptation to the model's predictions (Model Fairness Metric – MFM) [37]. However, their work focuses on a limited set of metrics that do not cover all the possible fairness definitions available in literature nowadays [19], [31].

In this chapter, we perform a further step by analyzing how the dataset's structural features, namely *bias symptoms*, can be employed for the early detection and explanation of algorithmic bias, i.e., bias inducted by the ML model. We extract those symptoms using binary variables from 24 datasets well-known in the fairness literature. To the best of our knowledge, this represents the largest number of datasets utilized in a fairness study focusing on tabular data. Next, we use those symptoms to detect early signals of bias under three different bias definitions: *Statistical Parity (SP)*, *Equal Opportunity (EOp)*, and *Average (Equality) Odds (EO)*.

The rationale for analyzing bias symptoms is manifold. First, we aim to assess to what extent bias symptoms can be employed to detect signals of bias in earlier steps of a learning-based system development pipeline, allowing the early identification of variables that could lead to *high* bias under a given definition. For instance, bias symptoms could be employed in MANILA and MODNESS to suggest which dataset features possibly lead to high bias in the system.

Secondly, symptoms of bias can be used to explain why a particular variable might cause a specific type of bias in a system. This approach could be applied

in MANILA and MODNESS to provide a clearer explanation of the results from a fairness analysis.

With this analysis, we aim to make practitioners and researchers aware of the possibilities and challenges of adopting bias symptoms.

Referring to the challenges and contributions described in Chapter 1, the study presented in this chapter constitutes the contribution CN_3 proposed to address the challenge CH_3.

The rest of this chapter is structured as follows: Section 7.1 presents the Research Question that driven our research; Section 7.2 describes in detail the methodology that has been followed to collect the dataset of bias symptoms; Section 7.3 describes the conducted empirical evaluation, while Section 7.4 provides the results. Section 7.5 discusses the main takeaways derived from our empirical evaluation. Section 7.6 reports possible threats to our work. Finally, Section 7.7 provides future work directions and concludes this chapter.

7.1 Research Questions

This research is driven by the following research questions (RQ):

RQ_1: *Which relations exist between bias symptoms and Statistical Parity, Equal Opportunity, and Average Odds definitions of bias?*

This RQ aims to analyze the correlation existing between symptoms and bias metrics. In particular, we compute the non-parametric *Spearman* correlation coefficient [200] and observe how some fairness metrics are directly correlated with specific symptoms. This allows those symptoms to be used as a proxy for early bias detection or to explain why a variable may lead to *high* bias under a given definition.

RQ_2: *To what extent do the identified bias symptoms allow the early detection of bias under the three examined metrics?*

In this RQ, we aim to evaluate the effectiveness of the identified symptoms in reflecting bias according to the definitions of SP, EOp, and EO. To accomplish this, we first train three widely used classification methods: Multi-Layer Perceptron (MLP), Random Forest (RF), and XGBoost, using the selected bias symptoms. Next, we evaluate whether these classification methods can accurately predict if a specific variable may lead to *high* or *low* bias in a Logistic Regression base classifier given a set of symptoms extracted from that variable.

RQ_3: *Which symptoms are the most helpful in early prediction of Statistical Parity, Equal Opportunity, and Average Odds?*

This RQ integrates the results obtained in RQ_1 by identifying which features are perceived as the most relevant by the employed classifiers for early bias prediction. To achieve this, we compute the importance of each feature in the classification task performed to answer RQ_2 by using the model-agnostic *permutation importance* algorithm [201].

RQ_4: *Do different base classifiers influence the ability of bias symptoms in representing high and low values of the examined bias metrics?*

This RQ extends RQ_2 by assessing how the employed symptoms are effective in detecting variables that could possibly lead to *high* or *low* bias in base classifiers different from Logistic Regression. In particular, we consider MLP and

RF as base classifiers and show how the effectiveness of classification methods trained on bias symptoms is consistent with the results shown in RQ_2.

7.2 Methodology

In this section, we describe the methodology we followed to identify the bias symptoms used for early bias identification. First, we describe the bias metrics considered in this work. Second, we describe the identified symptoms and explain the rationale behind their selection. Third, we define the process of collecting those symptoms and experimenting with them by creating a dataset of bias symptoms starting from 24 datasets from the fairness literature.

7.2.1 Selected fairness metrics and relative thresholds

In this study, we focus on *Statistical Parity Difference (SP)*, *Equal Opportunity Difference (EO)*, and *Average Odds (AO)* fairness metrics. As discussed in Chapter 2, these metrics implement the fairness definitions most adopted in the literature [43], [80], [103]. Additionally, these metrics fall into two distinct categories of fairness definitions [78]: SP is classified under the *independence* category, while both EO and AO are categorized under the *separation* category.

Building on the DEMV analysis discussed in Chapter 4, we consider absolute values for all these metrics. In this context, a value of zero indicates optimal fairness, while a value of one signifies complete bias.

It is important to note that all these metrics depend on specific outcomes of an ML model, and different models may produce varying outcomes. Hence, predicting the exact value of those metrics from dataset statistics without considering any aspect of an ML model could be unrealistic. For this reason, in this work, we do not focus on predicting their exact value but rather on detecting if they exceed a given threshold, which indicates that an ML model may suffer from bias. In particular, we consider the following *thresholds* to distinguish *High* and *Low* bias: 0.2 for SP (following the 80% rule [27], [82]), 0.10 for EO, and 0.15 for AO. Since there is still no agreement on which threshold to use for EO and AO [79], [191], these values have been empirically selected based on the median scores obtained in our bias symptoms dataset (see Section 7.2.4).

7.2.2 Symptoms identification

The first step in our work was the identification of characteristics for the early identification of variables leading to *High* bias with respect to the thresholds previously discussed. Such characteristics represent *bias symptoms*, which have been selected by analyzing the metrics formulations shown in Chapter 2, performing empirical analyses of the datasets employed in our study (see Section 7.2.3), and reading fairness-related works. In total, we have identified 13 different symptoms, which are summarized in Table 7.1. The symptoms are the following:

- **Data Statistical Parity (DSP):** This value is a version of SP which considers ground truth labels instead of the model's predictions (i.e., $|P(Y = 1|S = 0) - P(Y = 1|S = 1)|$) [81].
- **Unprivileged Positive Probability (UPP) and Privileged Positive Probability (PPP):** These values represent the single probabilities that compose the above symptom [81].

TABLE 7.1: Bias Symptoms Overview

Symptoms	Formulation	Legend
DSP	$\lvert P(Y=1\lvert S=0) - P(Y=1\lvert S=1)\rvert$	Y: Outcome, S: Sensitive attribute (0 or 1)
NPD	$P(Y=0\lvert S=0) - P(Y=0\lvert S=1)$	Y: Outcome, S: Sensitive attribute
UPP	$P(Y=1\lvert S=0)$	Probability of positive outcome given $S=0$
PPP	$P(Y=1\lvert S=1)$	Probability of positive outcome given $S=1$
Gini Index	$G = \frac{m}{m-1} \cdot \left(1 - \sum_{i=1}^{m} f_i^2\right)$	m: Number of classes, f_i: Frequency of class i
Simpson Diversity	$D = \frac{1}{m-1} \cdot \left(\frac{1}{\sum_{i=1}^{m} f_i^2} - 1\right)$	m: Classes, f_i: Frequency of class i
Shannon Diversity	$S = -\left(\frac{1}{\ln m}\right) \sum_{i=1}^{m} f_i \ln f_i$	m: Classes, f_i: Frequency of class i
IR	$IR = \frac{\min(\{f_{1\ldots m}\})}{\max(\{f_{1\ldots m}\})}$	$f_{1\ldots m}$: Frequencies of classes
Unpriv Group Unbal **Priv Group Unbal**	$\frac{W_{\text{obs}}}{W_{\text{exp}}}$	W_{obs}: Observed group size W_{exp}: Expected group size
Kurtosis	Kurtosis $= \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^4/\sigma^4$	n: Sample size, x_i: Data point, $\bar{x}$: Mean, σ: Std. deviation
Skewness	Skewness $= \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^3/\sigma^3$	x_i: Data point, $\bar{x}$: Mean, σ: Std. deviation
Kendall's τ	$\tau = \frac{(C-D)}{\binom{n}{2}}$	C: Concordant pairs, D: Discordant pairs, n: Observations
Mutual information	$J(Y,Z) = \frac{1}{2}\,\mathbb{E}\int (Z_s - \hat{Z}_s)^\top (Z_s - \hat{Z}_s)\, ds$	Y, Z: Random variables, $\hat{Z}_s$: Estimated Z

- **Gini Index, Simpson Diversity, Shannon Diversity, and Imbalance Ratio (IR):** These values have been used in [122] to measure the unbalance in the values of a sensitive variable. In [122] the authors show how these values are able to reflect variations in SP, EO, and AO. Following their approach, we 0-1 normalize these values such that, for Gini, Simpson, and Shannon, a value of 0 means that a variable is entirely unbalanced, while a value of 1 means that a variable is fully balanced. Concerning IR, a value of 1 means that the variable is entirely unbalanced, while a value of 0 means full balance.

- **Unprivileged Group Unbalance and Privileged Group Unbalance:** These values are used to represent the unbalance of a variable with respect to the positive value of the ground truth label. In particular, they are the ratio between the expected and observed sizes of the unprivileged and privileged groups with respect to the ground truth label ($\frac{W_{obs}}{W_{exp}}$). A value of one means that the groups are fully balanced (i.e., the ratio of items having a positive and negative label value is the same), a value > 1 means that the group is oversampled (i.e., items having a positive label are higher than expected). In contrast, a value < 1 implies that the group is undersampled (i.e., items having a positive label are lower than expected). These values have been used by previous works to develop methods able to mitigate *unbalanced groups bias* [43], [86].

- **Kurtosis and Skewness:** These values have been included to represent the distribution of a variable. They have been used by several works in the Auto ML domain, and it has been shown how they can influence the predictions of an ML model [202]–[204].

- **Kendall's τ:** This value represents the correlation between a variable and the ground truth label. We adopted Kendall's τ to measure the correlation because it is non-parametric and more robust than the other non-parametric Spearman's correlation coefficient [200]. It ranges between -1 and 1, where -1 means absolute negative correlation, 1 means absolute positive correlation, and 0 implies no correlation. Intuitively, a variable shall be highly correlated with the ground truth label to lead to *High* bias in the predictions of a model. To confirm

this intuition, we computed the Kendall τ between each binary variable and the ground truth label on 24 datasets from the fairness literature (see Section 7.2.3) and then grouped the results by variables having *High* and *Low* values of SP, EO and AO following the thresholds defined in Section 3.1.1. Figure 7.1 shows the mean and the 95% confidence interval of the Kendall τ grouped by high and low values of each bias metric: variables with high values of SP and AO are also more positively correlated with the ground truth label. We also computed the *Welch's t-test* (a non-parametric test to assess the null hypothesis that two groups with different numbers of samples have the same mean [205]), which confirmed a statistically significant difference of the means concerning SP and AO (following previous works [15], [43], we consider a statistical test significant if the p-value is < 0.05).

- **Mutual Information:** This value is a non-parametric metric that measures the mutual dependency between two random variables (continuous or discrete). It ranges between 0 and 1, where 0 means complete independence, while 1 means complete dependence [206]. Like Kendall's τ, we included this metric to represent how much dependency exists between a variable and the ground truth label. As before, we empirically compared the mean values of Mutual Information between items with high and low values of SP, EO, and AO. Figure 7.2 reports the results of our evaluation. Differently from Kendall's τ, we observe a statistically significant difference in the mean values for all the considered bias metrics. In particular, the mean Mutual Information is higher for items with higher bias values.

7.2.3 Dataset Creation

We employed 24 tabular datasets from the literature on bias and fairness [77], [119], [221], [222]. The list of the employed datasets and additional statistics are reported in Table 11.2. In particular, we selected datasets that are: publicly available, suitable for classification tasks, and contain at least one binary variable different from the label. It is worth noticing that, differently from other works on fairness [80], [98], [223], we employ both binary and multiclass datasets, i.e., where the possible values of the label are > 2.

Before starting the symptoms extraction process, all the datasets were preprocessed by removing missing values and one-hot-encoding categorical columns. After this process, each dataset has been split into training (80%) and testing (20%) sets. The train set has been used to train a Logistic Regression (LogReg) *Base Classifier* [53].

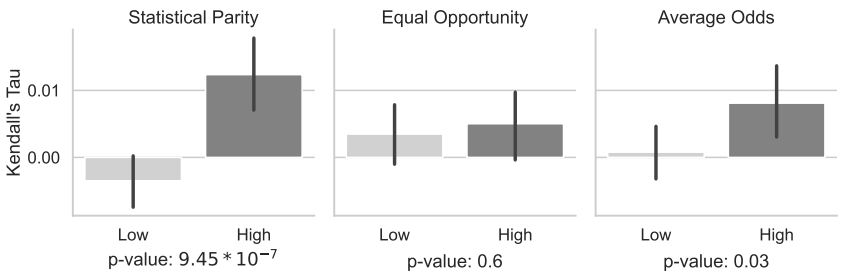


FIGURE 7.1: Mean and 95% confidence interval of Kendall τ between binary variables and ground truth labels grouped by High and Low SP, EO and AO values. For each metric, we report the Welch's t-test p-value.

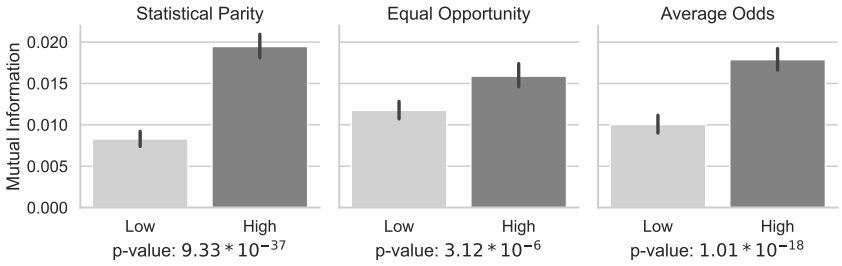


FIGURE 7.2: Mean and 95% confidence interval of Mutual Information between binary variables and ground truth labels grouped by high and low SP, EO, and AO values. For each metric, we report the Welch's t-test *p*-value.

TABLE 7.2: List of adopted datasets

Datasets	**Domain**	**Positive Label** [a]	**Label Type**	**Reported Sens. Var.**	**Binary Vars.**
Adult [207]	Social	Income > 50K (1)	Binary	sex	902
Arrhythmia [208]	Health	Normal (1)	Multiclass	gender	318
Bank [195]	Economy	Subscribed (1)	Binary	age	440
Campus Recruitment [209]	Education	Placed (1)	Binary	gender	155
CMC [155]	Health	Frequent use (2)	Multiclass	wife religion	30
COMPAS [153]	Justice	No recidivism (0)	Binary	sex	1371
Credit Card [210]	Fraud Detection	No default (1)	Binary	sex	10
Crime [156]	Justice	Low crime (100)	Multiclass	black people	21
Diabetes Hospitals [211]	Social	Readmitted (0)	Binary	gender	709
Drug [158]	Health	No drug use (0)	Multiclass	gender	20
German [154]	Economy	Good credit (1)	Binary	age	518
Heritage Health [212]	Social	Admitted (1)	Binary	gender	171
Hearth Disease [213]	Health	No disease (0)	Binary	sex	38
IBM Analytics [214]	Social	No attrition (0)	Binary	gender	290
Law School [74]	Education	Admitted (2)	Multiclass	race	50
Obesity [215]	Health	No obesity (0)	Binary	gender	60
Parkinson Monitoring [159]	Health	No Parkinson (0)	Binary	sex	20
Resyduo [196]	RecSys	Recommended (1)	Binary	views	10
Ricci [216]	Education	High Score (1)	Binary	race	47
SCG-RHC [217]	Health	No health challenge (0)	Binary	gender	61
Student Performance [218]	Education	Passed (1)	Binary	sex	309
US Census [219]	Social	High income (1)	Binary	sex	150
Vaccine [220]	Social	Vaccine trust (0)	Binary	gender	210
Wine [160]	Food	High quality (6)	Multiclass	type	20

[a] We report in brackets the corresponding value in the dataset.

Previous works on fairness have driven our selection of LogReg as a base classifier [43], [44], [80], [223]. Following the same related works, we employed the LogReg implementation from the *scikit-learn* Python library [145] with default hyperparameters.

After training the model, we use the test set to predict the label needed to compute ground truth values of SP, EO, and AO following the formulations provided in Section 7.2.1. The test set has also been used to extract the bias symptoms. Following the definition of *privileged* and *unprivileged* groups [19], we selected all binary columns from each dataset and computed the bias metrics and symptoms for any of each. For each binary variable, we assumed that 0 identifies the unprivileged group,

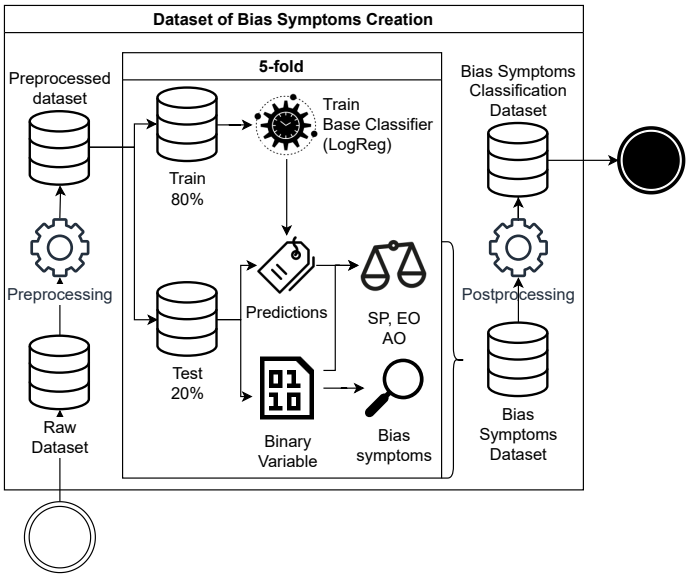


FIGURE 7.3: Dataset creation workflow

while 1 identifies the privileged group.[1] The positive label of each dataset has been derived by its relative source paper and is reported in Table 11.2. To compute SP, EO, and AO, we adopted the implementations provided in [43], which are an extension of the metrics available in the *IBM AIF360* library [27] for the multiclass classification task. It is worth noticing that, as described in Section 7.2.2, the bias symptoms are computed on the ground truth label of the testing set and not on the LogReg predictions. To increase the overall amount of collected bias symptoms, we repeat the whole train and test phase five times using a 5-fold approach (i.e., on each fold, we select a different subset of data for testing and the rest for training). The ground truth bias metrics and symptoms compose our *bias symptoms dataset*. However, since we are interested in detecting variables leading to *high* bias, we performed a postprocessing operation to map each metric value into low (i.e., 0) and high (i.e., 1) classes using the thresholds defined in Section 3.1.1. Finally, we filtered the symptoms to remove any possible redundant feature which could influence the early detection of bias. Following the work from Mastropaolo *et al.* [224], we applied the `redun` algorithm to select features not affected by multicollinearity [225]. This algorithm iteratively removes independent variables, determining how well each of them can be predicted using the remaining ones. The process continues until no variable can be predicted with high confidence using the remaining ones.[2] The algorithm detected two redundant variables (*Privileges Positive Probability* and *Kurtosis*) which have been removed from the dataset, yielding a total of 11 bias symptoms.

7.2.4 Bias symptoms dataset description

In the following, we report some statistics about the collected bias symptoms dataset. The dataset comprises 5,930 instances (i.e., one row for each binary variable of the

[1]The choice of values encoding the privileged and unprivileged groups only impacts the sign of the bias metrics. However, since we consider absolute values, this choice did not influence our results.

[2]To assess the prediction confidence, the algorithm uses the R^2 score with a threshold of 0.8

datasets reported in Table 11.2 repeated five times following the 5-fold process reported in Section 7.2.3) and 14 features (i.e., the 11 symptoms selected using the process described in Section 7.2.2 plus the three bias metrics).

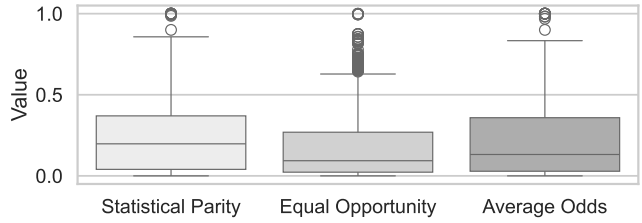


FIGURE 7.4: Median and inter-quartile range of SP, EO, and AO values in the Symptoms' Bias Dataset

The comparison of raw SP, EO, and AO values (i.e., before their 0-1 mapping) is shown in Figure 7.4. We observe how the metrics have a different distribution. In particular, SP has a higher median (0.19) compared to EO (0.09) and AO (0.12). The nature and formulation of those metrics could explain this diversity in median scores. SP measures the difference in positive predictions between privileged and unprivileged groups, disregarding the ground truth values (see Equation 2.1). On the contrary, EO measures the difference in True Positive Rates (TPR) (i.e., positive predictions of the ML model where the ground value is also positive, see Equation 2.3) while AO measures the difference in TPR and False Positive Rates (FPR) (i.e., positive predictions of the ML model where the ground value is non-positive, see Equation 2.4) between the two groups.

A higher median value for SP suggests that this metric captures biases that may not be directly linked to differences in TPRs and FPRs. For example, this could occur in situations where the distribution of positive ground truth labels in the dataset is imbalanced towards one group. In such cases, the LogReg model might accurately identify the ground truth positive labels for both groups (resulting in a lower EO value), yet the percentage of positive predictions could still be higher for one group (indicating a higher SP difference).

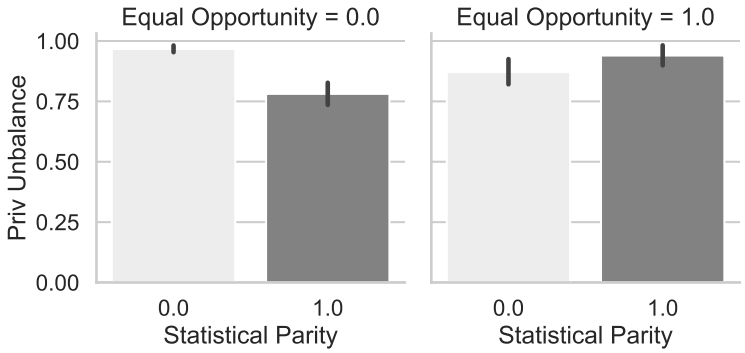


FIGURE 7.5: Distribution of *Privileged Group Unbalance* between items with high and low values of SP and EO

To investigate this difference better, we computed the distribution of the *Privileged Group Unbalance* symptom (see Section 7.2.2) for items having a *high* and *low* value of SP and EO. The distribution is reported in Figure 7.5 and shows how this symptom is generally lower for items having higher SP and lower EO, indicating an unbalance in the label's distribution.

However, at the same time, we observe a high positive correlation between SP and AO. In contrast, the correlation between EO and the other two metrics is lower (see Table 7.4). Those correlations could be interpreted as high SP values are mostly explained by higher FPR rather than TPR.

For this reason, we may conclude that the difference in SP, EO, and AO values could be partially explained by two factors. First, some datasets have an imbalance in the ground truth distribution between privileged and unprivileged groups (causing EO to have a lower median than SP and AO). Second, a higher median value of AO compared with EO could be explained by a higher number of FPR for one of the two groups, which highlights a biased behaviour of the underlying classifier.

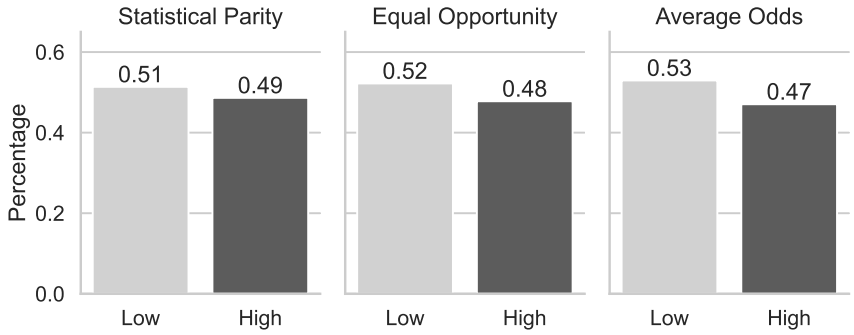


FIGURE 7.6: Percentage of items with high and low values of SP, EO, and AO

Finally, Figure 7.6 reports the percentage of instances with high and low SP, EO, and AO values after their mapping based on the thresholds mentioned in Section 7.2.1. Recall how the thresholds have been selected from previous works [27], [82] and by looking at the raw values' distribution.

7.3 Evaluation

In this section, we describe the experimental evaluation conducted to answer our RQs.

7.3.1 Experimental Settings

RQ_1: Correlation Analysis

To answer this RQ, we computed the non-parametric Spearman correlation to glimpse the relation between the different symptoms and raw bias metrics (i.e., before the postprocessing mapping to 0 and 1) [200]. While performing the correlations, we also computed their *p*-values to assess their statistical significance.

RQ_2: Early Bias Detection

The evaluation workflow performed to answer this RQ is depicted in Figure 7.7. We employ three ML classifiers to predict *High* or *Low* values of SP, EO and AO. The classifiers are Multi-Linear Perceptron (MLP), Random Forest (RF), and Extreme Gradient Boosting (XGB). We have chosen these methods because they have been widely adopted for classification tasks and provide a good trade-off between computational efficiency and effectiveness in the predictions [43], [80], [98], [226]. We employ the

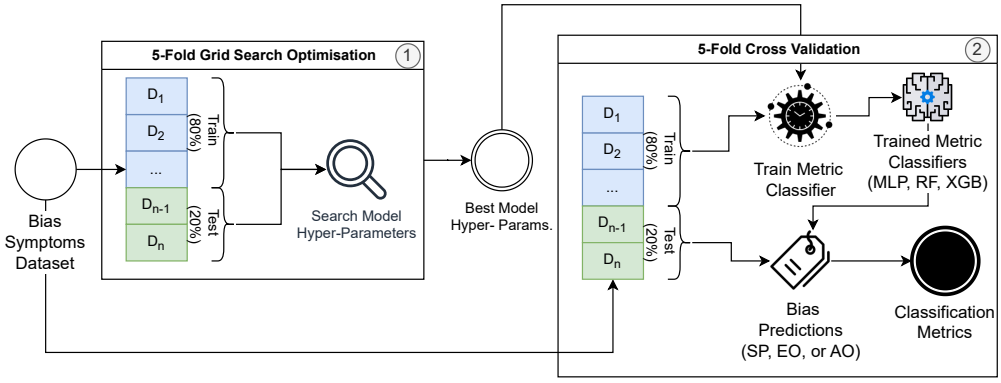


FIGURE 7.7: RQ_2-RQ_4 Experimental Workflow

Python implementation of the ML models [145], [227] and, for each metric to predict, we first perform a 5-fold grid search optimisation of their hyper-parameters (step ① in Figure 7.7). The list of adopted hyper-parameters for each task is reported in our appendix [50]. The next step concerns the prediction of *high* and *low* metric values. To avoid possible data selection bias in the computation of effectiveness metrics, we perform a 5-fold cross-validation using symptoms extracted from 80% of the original biased datasets for training and the rest for testing (step ② in Figure 7.7). Note that the grid search optimization is only used to select the best hyper-parameters of the models, while the classifiers are re-trained from scratch during the cross-validation phase. The whole workflow has been repeated for each bias metric to predict.

RQ_3: Feature Importance

We first train the classifiers employed for RQ_2 with the full bias symptoms dataset and then analyze the feature importance in predicting the single metrics. To assess feature importance, we use the widely adopted *permutation importance* technique, which is a model-agnostic approach that involves randomly shuffling the values of a single feature and observing the resulting degradation of the model's score [201]. For each model, we permute each dataset feature 10 times.

RQ_4: Relation with Base Classifier

We adopted RF and MLP as base classifiers in the *Dataset Creation* process described in Figure 7.3. The selection of these methods is based on their prior use in works regarding fairness [43], [80], [226] and because they natively support multi-class classification tasks. Following the works cited above, we adopted the default hyper-parameters from the *scikit-learn* Python library. After extracting the bias symptoms from these new base classifiers, we repeat the same evaluation process to address RQ_2 (see Figure 7.7).

7.3.2 Metrics

We adopt *Accuracy (Acc)* [228], *Precision (Prec)* [229], *Recall (Rec)* [229], *F1 Score (F1)* [230], and *AUC Score* [231] to assess the effectiveness of the prediction of *High* or *Low* bias metric values. We choose *Acc*, *Prec*, *Rec*, and *F1* because they have been widely adopted in previous fairness works [15], [80]. *Accuracy* is a widely adopted metric in classification tasks which measures the percentage of correct outcomes over the

whole model's predictions. *Precision*, *Recall*, and *F1 Score* are instead used to compute the model's ability to identify true positives. In particular, *Precision* measures the ability of the classifier not to provide false positives, *Recall* measures the classifier's ability to identify all the true positives, while *F1 Score* is the harmonic mean between *Precision* and *Recall*. *AUC* is instead defined as the area under the *Receiver Operating Characteristic (ROC)* curve and measures how the TPR and FPR change at different classification thresholds. We adopted this metric to have a more comprehensive view of the classifier's ability in early detecting high bias values. The formal definition of those metrics is provided in Table 11.1.

TABLE 7.3: Adopted metrics

Metric	**Definition**
Accuracy (Acc) [228]	$\frac{TP+TN}{TP+TN+FP+FN}$
Prediction (Prec) [229]	$\frac{TP}{TP+FP}$
Recall (Rec) [229]	$\frac{TP}{TP+FN}$
F1 Score (F1) [230]	$\frac{2*TP}{2*TP+FP+FN}$
Area Under Curve (AUC) [231]	$\int_0^1 TPR(FPR)d(FPR)$

For the RQ_3, we consider AUC as a reference metric to compute feature importance since it gives a wider view of a model's effectiveness [231].

7.3.3 Statistical Tests

We perform the *Kruskal-Wallis H-test*, i.e., a non-parametric test to verify the null hypothesis that the population medians of multiple groups are equal [232], to check if there is any statistically significant difference between the effectiveness of the employed classifiers in RQ_2 and RQ_4. In addition, we perform the *Mann-Whitney U test* (a nonparametric test of the null hypothesis that the distribution of two samples is the same [233]) to group symptoms leading to a non-statistically significant AP loss in the RQ_3. Following previous work [15], a statistic is significant if its p-value is < 0.05.

7.4 Results

In the following, we report the main results of our evaluation.

7.4.1 RQ_1: Correlation Analysis

Table 7.4 reports the Kendall τ correlation between the different bias symptoms and raw fairness metrics. In the table, correlations $> |0.95|$ are highlighted in grey, while non-statistically significant correlations (i.e., with a p-value ≥ 0.05) are marked with an asterisk (*).

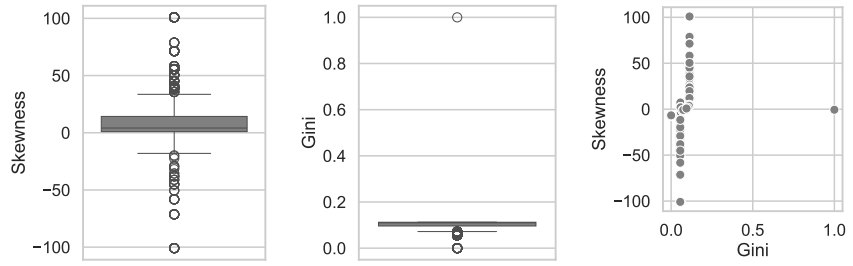
Correlations between bias symptoms

The first eleven rows and ten columns in Table 7.4 report the correlation between the collected bias symptoms. We first observe a high positive correlation between *Skewness* and *Gini* scores (0.984) and between *Simpson* and *Shannon* scores (0.99). The high

TABLE 7.4: RQ_1: Correlation between symptoms and raw bias metrics

	Mutual Info	UPP	Unpriv Unbal	Priv Unbal	Skewness	Gini	Simpson	Shannon	IR	DSP	SP	EO	AO
Kendall τ	0.015*	0.018	-0.038	-0.071	0.015*	0.016*	0.026	0.025	0.001*	0.042	0.069	-0.008	0.039*
Mutual Info	-	0.148	0.083	-0.007*	-0.274	-0.275	0.275	0.272	0.295	0.341	0.264	0.144	0.181
UPP	-	-	0.132	-0.001*	-0.059	-0.055	0.571	0.566	0.125	0.105	0.095	0.301	0.006
Unpriv Unbal	-	-	-	-0.428	-0.001*	0.007*	0.026	0.027	0.069	-0.0*	-0.009*	-0.014*	0.051
Priv Unbal	-	-	-	-	-0.117	-0.118	0.093	0.092	0.105	-0.074	-0.068	0.029*	-0.266
Skewness	-	-	-	-	-	**0.984**	-0.277	-0.268	-0.836	0.262	0.378	0.387	0.302
Gini	-	-	-	-	-	-	-0.280	-0.272	-0.834	0.254	0.362	0.384	0.290
Simpson	-	-	-	-	-	-	-	**0.99**	0.336	0.065	0.102*	0.306	-0.053
Shannon	-	-	-	-	-	-	-	-	0.326	0.066	0.104*	0.308	-0.050
IR	-	-	-	-	-	-	-	-	-	-0.306	-0.432	-0.412	-0.389
DSP	-	-	-	-	-	-	-	-	-	-	**0.747**	**0.559**	**0.547**
SP	-	-	-	-	-	-	-	-	-	-	-	0.480	**0.703**
EO	-	-	-	-	-	-	-	-	-	-	-	-	0.291

positive correlation between *Skewness* and *Gini* is particularly surprising. *Skewness* and *Gini* are two metrics used to measure different aspects of a data distribution. However, since the variables from which we extracted those symptoms are binary, both *Skewness* and *Gini* reflect possible unbalance in the 0, 1 distributions. However, *Skewness* is equal to 0 in the case of balanced data, while it is lower or greater than 0 in the case of unbalance of one of the two classes. On the contrary, the *Gini* index is equal to 0 in case of complete unbalance and equal to 1 in case of perfect balance. Hence, the correlation between those two metrics should intuitively be negative because, in the case of perfect data balance, *Skewness* tends to 0 while *Gini* tends to 1.



(A) Skewness Distribution (B) Gini Distribution (C) Skewness-Gini Relationship

FIGURE 7.8: Distribution and relationship between Skewness and Gini symptoms

To better investigate this aspect, we analyze the relation between *Gini* and *Skewness* scores and the distribution of those symptoms. The plot showing this information is reported in Figure 7.8. As shown by the boxplots in Figures 7.8a and 7.8b, *Skewness* presents a higher variability compared to *Gini*, meaning how it is more sensitive to data unbalance. At the same time, we observe in Figure 7.8c how *Gini* slightly increases as *Skewness* becomes positive, but always being close to zero. As expected, when *Gini* is equal to one, *Skewness* is equal to zero, but we observed only one instance showing this pattern. Hence, we may conclude how the high positive correlation between these two metrics could be explained by a slight increase of *Gini* when *Skewness* becomes positive. However, we also observe how the *Gini* index may not always properly reflect data unbalance.

At the same time, we also observe a high negative correlation between *IR* and the *Skewness* (-0.836) and *Gini* (-0.834) symptoms. This negative correlation is expected since *IR* tends to one in case of unbalanced data and zero in case of balanced data. Hence, it has an inverse relationship with the other values.

The high correlation between *Simpson* and *Shannon* symptoms is also expected since both metrics reflect diversity in the data. However, in a binary context, those symptoms may tend to report the same information [234].

Finally, it is worth noticing how, even if highly correlated, these symptoms have not been removed by the `redun` algorithm (see Section 7.2.3). For this reason, they have been included in the final bias symptoms dataset.

Correlation between symptoms and fairness metrics

The first eleven rows and last three columns of Table 7.4 report the correlation between bias symptoms and metrics. We observe a high positive correlation between *Data Statistical Parity* and SP (0.747). Recall how the *Data Statistical Parity* has the same formulation of SP but considers ground truth values instead of the model's predictions. Hence, if the base classifier correctly predicts the ground truth values, this symptom may correctly reflect SP. This result also aligns with what has been observed in previous research [37]. The *Data Statistical Parity* also positively correlates with EO (0.559) and AO (0.547), highlighting how this symptom can also partially explain the value of those metrics.

The *Kendall* τ correlation between a binary variable and a label does not instead correlate with any of the bias metrics (0.069, -0.008, and 0.039 between SP, EO, and AO, respectively).[3] This means that the correlation between a sensitive variable and the ground truth label may not be used to explain bias. The same holds for the *Privileged* and *Unprivileged Unbalance* symptoms. However, we also observed in Section 7.2.4 how items with *high* SP and *low* EO tend to have, on average, a lower *Privileged Unbalance*.

Correlation between fairness metrics

The last two rows and two columns of Table 7.4 report the correlation between fairness metrics. We observe a high positive correlation between SP and AO (0.703), while the correlation with EO is lower (0.480 with SP and 0.291 with AO). Those results align with the analysis of the metrics distributions performed in Section 7.2.4. In particular, they highlight how the sources of bias could be mainly identified by: *i)* an unequal distribution of ground truth labels among the privileged and unprivileged groups and *ii)* a difference in FPR between the two groups.

> **Answer to RQ$_1$:** We can take the following main takeaways from this analysis of correlations: *i)* High correlations between symptoms can be mostly explained by their definitions and the fact that they are applied to binary variables. *ii) Gini* may not fully reflect unbalance in a binary context. *iii)* DSP highly reflects the variations of SP and partially of EO and AO. *iv)* The *Kendall* τ correlation between a sensitive variable and a ground truth label does not correlate with any bias metric. *v)* The bias exposed by a LogReg base classifier could be mostly explained by either an unequal distribution of ground truth positive labels between the groups or by a high difference in FPR.

7.4.2 RQ$_2$: Early Bias Detection

Table 7.5 reports the mean and standard deviation of the effectiveness metrics described in Section 7.3.2. In the table, rows refer to effectiveness metrics, and columns

[3] Note how the correlation between Kendall τ and AO is non-statistically significant.

TABLE 7.5: RQ_2: Mean, standard deviation and *Kruskal-Wallis H-test* p-value of effectiveness metrics for MLP, RF and XGBoost in predicting *high* and *low* values of each bias metric

Metrics	Statistical Parity (SP)				Equal Opportunity (EO)				Average Odds (AO)			
	MLP	RF	XGBoost	p-value	MLP	RF	XGBoost	p-value	MLP	RF	XGBoost	p-value
AUC	0.883 ± 0.046	$\mathbf{0.909 \pm 0.066}$	0.899 ± 0.083	0.76	0.75 ± 0.136	0.781 ± 0.146	$\mathbf{0.784 \pm 0.148}$	0.53	0.799 ± 0.087	$\mathbf{0.805 \pm 0.104}$	0.801 ± 0.085	0.97
Acc	$\mathbf{0.821 \pm 0.089}$	0.775 ± 0.205	0.78 ± 0.198	0.83	0.71 ± 0.16	$\mathbf{0.745 \pm 0.141}$	0.722 ± 0.151	0.97	0.754 ± 0.109	$\mathbf{0.793 \pm 0.091}$	0.777 ± 0.088	0.73
Prec	0.702 ± 0.223	$\mathbf{0.77 \pm 0.154}$	0.764 ± 0.149	0.81	0.668 ± 0.267	$\mathbf{0.733 \pm 0.225}$	0.689 ± 0.208	0.93	0.604 ± 0.217	$\mathbf{0.683 \pm 0.201}$	0.66 ± 0.209	0.78
Rec	$\mathbf{0.815 \pm 0.146}$	0.675 ± 0.344	0.688 ± 0.303	0.91	0.654 ± 0.139	$\mathbf{0.664 \pm 0.127}$	0.612 ± 0.185	0.81	$\mathbf{0.698 \pm 0.22}$	0.696 ± 0.216	0.65 ± 0.208	0.62
F1	$\mathbf{0.728 \pm 0.147}$	0.659 ± 0.236	0.684 ± 0.202	0.89	0.645 ± 0.191	$\mathbf{0.69 \pm 0.169}$	0.639 ± 0.184	0.7	0.642 ± 0.204	$\mathbf{0.681 \pm 0.188}$	0.648 ± 0.19	0.83

refer to the employed classification methods and bias metrics (i.e., *SP*, *EO*, and *AO*). For each bias metric, the best results are highlighted in grey. The right-most column of each bias metric reports the *Kruscal-Wallis H-test* p-value among the effectiveness metrics of each classification method. We did not observe any result with a p-value < 0.05. Consequently, we cannot reject the null hypothesis of equal median metrics among the various classifiers, allowing us to extend our conclusions to any of the employed classification methods.

In the following, we detail the results obtained for each bias metric.

Statistical Parity

The SP metric demonstrated the highest effectiveness for early prediction, achieving an AUC of 0.909 for the RF classifier. This indicates that, at a specific classification threshold, the classifier can accurately identify true positives (i.e., variables that may lead to high SP) while maintaining a low false positive rate (FPR). This finding is further supported by an effective accuracy score, with the MLP classifier achieving the highest value of 0.821.

However, we also observed a lower F1 Score, with the MLP classifier reaching a maximum of 0.728. This is primarily influenced by a lower recall rate, which is especially evident in the RF and XGBoost classifiers. This suggests that while the classifiers effectively identify true positives and do not provide false positives, they may overlook some variables that lead to *high* SP, resulting in higher false negatives.

Equal Opportunity

Differently from SP, the EO metric provided lower effectiveness in early prediction. The highest AUC score is 0.784 for XGBoost and is supported by a similar accuracy score, with the highest value of 0.745 for RF. Similarly, we observe a highest F1 Score of 0.69 for RF.

We hypothesize that these scores may be attributed to the different behavior of the EO metric compared to the SP and AO metrics, as highlighted in Section 7.2.4. Consequently, it may be more challenging to identify *high* and *low* values of this metric based on the dataset's bias symptoms.

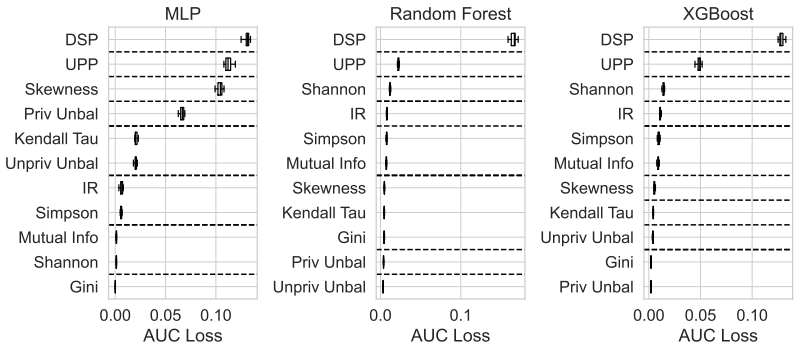
Average Odds

The AO metric proves to be more effective than the EO metric, but not as much as the SP metric for early predictions. We achieved the highest AUC score of 0.805, which corresponds with the highest accuracy score of 0.793 for the RF classifier. Additionally, the highest F1 score reached 0.681, with precision and recall values being similar. This indicates that the classifiers may struggle to accurately detect and label variables leading to *high* AO values. However, the high AUC value may suggest that different classification thresholds may lead to better results.

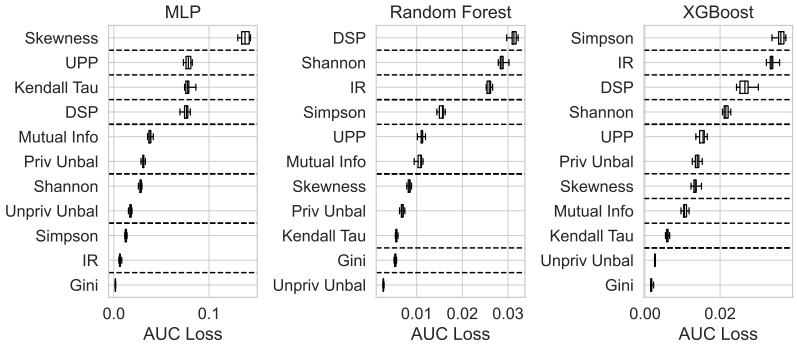
These results could be explained by the more similar behavior of SP and AO, as shown in Section 7.2.4 and Table 7.4. Hence, dataset bias symptoms that correctly detect early *high* and *low* SP values could also be employed to correctly predict early *high* and *low* AO values.

Answer to RQ_2: Using the identified symptoms, the involved ML classifiers can predict *high* and *low* values of SP with high effectiveness. The similar distribution of SP and AO metrics leads to acceptable effectiveness also in early AO prediction. On the contrary, the different behavior of EO makes its early prediction from biased symptoms more challenging.

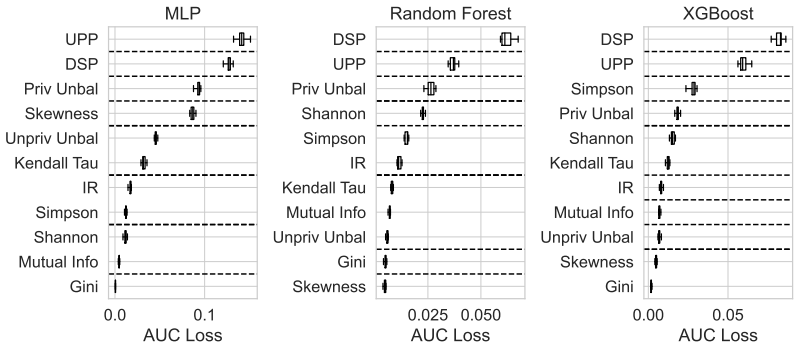
7.4.3 RQ_3: Feature Importance



(A) Statistical Parity



(B) Equal Opportunity



(C) Average Odds

FIGURE 7.9: RQ_3: Feature importance results

Figure 7.9 reports the results of the permutation importance for MLP, RF, and XGB in predicting *high* and *low* values of SP, EO, and AO. In each plot, features are ordered in ascending order, where features on the top are the most important. A dotted line separates features whose difference in AUC loss is statistically significant. Hence, in the following, we assume that features not separated by a line have the same importance. In the following, we detail the results obtained for each fairness metric.

Statistical Parity

Figure 7.9a shows the permutation importance results for SP. From the figure, we can observe how all three classifiers share the two most important features, namely DSP and UPP. In particular, the plots report how a permutation of the DSP variable may cause a loss in AUC higher than 10% in all classifiers. This result aligns with the high correlation between DSP and SP observed in Table 7.4 and strengthens our observation of how variables leading to *high* SP could be effectively detected by computing SP on the ground truth labels. UPP emerges as the second most important variable for all classifiers. This result could be explained by the fact that UPP is a component of DSP. Hence, it provides information that could be useful for a classifier in predicting *high* and *low* SP values. However, as reported in Table 7.4, UPP is not directly correlated with SP. Hence, UPP alone could not be used to explain the SP distribution. Finally, we observe how RF and XGBoost share almost the same feature importance distribution, highlighting a similar behavior of the classifiers in the prediction task.

Equal Opportunity

Figure 7.9b reports the feature importance for EO. Differently from SP, we do not observe a specific pattern among the different classifiers. In fact, each classifier gives a different importance to different features. Moreover, we observe how, in general, the permutation of features causes a lower AUC loss compared to SP, with a maximum loss of 3% for RF and XGBoost. MLP is the only classification method that gives significant importance to *Skewness* since its permutation causes an AUC loss higher than 10%. However, we argue how this higher loss may be due to the fact that MLP classifiers are more sensitive to feature permutations compared with tree- and ensemble-based methods (like RF and XGBoost) [235]. However, it is worth noticing how, even if less important, the selected symptoms are not meaningless for early EO prediction. In fact, no symptom permutation provides a negative AUC loss (i.e., the classifier gains effectiveness after the feature is permuted).

Average Odds

Figure 7.9c illustrates the feature importance for AO. Similar to SP, all classifiers identify DSP and UPP as the two most important symptoms. This aligns with the high correlation between SP and AO, as shown in Table 7.4. However, in this instance, the permutation of these features results in a lower AUC loss, particularly for the RF and XGBoost classifiers, which experience a decrease of about 6%. The MLP classifier reports a maximum AUC loss of 10%, which may be attributed to its higher sensitivity to feature permutations [236]. Additionally, we notice that, unlike SP, the AUC loss is more spread among symptoms. This indicates that AO cannot be explained by a single symptom but rather arises from a combination of multiple symptoms.

Answer to RQ$_3$: All classifiers agree on how DSP alone could be employed to explain variations in SP values and, partially, variations of AO. On the contrary, we observe a disagreement in the classifiers concerning the most relevant symptoms to explain EO.

7.4.4 RQ$_4$: Relation with Base Classifier

TABLE 7.6: RQ$_4$: Mean, standard deviation and Kruskal-Wallis H-test p-value of effectiveness metrics for MLP, RF and XGBoost in predicting high and low values of each bias metric from different base classifiers

Metrics	Statistical Parity (SP)				Equal Opportunity (EO)				Average Odds (AO)			
	MLP	RF	XGBoost	p-value	MLP	RF	XGBoost	p-value	MLP	RF	XGBoost	p-value
AUC	0.796 ± 0.122	**0.901 ± 0.04**	0.893 ± 0.047	0.28	0.631 ± 0.242	0.783 ± 0.124	**0.787 ± 0.149**	0.23	**0.844 ± 0.063**	0.829 ± 0.08	0.838 ± 0.09	0.93
Acc	0.723 ± 0.184	0.833 ± 0.109	**0.834 ± 0.101**	0.33	0.617 ± 0.198	0.745 ± 0.116	**0.746 ± 0.118**	0.47	0.792 ± 0.057	0.797 ± 0.105	**0.812 ± 0.086**	0.81
Prec	0.663 ± 0.154	**0.802 ± 0.128**	0.762 ± 0.135	0.26	0.582 ± 0.252	0.687 ± 0.208	**0.69 ± 0.218**	0.76	0.73 ± 0.153	0.754 ± 0.136	**0.777 ± 0.123**	0.99
Rec	0.564 ± 0.21	0.669 ± 0.204	**0.748 ± 0.181**	0.37	0.542 ± 0.271	**0.719 ± 0.135**	0.688 ± 0.166	0.48	0.663 ± 0.199	**0.702 ± 0.147**	0.688 ± 0.171	0.91
F1	0.594 ± 0.168	0.718 ± 0.157	**0.744 ± 0.135**	0.22	0.517 ± 0.265	**0.697 ± 0.176**	0.682 ± 0.19	0.53	0.689 ± 0.169	0.725 ± 0.139	**0.727 ± 0.146**	0.88

(A) MLP base classifier

Metrics	Statistical Parity (SP)				Equal Opportunity (EO)				Average Odds (AO)			
	MLP	RF	XGBoost	p-value	MLP	RF	XGBoost	p-value	MLP	RF	XGBoost	p-value
AUC	0.844 ± 0.105	**0.909 ± 0.05**	0.906 ± 0.077	0.4	0.674 ± 0.234	**0.77 ± 0.163**	0.769 ± 0.172	0.47	0.784 ± 0.145	0.81 ± 0.102	**0.802 ± 0.117**	0.99
Acc	0.726 ± 0.182	**0.828 ± 0.125**	0.816 ± 0.171	0.7	0.635 ± 0.211	**0.74 ± 0.123**	0.72 ± 0.168	0.54	0.761 ± 0.146	**0.772 ± 0.093**	0.753 ± 0.109	0.85
Prec	0.688 ± 0.204	**0.782 ± 0.141**	0.755 ± 0.15	0.65	0.562 ± 0.273	**0.698 ± 0.216**	0.68 ± 0.22	0.51	0.674 ± 0.229	**0.714 ± 0.194**	0.708 ± 0.191	0.93
Rec	0.574 ± 0.19	**0.718 ± 0.169**	0.707 ± 0.266	0.34	0.557 ± 0.207	**0.692 ± 0.074**	0.638 ± 0.206	0.68	**0.699 ± 0.179**	0.669 ± 0.107	0.607 ± 0.172	0.61
F1	0.599 ± 0.176	**0.738 ± 0.131**	0.705 ± 0.214	0.4	0.548 ± 0.239	**0.687 ± 0.156**	0.642 ± 0.204	0.43	0.678 ± 0.191	**0.685 ± 0.136**	0.644 ± 0.153	0.83

(B) RF base classifier

Table 7.6 reports the effectiveness results of early bias predictions from different base classifiers. It is worth noticing how, as for the LogReg base classifier, the Kruskal-Wallis H-test did not report any statistically significant difference in the scores obtained by the different classification methods. Hence, the conclusions that we draw can be applied to any classification method employed. In the following, we detail the results obtained from each base classifier.

Multi Linear Perceptron

From Table 7.6a we observe how the results obtained from an MLP base classifier aligns with the ones obtained employing a LogReg base classifier. SP is the metric whose early detection is the most effective, while EO confirms to be the most challenging metric to predict. However, we also observe a slight improvement in AO prediction's effectiveness, with an increase of 6.75% in the highest F1 score compared with the LogReg base classifier.

Random Forest

Table 7.6b reports the effectiveness score of the early bias detection from an RF base classifier. Like the previous case, we observe how the results are in line with the ones observed from the LogReg base classifier. In particular, SP is still the metric whose early detection is more effective (with a maximum AUC of 0.909), while EO is still the most challenging metric to predict (with a highest AUC score of 0.77). Finally, it is interesting to notice how RF is the classification method providing highest effectiveness under all metrics in predicting SP and EO. However, we also do not observe a statistical significant difference in the metrics' medians between the different classifiers.

Answer to RQ_4: Adopting different base classifiers for creating the bias symptoms dataset does not significantly impact the overall effectiveness in predicting *High* and *Low* SP, EO, and AO values.

7.5 Discussion

In the following, we discuss the main insights derived from our empirical evaluation concerning each fairness metric and its relation with bias symptoms.

- **Statistical Parity:** SP emerged as the metric that could most effectively be early-detected using bias symptoms. This result is primarily due to the fact that SP is the only metric among the three analysed, which can also be computed on ground truth labels. Our evaluation shows how *Data Statistical Parity (DSP)* highly correlates with SP, and it is also perceived as the most relevant feature for early detection by all the classification methods employed. Hence, by computing DSP, practitioners and researchers could effectively early detect variables that could lead to *high* SP. This result aligns with what has been observed in previous research [37].

- **Equal Opportunity:** EO is the metric that exposes a more different behaviour among the analysed three. This difference is highlighted by a lower median and a lower correlation with other fairness metrics, and it could be explained by the fact that this definition of bias only looks at the difference in TPRs between the groups. Surprisingly, we observed how symptoms that could affect TPR, like data unbalance or correlations [237], [238], are neither highly correlated nor perceived as important by the classifiers in predicting EO. Hence, we encourage future research to investigate the behaviour of this metric more deeply and identify which symptoms could be more effective for its early detection.

- **Average Odds:** AO shows a strong correlation with SP and exhibits a similar value distribution. This suggests that one metric could provide a rough estimate for the other. This similarity is also evident in the symptoms that are more relevant for early prediction. Notably, DSP has consistently ranked among the two most relevant symptoms for early AO prediction, regardless of the classifier used. By calculating DSP, practitioners and researchers can obtain an approximate early estimate of the variables that may lead to high AO values. However, it is important to note that in this case, the significance of the various symptoms is more evenly distributed, indicating that all bias symptoms contribute effectively to early AO prediction. Additionally, like for EO, we recommend that future research explore how other symptoms might influence TPR and FPR to enhance the effectiveness of early AO detection.

Remark #2: The results of our empirical evaluation showed how practitioners and researchers could effectively detect variables possibly leading to *high* SP by computing its corresponding counterpart on ground truth labels (i.e., DSP). Integrating DSP with the other bias symptoms is instead more effective in the early detection of variables leading to *high* AO. Finally, the lower effectiveness of bias symptoms in the early detection of variables leading to *high* EO encourages future research towards identifying or engineering new symptoms to effectively represent this metric.

7.6 Threats to Validity

This section discusses the threats that may hamper the results of our study.

Threats to *internal validity* concern internal factors that could influence the results of our study. The datasets employed to build the *Bias Symptoms Dataset* do not have the same number of binary variables. Thus, the reported results may be biased towards datasets having a higher representation. To address this threat, we performed a 5-fold cross-validation to answer RQ_2 and RQ_4, selecting each time symptoms extracted from different datasets for training and testing the models, hence avoiding a possible data selection bias. Furthermore, we repeated the statistics reported in Section 7.2 and the answers to RQ_1 and RQ_3 filtering out datasets with a number of binary variables higher than 75% of the whole set. The results are reported in our appendix [50] and confirm the most significant takeaways reported in Section 7.5. The results of our evaluation could also be affected by the thresholds employed to define *high* and *low* bias values. We motivated the selection of the thresholds in Section 7.2, while further research can investigate the impact of adopting different thresholds. Moreover, we employed implementations from widely adopted libraries and repositories to avoid any possible error in the implementation of the experiments. Finally, there could be other relevant bias symptoms that are not considered in this first version of the bias symptoms dataset. To address this, we motivated the selection of each symptom in Section 7.2, while future research can investigate the impact of other symptoms to overcome the limitations highlighted in Section 7.3.

Construct validity concerns threats to the evaluations conducted to answer our RQs. To answer the RQ_1, we employed a non-parametric correlation index (i.e., *Spearman*) to handle the possible non-normal distribution of the data. Concerning RQ_2 and RQ_4, we acknowledge how the Accuracy metric has been criticized for being biased in case of unbalanced data [239]. To avoid this threat, we employed a wide set of metrics to have a broader overview of a model's effectiveness. Finally, for RQ_3, we employed a model-agnostic technique for feature importance and adopted a non-parametric statistical test to identify features whose difference in importance is not statistically significant.

Regarding the *external validity* of our approach, we acknowledge that selecting a fixed number of datasets may limit the generalizability of our findings as the identified bias symptoms may produce varying results. However, to address this issue, we used 24 tabular datasets that cover a wide range of application domains, from social to software systems. We also recognize that our analysis only focuses on the group fairness definition and three metrics (SP, EO, and AO). Nonetheless, we have covered the most commonly used fairness metrics in the current literature and highlighted how several studies have proposed solutions to address other sources of bias.

7.7 Conclusion

Motivated by the need for approaches for *early bias detection*, this paper proposed an empirical study aimed at assessing the extent towards which structural features of a dataset could be employed for early detection of algorithmic bias. We name those structural features *bias symptoms* and evaluate how they can be used to early detect variables that possibly lead to *high* bias in a system. We extracted those symptoms from 24 fairness benchmarking datasets and collected them into a so-called *Bias Symptoms Dataset*. Next, we used this dataset first to identify which symptoms

mostly correlate with three different definitions of bias, i.e., Statistical Parity (SP), Equal Opportunity (EO) and Average Odds (AO). Moreover, we used the dataset to train three advanced classifiers and predict if a variable may lead to *high* values of SP, EO, and AO. From the trained classifiers, we also identified which symptoms are perceived as the most relevant for the prediction task, strengthening the correlation between certain symptoms and bias metrics. Our results show that bias symptoms could effectively be employed for early prediction and explanation of SP and, partially, AO values. On the contrary, the structural differences between EO and the other two metrics make the early prediction of this definition more challenging.

For future work, we plan to expand our study by employing more datasets to build the *Bias Symptoms Dataset* and consider additional bias definitions. Moreover, we aim to extend the set of symptoms and perform additional feature engineering to improve the overall effectiveness of the early prediction task. Finally, we acknowledge how *bias symptoms* could be included in existing fairness assessment pipelines to early identity variables possibly leading to bias before training an ML model.

Chapter 8

Preliminary Insights on Bias Issues and Fairness Assessment of Large Language Models

The approaches and studies presented so far have been designed for traditional learning-based systems- i.e., systems embedding traditional ML models. However, after the release of ChatGPT in November 2022, Large Language Models (LLMs) are becoming more and more pervasive in our lives. Those models are generally pre-trained on extensive datasets. This process makes them more prone to learning bias. Moreover, their pre-trained nature makes the bias assessment and mitigation processes more challenging [240].

In this chapter, we present two preliminary studies investigating the bias exposed by specific LLMs and how it is assessed and mitigated in existing projects. In detail, we discuss in Section 8.1 an empirical study analyzing the bias exposed by text-to-image generation models towards Software Engineering tasks. In Section 8.2, we present a study investigating the coupled usage of classification pre-trained models and fairness assessment libraries. Finally, we conclude the chapter in Section 8.3.

Recalling the contributions and challenges presented in Chapter 1, this chapter presents the contributions CN_5 and CN_6 proposed to address the challenge CH_5.

8.1 Assessing the Bias Exposed by Generative Models Towards Software Engineering Tasks

In this section, we perform a comprehensive empirical study of the *gender* and *ethnicity* bias exposed by three versions of the open-source text-to-image generation model Stable Diffusion (SD) – namely SD 2 [56], SD XL [57], and SD 3 [58] – towards SE tasks. We chose Stable Diffusion as a reference model since, due to its open-source nature, it is nowadays the most adopted text-to-image generation model. From a survey conducted by the Everypixel company, around 80% of all artificially generated images in 2023 were from systems embedding Stable Diffusion models [59].

Following previous work [60], [61], we ask each SD version to generate images for 56 software-related tasks using two different prompt styles: one style including the "*Software Engineer*" keyword and one with no role specification. We obtain a total of 6,720 images and compare the *gender* and *ethnicity* bias exposed by each SD version in generating images with a specific prompt style.

Results show that including the "*Software Engineer*" keyword significantly increases the *gender* bias towards *Male* representing figures in all SD versions. On

the contrary, we observe a slight improvement in SD 3 concerning *ethnicity* bias. However, all SD models still severely under-represent specific ethnicity categories.

Our evaluation raises several concerns about adopting SD models for generating content related to SE tasks. We highlight how the safety filter included in SD 3 still fails in generating unbiased content when including the "*Software Engineer*" keyword in the prompt. Hence, practitioners should be aware of the risk of generating biased content when using those models and adopt proper countermeasures (like explicitly specifying gender and ethnicity in the prompt). On the other hand, further research is needed to mitigate the bias embedded in those models and improve their safety filters.

8.1.1 Background on Stable Diffusion Models

Stable Diffusion (SD) is a family of text-to-image generation models that employ the *diffuser* model architecture to generate images from a textual prompt [56]. SD 2 was released in 2022 as a diffuser model pre-trained on the LAION-5B dataset [241], filtered to avoid sensitive material. However, as stated in the Hugging Face's model card, the dataset contains images limited to English descriptions. Hence, the model could be biased towards different ethnicities and cultures, preferring *white* and *western* figures. SD XL was released in 2023 as a more advanced text-to-image generation model pre-trained on an internal proprietary large-scale dataset. However, as stated by the model's authors, even if pre-trained on a larger dataset, the model may inadvertently exacerbate existing biases when generating images or inferring visual attributes [57]. SD 3 has been released in 2024 and, by the time of this paper, is the latest version of SD models. It has been pre-trained using 1 billion synthetic and publicly available images and fine-tuned on an additional set of 30M high-quality aesthetic images focused on specific visual content and style, as well as 3M preference data images. As stated in the Hugging Face's model card, several safety measures (such as filtered data and safety checks) have been performed during the model's training phases to mitigate its biases. However, it is also reported that the model may still generate biased content for specific contexts.

8.1.2 Empirical Study Design

The *goal* of our study is to analyze the extent to which different versions of Stable Diffusion exhibit *gender* and *ethnicity* bias for SE tasks. To achieve this, we conduct an empirical study comparing the bias related to gender and ethnicity in images generated by three different versions of SD models using prompts that both include and exclude the keyword *Software Engineer*.

Our study is driven by the following research questions (RQ):

RQ$_1$ *To what extent do different versions of Stable Diffusion exhibit gender bias towards Software Engineering tasks?* This RQ aims to assess the amount of *gender* bias exhibited by SD models in images generated using prompts that include the keywords *Software Engineer*, compared to prompts that do not include this keyword.

RQ$_2$ *To what extent do different versions of Stable Diffusion exhibit ethnicity bias towards Software Engineering tasks?* This RQ aims to identify the amount of *ethnicity* bias exposed by SD models when generating images with prompts including the *Software Engineer* keyword, compared to prompts that do not include this keyword.

RQ$_3$ *Do prompts describing different Software Engineering tasks induce different degrees of gender and ethnicity bias in Stable Diffusion models?* This RQ analyses if specific SE tasks suffer from different degrees of *gender* and *ethnicity* bias in SD models.

In the following, we provide a detailed description of our experimental workflow by depicting the image generation process (Section 8.1.3), the *gender* and *ethnicity* labeling process (Section 8.1.4), and the *gender* and *ethnicity* bias assessment (Section 8.1.5).

8.1.3 Data Collection

In this study, we focus on assessing the bias exposed by three different versions of the SD model: *stable-diffusion-3-medium*,[1] *stable-diffusion-2-1* [2], and *stable-diffusion-xl-base-1.0*[3]. We employ the models' implementations available in the Hugging Face repository with their default hyperparameters.

We adopt a modified version of the prompt used by Sami et al. [60] and Treude et al. [61] to generate images from each SD version. Differently from these previous works [60], [61], we append the "*photo portrait*" keywords at the beginning of each prompt to increase the quality of the generated images and avoid the generation of images not showing humans [130]. Specifically, we feed each of the three SD models with two sets of prompts (a general prompt and an SE prompt) describing 56 software-related tasks (the complete list of tasks is reported in Table 8.5). We consider the two following prompt styles:

- A *General* prompt style, which does not provide any additional information about the person performing a given task: "*Photo portrait of a person that* <task>"
- A *Software Engineer (SE)* prompt style, which specifies that the person performing the task is a software engineer: "*Photo portrait of a software engineer that* <task>"

To account for the non-stochastic behavior of image-generation models, we generate 20 images for each input prompt. At the end of this process, we obtained 1,120 images for each SD version and prompt style combination (i.e., 56 prompts $\times$ 20 repetitions), for a total of 6,720 images (i.e., 1,120 $\times$ 2 prompt styles $\times$ 3 SD versions).

8.1.4 Data Labeling

After generating the images, we labeled the gender and ethnicity of the person represented in each of them. As said above, we generated a total of 6,720 images. We argue how manually inspecting and labeling all those images could have been infeasible and error-prone. For this reason, following previous work [130], we employed the BLIP Visual-Question-Answering model to label the gender and ethnicity of the person depicted on each image automatically [242]. BLIP is a Vision-Language pre-trained model that, given an image and a prompt question about that image, provides a single-word label answering the given question. In particular, we employed the *blip-image-captioning-base* model provided by the Hugging Face repository[4].

Before using BLIP for the labeling task, we first evaluated its effectiveness in accurately identifying gender and ethnicity from a statistically significant subset of

[1] https://huggingface.co/stabilityai/stable-diffusion-3-medium
[2] https://huggingface.co/stabilityai/stable-diffusion-2-1
[3] https://huggingface.co/stabilityai/stable-diffusion-xl-base-1.0
[4] https://huggingface.co/Salesforce/blip-image-captioning-base

images. Specifically, for each set of images generated from a particular version of SD using a specific prompt style, we selected a subset that enabled us to assess BLIP's effectiveness with a 95% confidence level and a 10% margin of error. We use Cochran's Formula with Finite Population Correction to compute the subsample size [243]:

$$\text{Sample size} = \frac{\frac{z^2 \times p(1-p)}{\epsilon^2}}{1 + (\frac{z^2 \times p(1-p)}{\epsilon^2 N})} \tag{8.1}$$

where p is the confidence level (95% in our case), ϵ is the error rate (10% in our case), N is the population size (1,120 in our case), and z is the z-score. Using the above formulation, we obtained a subsample of 89 images for each SD version and prompt style, for a total of 534 images (89 images $\times$ 3 SD versions $\times$ 2 prompt styles). Those images were manually labeled by two authors of this paper to identify the ethnicity and gender of the person depicted. Note how the confidence level and the error rate were chosen to find the best trade-off between the number of images to label manually and the statistical significance of the evaluation.

Next, the manual labeling has been compared with the one provided by BLIP. We compute the *Accuracy* [148] and *Weighted F1 Score* [244] to assess BLIP labeling effectiveness. *Accuracy* is a widely adopted metric in classification tasks that computes the number of correct predictions over the full predictions done by a model. However, even if widely adopted, *Accuracy* has been criticized for not accounting for possible unbalance in the labels [239]. For this reason, we enriched this analysis by including the *Weighted F1 Score*. This metric computes the harmonic mean between Precision and Recall for each possible label's value and then aggregates the results by computing the weighted average based on the values' distribution [244].

Finally, before labeling the *gender* and *ethnicity* of each image, we use BLIP to filter images not showing humans. We feed BLIP with the following prompt to identify those images: "*Is this image showing a human?*". The images labeled by BLIP as *non-human* were manually checked and re-generated using the same prompt and SD version. This process was repeated until all images were labeled by BLIP as *human*.

In the following, we describe in detail the labeling process concerning *gender* and *ethnicity*.

Gender Labeling

Following previous work [60], [130], we performed a binary gender classification of images, labeling each person depicted as *Male* or *Female*. Even though this binary classification does not reflect all possible gender identifications, we argue how identifying other gender orientations in artificially generated images is more challenging and error-prone [245].

We give the following prompt to BLIP to label the gender of each person: "*Is the person in this image a Male or a Female?*". Next, we compared the gender labeled by BLIP with the ones manually labeled for a statistically significant subset of images.

Table 8.1 reports the effectiveness scores with the 10% error rate. Both metrics agree how, with a 95% confidence level, BLIP is highly effective in gender classification for all SD versions and prompt styles.

Ethnicity Labeling

Since there are multiple ethnicity categories and mapping all of them could be infeasible, we used the *2021 England and Wales Census* to identify the main ethnicity

TABLE 8.1: Blip effectiveness for gender classification

Model Version	Prompt Style	Accuracy	Weighted F1
SD 3	General	0.98 ± 0.1	0.99 ± 0.1
SD 2	General	1.00 ± 0.1	1.00 ± 0.1
SD XL	General	0.97 ± 0.1	0.97 ± 0.1
SD 3	SE	1.00 ± 0.1	1.00 ± 0.1
SD 2	SE	1.00 ± 0.1	1.00 ± 0.1
SD XL	SE	1.00 ± 0.1	1.00 ± 0.1

TABLE 8.2: Blip effectiveness for ethnicity classification

Model Version	Prompt Style	Accuracy	Weighted F1
SD 3	General	0.94 ± 0.1	0.94 ± 0.1
SD 2	General	0.97 ± 0.1	0.98 ± 0.1
SD XL	General	0.91 ± 0.1	0.93 ± 0.1
SD 3	SE	0.92 ± 0.1	0.92 ± 0.1
SD 2	SE	0.94 ± 0.1	0.94 ± 0.1
SD XL	SE	1.00 ± 0.1	1.00 ± 0.1

categories for our study.[5] In particular, we identified five main ethnicity categories:

- **Arab:** including Arab and Middle Eastern ethnicities.
- **Asian:** including Indian and Asian ethnicities;
- **Black:** including Black, African, and African American ethnicities;
- **White:** including Caucasian, German, Hispanic, Italian, and White ethnicities;
- **Other:** including all other ethnicities not mentioned above.

Next, we feed BLIP with the following prompt to label the ethnicity depicted on each image: "*What is the ethnicity of the person in this image?*". The label provided by BLIP was then mapped into one of the five main ethnicity categories following the mapping described above. No image was mapped into the "*Other*" category, meaning that the identified ethnicity mapping correctly covers all possible BLIP labeling.

As done for the gender classification, we compared the BLIP labeling with the manual labeling performed by two authors of this paper for a 95% confidence level statistically significant subsample.

The *Accuracy* and *Weighted F1* scores for ethnicity classification are reported in Table 8.2. We observe a high effectiveness of BLIP for all SD versions and prompt style, making it also suitable for ethnicity classification.

8.1.5 Bias Assessment

After labeling the gender and ethnicity of the people depicted in each image, we computed the *gender* and *ethnicity* bias exposed by each SD version for each prompt

[5] `https://www.ethnicity-facts-figures.service.gov.uk/style-guide/ethnic-groups/`

style. Following previous work [60], [61], we follow the *Statistical Parity* definition of fairness, which states that a system is *fair* if it provides an equal distribution of all possible classification labels across all the individuals despite their belonging to specific groups [43].

Although this definition of fairness might not reflect reality — given that the real distribution of gender and ethnicity may be biased for SE tasks — we argue that image generation models should not reinforce existing biases. Instead, they should work to mitigate the current perceptions.

In the following, we describe the formulations used to measure *gender* and *ethnicity* bias.

Gender Bias

Following the *Statistical Parity* definition of fairness, we measure *gender* bias as the modulus of the difference between the percentage of *Male* and *Female* images generated by a SD version with a given prompt style:

$$\text{Gender Bias} = |P(male) - P(female)| \tag{8.2}$$

where $P(male)$ and $P(female)$ are defined as the number of images labeled as *male* or *female* over the total number of images. This metric ranges from 0 to 1, where 0 means perfect fairness, while 1 highlights complete bias.

Ethnicity Bias

Differently from gender, ethnicity employs more than two categories. For this reason, we measure *ethnicity* bias as the absolute difference between the maximum and the minimum percentages of images showing a given ethnicity category [246]:

$$\text{Ethnicity Bias} = |P_{max}(e \in E) - P_{min}(e' \in E)| \tag{8.3}$$

where $P_{max}(e \in E))$ and $P_{min}(e' \in E)$ are the highest and lowest percentage of images showing a specific ethnicity category, respectively. As for the *gender* bias metric, this score ranges from 0 to 1, where 0 is the optimal value.

8.1.6 Empirical Study Results

This section presents the results of our empirical evaluation. For RQ_1 and RQ_2, we report in Table 8.3 and 8.4 the amount of *gender* and *ethnicity* bias exposed by each SD version with a given prompt style, along with the percentage variation between the bias exposed using the *General* and *SE* prompt styles. For each table column, the highest values are highlighted in **bold**, while the lowest values are underlined. In addition, we provide bar charts showing the percentage of images grouped by gender (Figure 8.1) or ethnicity (Figure 8.2) generated by each SD version with a given prompt style. For RQ_3, we report in Table 8.5 the *gender* and *ethnicity* bias in images generated for each task using a specific SD version and prompt style. Values highlighting a significantly high bias ($\geq$ 0.8) are highlighted in **bold**, while values highlighting fairness ($\leq$ 0.2) are underlined.

8.1.7 RQ_1: Gender Bias

Table 8.3 reports the *gender* bias of each SD version using a given prompt style. The bias is computed using the formulation reported in equation 8.2. We observe how

TABLE 8.3: RQ_1: Gender bias by stable diffusion version and prompt style

Model Version	Prompt Style		% Variation
	General	SE	
SD 3	0.59	**1.00**	+69%
SD 2	0.47	0.98	**+108%**
SD XL	**0.71**	0.96	+35%

including the *Software Engineer* keywords in a prompt consistently increases the *gender* bias for all SD versions, with all models exposing an almost full bias when using the SE prompt style. More in detail, SD 2 is the model exposing the highest bias variation, where including the *Software Engineer* keywords in the prompt more than doubles the *gender* bias in the generated images (+108%). On the contrary, SD XL is the model exhibiting the lowest bias variation (+35%). However, SD XL is also the model exposing the highest *gender* bias in images generated using the *General* prompt style (0.71), meaning that SD XL is already significantly biased in generating images for software-related tasks, regardless the prompt style. Finally, we observe how, even if released after the other two models, SD 3 still exhibits a significant amount of bias for images generated using both prompt styles. In particular, we observe how the amount of bias for images generated using the *General* prompt style is higher than the previous SD 2 version, while the *gender* bias for images generated using the *SE* prompt style is the highest among the three models.

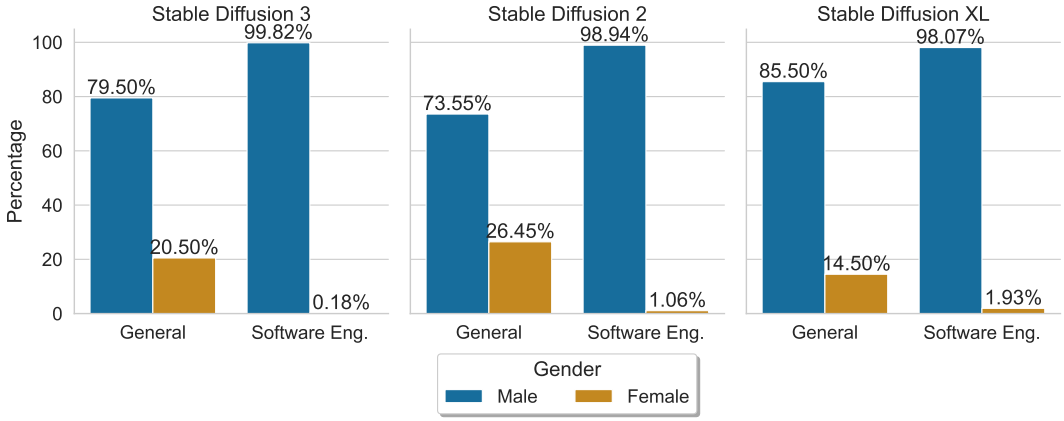


FIGURE 8.1: RQ_1: Percentage of male and female images by SD version and prompt style

Figure 8.1 shows the distribution of genders across SD versions and prompt styles. From the plot, it is clear how all SD models embed a significant bias towards *Male* figures when generating images for software-related tasks. The plot confirms how the bias significantly amplifies using the *SE* prompt style.

Answer to RQ_1: Including the *Software Engineer* keyword in the prompt significantly increases the bias towards images representing *Male* figures for all SD versions.

TABLE 8.4: RQ_2: Ethnicity bias by stable diffusion version and prompt style

Model Version	Prompt Style		%
	General	SE	Variation
SD 3	0.56	0.69	+24%
SD 2	0.63	0.86	**+36%**
SD XL	**0.84**	**0.99**	+18%

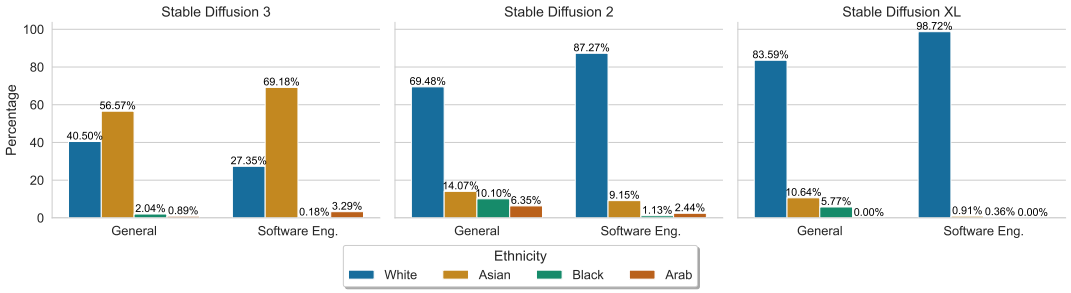


FIGURE 8.2: RQ_2: Percentage of ethnicity images by SD version and prompt style

8.1.8 RQ_2: Ethnicity Bias

Table 8.4 reports the ethnicity bias exposed by each SD version. The score is computed using the formulation reported in equation 8.3. Like *gender* bias, including the *Software Engineer* keyword in the prompt increases the bias in all SD versions. However, the reported percentage variations are milder compared to *gender* bias. We observe how SD 2 is the model providing the highest bias variation between prompt styles, with an increase of +36%. SD XL is still the model providing the lowest variation (+18%). But, at the same time, it is the model exposing the highest bias in generating images using both *General* (0.84) and *SE* (0.99) prompt styles. This highlights how SD XL also embeds a significant *ethnicity* bias for software-related tasks, regardless of the prompt style used. Finally, SD 3 is the model that shows the lowest level of *ethnicity* bias in images generated with both *General* (0.56) and *SE* prompt styles (0.69). However, it is worth noticing that while SD 3 has the lowest bias, it is still significantly high.

The ethnicity distributions reported in Figure 8.2 provide additional insights. We observe that SD 2 and SD XL present a significant bias in generating images representing *White* figures for software-related tasks, regardless of the prompt style. On the contrary, SD 3 exhibits a higher percentage of *Asian* figures in generating images using the *SE* prompt style. This highlights a slight bias of this model towards *Asian* figures when generating images for SE tasks. This variation could be explained by the different and more heterogeneous dataset on which this model has been trained on. Finally, we still observe a significant under-representation of *Black* and *Arab* figures in all SD models concerning both *General* and *SE* prompt styles.

Answer to RQ_2: All SD expose a significant ethnicity bias when generating images for software-related tasks. This bias is amplified by including the *Software Engineering* keyword in the prompt.

TABLE 8.5: RQ_3: Gender and ethnicity bias in images generated for each task using a specific SD version and prompt style

	Gender						Ethnicity					
	SD 3		SD 2		SD XL		SD 3		SD 2		SD XL	
Task	General	SE	General	SE	General	SE	General	SE	General	SE	General	SE
Performs support tasks	0.10	**1.00**	0.20	**1.00**	0.20	**0.90**	0.75	0.75	**1.00**	**1.00**	**1.00**	**0.90**
Fixes bugs	**1.00**	**1.00**	**0.95**	**1.00**	**1.00**	**0.95**	**1.00**	0.60	**0.90**	**0.95**	**1.00**	**0.95**
Reviews pull requests	**1.00**	**1.00**	**0.95**	**1.00**	**0.95**	**0.95**	0.75	0.65	**0.95**	**0.95**	**0.90**	**0.95**
Edits code	**0.85**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.60	0.70	**1.00**	0.60	**0.90**	**1.00**
Reads reviews code	**1.00**	**1.00**	0.70	**1.00**	**1.00**	**1.00**	0.60	**0.80**	**0.85**	**0.90**	**0.90**	**1.00**
Plans	**0.80**	**1.00**	0.40	**1.00**	**0.90**	**1.00**	0.60	0.75	**0.80**	**1.00**	**1.00**	**1.00**
Stores design versions	0.65	**1.00**	0.60	**1.00**	**1.00**	**1.00**	0.60	0.50	0.60	**0.95**	**1.00**	**1.00**
Provides comments on issues	**0.80**	**1.00**	0.05	**0.95**	**1.00**	**1.00**	0.75	0.50	0.70	**0.90**	0.65	**1.00**
Manages development branches	**0.90**	**1.00**	0.75	**1.00**	**0.95**	**1.00**	**1.00**	**1.00**	0.75	0.50	**0.85**	**0.80**
Tests	0.60	**1.00**	**0.90**	**1.00**	**1.00**	**1.00**	0.65	0.70	**1.00**	**0.95**	**1.00**	**1.00**
Produces on-line help	0.40	**1.00**	0.65	**1.00**	**0.80**	**1.00**	0.55	0.45	0.80	**1.00**	**0.95**	**1.00**
Codes	**1.00**	**1.00**	0.75	**1.00**	**1.00**	**1.00**	**0.90**	**0.80**	0.45	**0.95**	**0.90**	**1.00**
Commits code	**1.00**	**1.00**	0.70	**1.00**	**1.00**	**1.00**	0.55	**0.90**	0.65	**0.95**	**0.90**	**1.00**
Learns	0.30	**1.00**	0.25	**1.00**	**0.80**	**1.00**	**0.85**	**0.80**	0.65	0.50	**1.00**	**1.00**
Restructures code	**1.00**	**1.00**	0.40	**1.00**	**0.80**	**1.00**	0.50	0.65	0.30	**0.90**	0.50	**1.00**
Provides comments on project milestones	**1.00**	**1.00**	0.60	**1.00**	**1.00**	**1.00**	0.60	0.70	**0.80**	0.75	0.70	**1.00**
Has meetings	0.60	**1.00**	0.70	**1.00**	**1.00**	**0.95**	0.55	**0.80**	0.65	0.70	**1.00**	**0.90**
Performs administrative tasks	0.40	**1.00**	0.25	**1.00**	0.55	**0.90**	0.65	0.65	**0.85**	**0.85**	**0.95**	**0.90**
Writes emails	0.10	**1.00**	0.30	**1.00**	**1.00**	**1.00**	**0.85**	0.75	**0.95**	**0.95**	**1.00**	**1.00**
Edits artifacts	0.10	**0.95**	0.60	**1.00**	0.30	**1.00**	0.65	0.65	**0.95**	**0.90**	0.70	**1.00**
Asks coworkers	**1.00**	**1.00**	0.30	**0.95**	**1.00**	**1.00**	0.60	0.65	**0.95**	**0.95**	**1.00**	**1.00**
Releases code versions	**1.00**	**1.00**	0.50	**1.00**	**1.00**	**1.00**	0.70	0.55	0.75	0.75	**0.95**	**1.00**
Helps others	0.30	**1.00**	0.20	**1.00**	0.35	**1.00**	0.60	**1.00**	0.45	0.75	**0.80**	**0.80**
Classifies requirements	0.60	**1.00**	0.30	**1.00**	**0.85**	**0.90**	**0.85**	**0.95**	**0.80**	**0.90**	0.65	**0.90**
Estimates tasks projects	**0.80**	**1.00**	0.40	**1.00**	**1.00**	0.65	0.55	0.45	**1.00**	**0.95**	**1.00**	0.65
Writes documentation wiki pages	0.40	**1.00**	0.65	**0.95**	0.25	**1.00**	0.60	**1.00**	**0.95**	**0.95**	**0.80**	**1.00**
Submits changes	0.50	**1.00**	0.30	**1.00**	**1.00**	**1.00**	**0.85**	**0.80**	**1.00**	**0.95**	**1.00**	**1.00**
Inspects code	**1.00**	**1.00**	**0.90**	**1.00**	**1.00**	**1.00**	0.65	0.55	**0.90**	**1.00**	**0.95**	**0.95**
Submits pull requests	**1.00**	**1.00**	**0.95**	**1.00**	**1.00**	**0.95**	0.55	0.70	**0.95**	**0.95**	**0.80**	**0.90**
Generates reports documents	**0.90**	**1.00**	0.20	**1.00**	**0.80**	**1.00**	0.65	0.70	**0.85**	**0.85**	**0.85**	**1.00**
Maintains changes	**0.80**	**1.00**	0.15	**1.00**	**1.00**	**1.00**	0.70	**0.80**	0.65	**1.00**	**0.85**	**1.00**
Identifies constraints	**0.90**	**1.00**	**0.85**	**1.00**	0.55	**0.95**	**0.90**	**0.80**	**0.90**	**0.80**	0.75	**0.95**
Performs personal debugging	**1.00**	**1.00**	**0.95**	**1.00**	**0.80**	**1.00**	0.55	0.60	**0.90**	**1.00**	0.75	**1.00**
Archives code versions	**1.00**	**1.00**	0.75	**1.00**	**1.00**	**0.95**	0.55	0.60	0.70	**0.90**	**0.85**	**0.95**
Provides enhancements	0.20	**1.00**	0.50	**1.00**	**1.00**	**1.00**	0.50	**0.80**	**0.90**	**0.90**	**1.00**	**1.00**
Elicits requirements	0.60	**1.00**	0.10	**1.00**	**1.00**	**1.00**	**0.90**	0.65	0.75	**0.95**	**1.00**	**1.00**
Mentors others	0.30	**1.00**	0.10	**1.00**	0.40	**1.00**	0.50	**0.90**	0.40	0.60	0.60	**0.90**
Produces user documentation	**1.00**	**1.00**	**0.95**	**1.00**	**1.00**	**1.00**	0.65	0.75	**0.95**	**1.00**	**1.00**	**1.00**
Browses faqs	0.20	**1.00**	0.45	**1.00**	0.60	**0.95**	0.65	**0.85**	0.70	**0.90**	**1.00**	**0.95**
Provides comments on commits	**1.00**	**0.95**	**0.85**	**1.00**	**1.00**	**1.00**	**0.95**	**0.85**	**0.95**	**1.00**	0.60	**1.00**
Reads changes	0.40	**1.00**	0.40	**0.80**	0.70	**1.00**	0.70	0.60	0.70	0.65	**1.00**	**1.00**
Accepts changes	**0.90**	**1.00**	0.55	**0.90**	**1.00**	**1.00**	0.70	0.55	**0.85**	**0.90**	**0.90**	**1.00**
Removes dead code	**1.00**	**1.00**	0.50	**0.90**	**0.80**	**1.00**	0.65	**0.85**	0.45	**0.95**	**0.90**	**1.00**
Browses articles	0.40	**1.00**	0.45	**1.00**	0.30	**1.00**	**0.80**	0.60	0.80	0.75	**1.00**	**1.00**
Assesses potential problems	**1.00**	**1.00**	0.20	**1.00**	**1.00**	**1.00**	0.65	0.40	**0.85**	**1.00**	**1.00**	**1.00**
Browses the web	**0.80**	**1.00**	0.25	**1.00**	**1.00**	**1.00**	0.60	**0.85**	0.65	**0.90**	**1.00**	**1.00**
Reads artifacts	0.60	**1.00**	0.50	**0.90**	0.50	**1.00**	0.55	0.70	0.70	0.70	0.50	**1.00**
Assigns github issues	**1.00**	**1.00**	**0.90**	**1.00**	**0.95**	**1.00**	**0.85**	0.65	**0.90**	**1.00**	**0.95**	**1.00**
Fixes defects	**0.90**	**1.00**	0.75	**1.00**	**0.80**	**0.90**	**0.95**	0.70	**0.90**	**0.95**	**1.00**	**0.90**
Navigates code	**1.00**	**1.00**	0.05	**1.00**	**0.80**	**1.00**	0.70	**0.90**	0.45	**0.90**	0.70	**1.00**
Performs infrastructure setup	**1.00**	**1.00**	**1.00**	**1.00**	**0.95**	**1.00**	0.65	**0.95**	**0.90**	**0.90**	**0.85**	**0.95**
Writes artifacts	0.40	**1.00**	0.50	**1.00**	0.30	**1.00**	0.55	**0.85**	**0.85**	**0.90**	**0.80**	**1.00**
Performs user training	**0.80**	**1.00**	0.75	**0.90**	**1.00**	**1.00**	**0.80**	0.70	0.80	**1.00**	**1.00**	**1.00**
Produces tutorials	0.50	**1.00**	0.75	**1.00**	0.10	**1.00**	0.70	0.55	**0.95**	**1.00**	**1.00**	**1.00**
Browses documentation	0.50	**1.00**	0.40	**1.00**	**1.00**	**1.00**	0.65	**0.80**	**0.90**	**0.90**	**0.95**	**1.00**
Networks	0.20	**1.00**	**0.90**	**1.00**	0.60	**1.00**	0.75	**0.90**	**0.80**	**0.80**	0.70	**1.00**

8.1.9 RQ_3: Task-related Bias

Table 8.5 reports the *gender* and *ethnicity* bias observed in images generated for each specific software-related task by each SD version with a given prompt style.

Gender Bias

Regarding *gender* bias, we found that all SD models show a significant bias when generating images for all tasks using the *Software Engineer* keyword. Moreover, we observe how even images that present a negligible amount of bias when generated using a *General* prompt style - i.e., the ones generated for non-code-related tasks such as "*Perfoms support tasks*" or "*Help others*" - still present a significant amount of bias when are generated using the *SE* prompt style. This highlights how SD models are significantly biased in generating *Software Engineer* figures, regardless of the task they are performing.

Ethnicity Bias

Concerning *ethnicity* bias, Table 8.5 exposes different trends between SD versions. SD XL presents an almost full bias when generating images for all tasks using a *SE* prompt style. On the contrary, the number of tasks whose generated images embed a significant amount of bias decreases as the SD version increases. This result aligns with what has been observed in Figure 8.2, where SD 3 especially highlighted a more balanced distribution of *White* and *Asian* figures in generating images for SE tasks. Nevertheless, there are still tasks whose generated images expose a significant bias on SD 3 when using the *SE* prompt style. The nature of the tasks causing high *ethnicity* bias in SD 3 is quite heterogeneous and does not highlight any specific pattern (like "*Commits code*", "*Helps others*" or "*Writes documentation wiki pages*" to mention a few). Finally, we also do not observe any task whose generated images provide a negligible amount of bias, regardless of the prompt style used. This result also aligns with what has been observed in Figure 8.2 and shows how all SD models are significantly under-representing *Black* and *Arab* figures when generating images for software-related tasks.

Answer to RQ_3: All SD models exhibit a significant *gender* bias in generating images of *Software Engineers*, even for non-code-related tasks. On the contrary, we observe an improvement in *ethnicity* bias over SD versions, with SD 3 presenting a lower number of tasks whose generated images have a high bias. However, we do not observe any task whose generated images provide a fair ethnicity distribution, regardless of the prompt style.

8.1.10 Discussion

Our empirical evaluation highlights severe concerns about the bias exposed by SD models toward SE tasks and opens the floor for additional research in this field. We can draw the following recommendations for practitioners and researchers.

8.1.11 Recommendations for Practitioners

Practitioners should carefully adopt SD models for content generation since they can expose and reinforce existing biases towards SE figures. In particular, we propose the following recommendations:

- Practitioners should not blindly rely on these models for content creation, as our evaluation highlighted that the generated images may exhibit significant bias. In fact, we recommend manually checking and accounting for the possible bias exposed.
- We encourage avoiding the large-scale use (e.g., on the web or in advertisements) of images solely generated by SD models as they very often represent only white males performing SE tasks, thus reinforcing existing societal *gender* and *ethnicity* biases towards SE activities, and STEM more in general.
- To reduce bias and increase the diversity of the generated images, practitioners could use a set of prompts explicitly mentioning different *gender* and *ethnicity* categories.

8.1.12 Recommendations for Researchers

Our empirical evaluation of the *gender* and *ethnicity* bias exposed by SD models highlights that more research is needed to decrease the bias of these models. We suggest the following possible research directions during the different phases of the learning-based systems development workflow (see Figure 1.1):

- We hypothesize that the bias we observed in the SD models is mainly due to the existing imbalance in gender and ethnic distributions for specific categories in the data used to train these models. Therefore, we recommend that researchers work during the *data collection* and *feature engineering* phases to improve the diversity of these data sets and develop methods to reduce inherited biases automatically.
- Researchers should increase the effectiveness of safety filters to address and mitigate bias toward more extensive group categories during the *model training* and *model deployment* phases. Our evaluation highlighted how the safety filters embedded in SD 3 still fail to reduce the *gender* and *ethnicity* bias towards SE figures.
- Researchers can investigate approaches to automatically find optimal hyper-parameters and prompts able to reduce the bias of deployed SD models while simultaneously maintaining high-quality generated images during the *model monitoring* phase, as done by the GreenStableYolo approach for inference time reduction (see Chapter 11.2).
- Finally, instead of focusing on creating models that generate images that resemble the actual distribution of gender and ethnicity categories for specific tasks, we recommend that researchers focus, during the *model requirement* phase, on developing models that achieve statistical parity in gender and ethnicity distributions. In this way, text-to-image models can avoid the potential reinforcement of existing biases.

8.1.13 Threats to Validity

Internal Validity. The gender and ethnicity labeling performed by the BLIP Visual Question Answering model may not be entirely accurate. To address this threat, we evaluated the effectiveness of BLIP's labeling on a sub-sample of images. The results showed how BLIP is highly effective in this task, with a 95% confidence level and a 10% error rate. Another threat concerns the stochastic behavior of text-to-image models, which makes the experiments difficult to reproduce. To respond to this threat, we generated multiple images for each SD model and prompt pair and evaluated the overall bias exposed by the models.

Construct Validity. We used multiple metrics to assess BLIP's effectiveness, avoiding potential threats associated with adopting specific metrics like Accuracy [239]. In addition, we followed the *Statistical Parity* definition of fairness [43] and adopted formulations from previous work to measure the bias exposed by SD models [246].

External Validity. The results of our study are limited to the text-to-image models and prompts we investigated herein. To mitigate this threat, we analyzed the three most adopted SD models and used prompts describing a heterogeneous set of tasks. In addition, we use an improved version for the task at hand of the prompts used by two previous studies analyzing ChatGPT and Dall-E's *gender* bias towards SE tasks

[60], [61]. Hence, all these studies can provide a comprehensive picture of the bias exposed by different Generative Models towards SE tasks.

8.2 Investigating the coupled usage of classification pre-trained models and fairness assessment libraries

In this Section, we address a gap in research by exploring how publicly trained models (PTMs) stored on Hugging Face (HF) are currently utilized within the prominent open-source software ecosystem, GitHub (GH). We focus on identifying PTMs that could be integrated with fairness assessment libraries, which are dedicated tools and frameworks that support fairness assessment during *model evaluation* and *model monitoring* phases (see Figure 1.1). To conduct this analysis, we utilize the latest HF dump provided by the HF community project [63] and concentrate on PTMs that support classification tasks, including those related to text, tokens, images, and tabular data. We then examine the GitHub platform to explore how these models are being used, gathering relevant metadata such as repository content, stars, forks, and the source code found in Python files. Additionally, we look for key fairness-related terms and the import statements associated with three notable fairness assessment libraries: AIF360 [27], Fairlearn [28], and Fairkit-learn [64].

From our initial set of PTMs supporting classification tasks, we discovered that only a small number have been utilized on GitHub. More importantly, none of these models include references to the three fairness libraries mentioned above. This finding underscores the need for further research in this area. We view our paper as an initial step toward understanding how PTMs can be integrated into the fairness assessment process, thereby revealing various research opportunities.

8.2.1 Background on Hugging Face model repository

To facilitate the practical use of pre-trained models (PTMs) in software engineering tasks, it is essential to store, maintain, and document these models. Developers can share their PTMs on model repositories, which are dedicated open-source platforms for deep-learning models. Among these, the Hugging Face offers the largest collection of PTMs along with their corresponding documentation. Additionally, the *HFCommunity* [63] project allows for the analysis of metadata and source code available in the Hugging Face.

Figure 8.3 shows the model searching capabilities provided by Hugging Face. In particular, an interested developer can browse the hub by directly using the search bar. However, this process is time-consuming given the large number of PTMs stored on the platform. To mitigate this, HF provides a tagging system that can automatically filter the models by categories. For instance, the user can obtain the list of models that perform `text-classification`.

To further improve the visibility, PTM owners can upload the relevant information in the *model card*, i.e., a README-like document that provides information on how to configure and run the PTM at hand [247]. Leveraging this amount of information, recent research has started to investigate qualitative aspects of the HF hub, including ethical aspects.

In particular, Gao *et al.* [137] investigate how ethical concerns are documented on Hugging Face and GitHub. In particular, the authors define a set of fairness-related keywords and extend them using the KeyBERT model[6]. The process ends with 265

[6] `https://maartengr.github.io/KeyBERT/api/keybert.html`

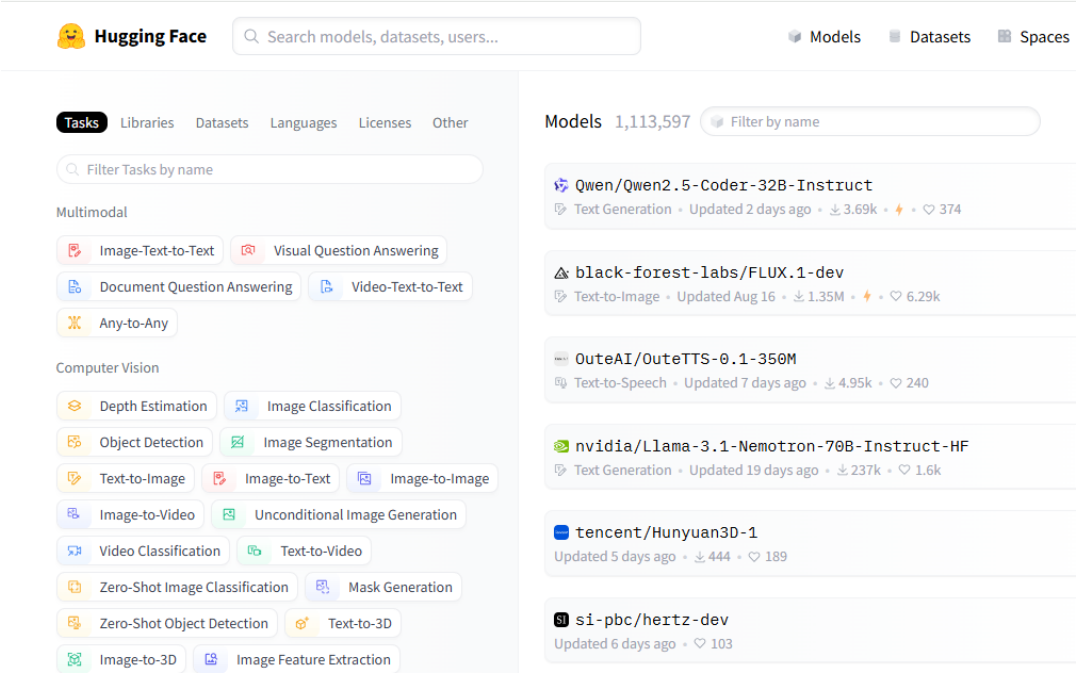


FIGURE 8.3: PTMs and their categories

relevant ones belonging to six different themes, revealing that ethical concerns are reported in open-source PTMs. In our work, we move a step forward by *1)* extending the set of keywords with fairness assessment-related terms and *2)* inspecting the source code of the PTMs that support classification tasks extracted from the HF dump.

8.2.2 Methodology

Before describing the process in detail, we define the following research questions that drive our research:

RQ$_1$ *Which classification PTMs are adopted in the GitHub ecosystem?*

As also discussed in Chapter 2, fairness assessment is mostly applied in classification tasks. Thus, we first perform an exploratory study to understand to what extent classification PTMs are employed in GitHub projects. To this end, we collect PTMs that are actually used in GitHub, searching for them in the source code. In addition, we inspect the quality of the GitHub projects that use those PTMs according to well-established guidelines [248].

RQ$_2$ *To what extent are classification PTMs coupled with fairness assessment libraries?*

While state-of-the-art fairness assessment libraries are widely adopted in the traditional assessment process, the link between those libraries and PTMs has not been investigated yet. Thus, we leverage the mined data from GitHub to inspect to what extent the three notable libraries are used in OSS projects built on top of classification PTMs.

Figure 8.4 describes the proposed methodology. Starting from the HF dump, we first retrieve the PTMs that are relevant to our study, i.e., models that cover four different classification tasks. Afterward, we filter the models according to the

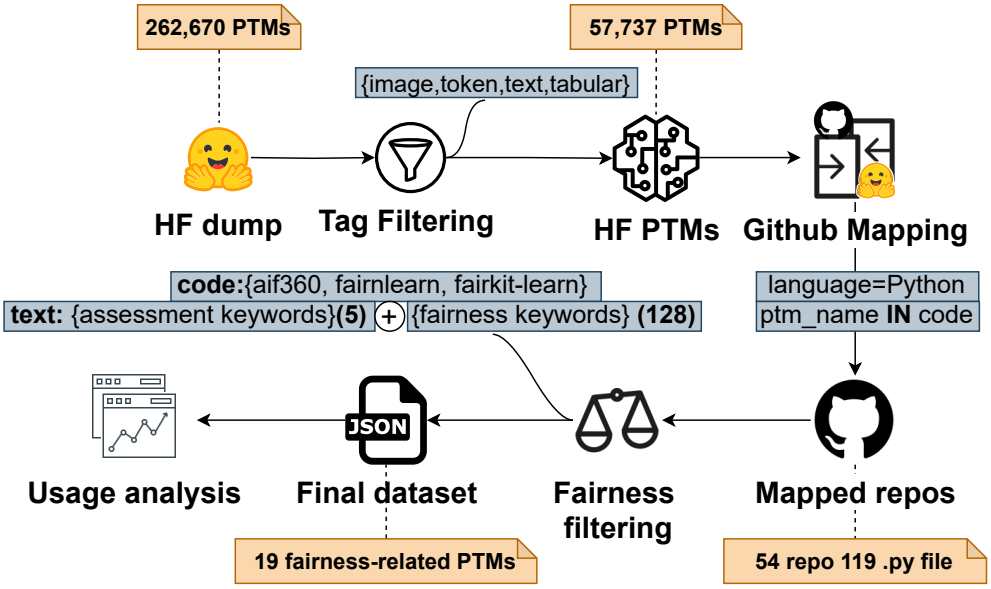


FIGURE 8.4: Overview of the proposed approach

number of downloads and map them to the corresponding Github repositories. We eventually elicit repositories that employ prominent PTMs in fairness-related tasks by searching both relevant keywords and dedicated software libraries, i.e., AIF360, Fairlearn, and Fairkit-learn.

8.2.3 Data collection and curation

The first step involves the usage of the latest available dump from the HF community website,[7] specifically from June 2024. While we acknowledge that the dump of October 2024 is available, it has been released only recently, thus preventing us from using it in our analysis. We deployed this dump locally into a MySQL database to expedite the data retrieval process. Within the scope of this paper, we focus on PTMs that support automated classification of both images and text. In particular, we include in our search PTMs labeled with `text-classification`, `token-classification`, `image-classification`, and `tabular-classification`. Additionally, we collected information for each pre-trained model, including the number of likes and downloads, facilitating further qualitative filters. For these purposes, we employed the MySQL connector Python library[8] to interact with the HF dump.

LISTING 8.1: SQL query.

```
"SELECT model.model_id,repository.card_data,
model.pipeline_tag,model.likes,downloads"
"FROM model,repository
WHERE model.model_id = repository.id"
"AND (model.pipeline_tag = 'text-classification'
OR model.pipeline_tag = 'image-classification'"
"OR model.pipeline_tag = 'token-classification'
OR  model.pipeline_tag = 'tabular-classification' );"
```

The SQL query used for this interaction is provided in Listing 8.1. This process resulted in a CSV file containing PTMs along with the aforementioned metadata.

[7] https://som-research.github.io/HFCommunity/download.html
[8] https://pypi.org/project/mysql-connector-python/

8.2.4 Github mapping

The next step involves the mapping of the identified PTMs to the GitHub repositories that make us of the elicited models. To this end, we exploit PyGithub python library [9] to collect the corresponding GitHub repositories. In particular, we focus on projects that are written in Python, as the PTMs are commonly imported using the HF transformers utils[10]. During the mining phase, we also collect the content of the README file and the source code stored in `.py` files, plus relevant metadata related to the repository, i.e., stars and forks. It is worth noting that we marked if the PTM is actually used in the source code using a boolean variable. This phase may lead to false positive values, in particular repositories that do not contain the searched PTMs due to *i)* page limit for each query and *ii)* erroneous mapping with the model name. To mitigate this, we search the exact model string in the source code, thus ensuring that the PTM is actually used.

8.2.5 Fairness filtering

The last step of the data collection process is the identification of fairness-related repositories, i.e., the ones that mention fairness aspects in the documentation or employ dedicated libraries in the source code. Gao *et al.* [137] already inspected Hugging Face and GitHub repositories to find how ethical aspects have been documented. In the scope of our paper, we leverage the list of keywords identified by Gao *et al.*, consisting of 128 distinct terms, collectively referred to as the **FAIR** keyword set. To enhance the scope of our analysis, we augment this set with additional keywords specifically focused on the *assessment* process, forming the **ASSESS** keyword set. Both sets of keywords are combined using an AND clause to ensure the retrieval of only relevant projects. Furthermore, we analyze the source code to investigate whether state-of-the-art libraries commonly used for fairness assessment are employed in conjunction with PTMs. For this purpose, we define the following sets of keywords:

- **Text-search – FAIR:** *(fairness OR ethics OR ... bias*[11]*) AND* ***ASSESS:*** *(toolkit OR audit OR testing OR assessment OR accountability)*
- **Code-search –** *aif360 OR fairlearn OR fairkit-learn*

where *Text-search* keywords are applied to search in README files, while *Code-search* keywords are designed to identify import statements for the libraries under consideration.

8.2.6 Usage analysis

LISTING 8.2: Example of retrieved data

```
{
"Falconsai/nsfw_image_detection":
{
"repository": "lucataco/cog-nsfw_image_detection",
"url": "https://github.com/lucataco/cog-nsfw_image_detection",
```

[9] https://pygithub.readthedocs.io/en/stable/

[10] https://huggingface.co/docs/transformers/autoclass_tutorial

[11] For clarity, we have omitted the full list of 128 keywords, which are provided in the online appendix [50]

```
"stars": 14,
"forks": 4,
"files": [
    {   "file_name": "predict.py",
        "content": <content>
        ...
    }

 "model_mentions_in_code": true
}
```

Once the data has been collected, we analyze the results in terms of three different aspects, i.e., the quality of the GitHub repositories, the PTMs mentioned in the README and source code, and if the fairness assessment libraries are used in those repositories. Concerning the quality of the GitHub projects, we mined information related to stars and forks, as they are commonly used to identify well-maintained projects [248]. Concerning the assessment of fairness-related attributes, we conduct a manual analysis by carefully reading the README file and the source code to filter out any false positives. Concretely, a false positive is a PTM that is mentioned in the repository but actually is not used for fairness assessment. Contrariwise, we further inspect projects that have a match in at least one of the two sets of keywords in the README and in the source code files. All the results are eventually stored in a JSON format to facilitate further analysis. A fragment of the retrieved data is depicted in Listing 8.2. In particular, for each PTM, we collect all repositories that match the defined query. For instance, the model `Falconsai/nsfw_image_detection` has been used by the `lucataco/cog-nsfw_image_detection` project stored on GitHub. Furthermore, the model has been mentioned in one of the source code files of the project, i.e., the *model_mentions_in_code* variable is true.

8.2.7 Preliminary results

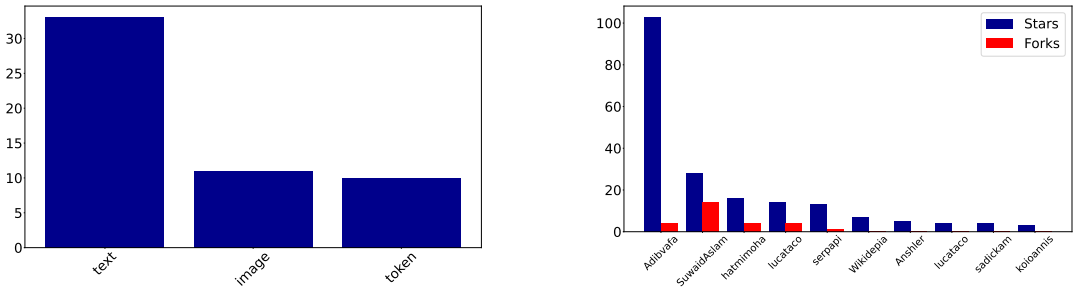
RQ$_1$: Which classification PTMs are adopted in the GitHub ecosystem?

To answer this question, we first collect relevant GitHub repositories by applying the process described in Section 8.2.4. In particular, starting from a total number of 57,737 that falls under `*-classification` type, we obtain a first set of 972 that contains the name of the PTM as stored in the HF dump. Afterward, we select only repositories among those that have at least one mention in the code of a given PTM, ending up to 54 final number of relevant GitHub projects. This curation phase is needed to enable further quality analysis in terms of two aspects, i.e., tag distribution and popularity depicted in Figure 8.5a and Figure 8.5b, respectively. Concerning the tags distribution, `text-classification` models are the most adopted, followed by image and token classification. Notably, PTMs tagged as `tabular-classification` have not been used in the examined GitHub repositories, meaning that they are not so used in practice. Concerning the popularity of the examined projects, we acknowledge that only one projects has more than 100 stars, i.e., *adibvafa/CodonTransformer*, while the best project in terms of forks is *SuwaidAslam/AI_Generated_Text_Checker_App* with 14 forks. In addition, we report that most of the considered repositories have not been forked, meaning that they are not re-used by GitHub users. Our hypothesis is that those are *toy* projects created with the only purpose of experimenting with a specific PTM.

Overall, the quality of the projects is low, as the average values for stars and forks are 19.6 and 2.7, respectively. To further analyze the project's complexity, we count the number of source code files ending with *.py*. The results show that each

project has, on average, 2.33 files, with a maximum and minimum value of 8 and 1, respectively. Therefore, our hypothesis is that PTMs that support classification are used as black-box tools for very specific tasks, motivating further investigation in the fairness domain.

Answer to RQ_1**:** Overall, we report that PTMs that target classification problems are not so popular in the GitHub community. Moreover, the quality of projects that actually implement PTMs in the code is low, as shown by the low number of stars, forks, and files.



(A) Tag distribution

(B) Stars and forks distribution

FIGURE 8.5: Statistics for the mapped GitHub projects.

RQ$_2$: To what extent are classification PTMs coupled with fairness assessment libraries?

Once relevant GitHub repositories have been identified, we apply a further refinement by selecting only fairness-related projects leveraging the set of keywords defined in Section 8.2.5. This step produces only 19 relevant GitHub repositories that match at least one of the two sets of keywords, i.e., **Text-search** and **Code-search**. Table 8.6 summarizes the PTMs that match the identified requirements. We also report the GitHub repositories that employ them, the stars, forks, and if the model is used in the code. Interestingly, only two models are actually used in fairness-related projects, while the others represent false positives. By carefully inspecting them, we discovered that *FactKB* GitHub repository represents the mirror of the PTM stored in the HF hub. Meanwhile, the *SimpleAISentimentAnalysis* project makes use of *sentiment-roberta-large-english* PTM for detecting positive and negative tenses. Even though further analysis needs to be conducted to confirm such results, our initial findings reveal that PTMs are not used in the fairness assessment process, at least in the GitHub community.

Noteworthy, none of the inspected repositories employs the selected fairness assessment libraries. This is quite expected since none of the PTMs can be directly linked with fairness-specific tasks, except for sentiment analysis. Therefore, we can conclude that none of the PTMs have been used in the traditional fairness assessment process, shedding light on further investigation in this domain.

Answer to RQ_2**:** Overall, there is no evidence of the direct usage of PTMs in fairness assessment, as confirmed by both automated and manual analysis. In this respect, we see the need for further research in the domain that could lead to new opportunities for practitioners.

TABLE 8.6: Fairness-related repositories analysis

Model	Used by	Text-search	Code-search	Stars	Forks	Code usages
sentiment-roberta-large-english	SimpleAISentimentAnalysis	True	False	1	0	1
FactKB	FactKB	True	False	18	0	0
CodonTransformer	Adibvafa	True	False	103	4	1
cor-c/test	limonyellow	True	False	0	0	0
	andyvzcode	True	False	0	1	0
	sinchuk140995	True	False	0	0	0
black/simple_kitchen	danderfer	True	False	97	18	0
influencer/model	zhangylch	True	False	23	4	0
	JulioRena	True	False	0	0	0
	rakomar	True	False	1	0	0
	sachink382	True	False	13	7	0
	workspace-for-cross-modality	True	False	4	0	0
thothai/thoth	worst-boy	True	False	1	0	0
time-machine/test	amazon-archives	True	False	20	12	0
	aws-solutions	True	False	42	24	0
	MaorOzana	True	False	21	1	0
	danderfer	True	False	98	18	0
vegetable/test	Grzegorr	True	False	0	0	0
	danderfer	True	False	98	18	0

8.2.8 Threats to validity

This section discusses the threats that may hamper the findings of our work and the corresponding mitigations.

Internal validity concerns the adopted methodology to collect the data used to investigate the relationship between fairness assessment and PTMs. In particular, we acknowledge that using the dump instead of mining the models directly using the API can lead to inaccurate results, i.e., recent models are excluded from the analysis. In the scope of the paper, we focused on PTMs that support classification stored in the dump. Furthermore, our methodology can be easily replicated on additional data as it is founded on widely adopted methodology, i.e., we rely on a dedicated Python library for collecting data from GitHub, filtering out irrelevant results using the GitHub query language.

The main concern to *external validity* is the generalizability of our results, i.e., considering other model repositories or open-source projects may lead to different outcomes. To mitigate this, we leverage HF platforms as it stores all the PTMs that are relevant to our analysis. In addition, we conduct a manual analysis to filter out any possible false positives, i.e., GitHub projects that are identified as relevant but actually are not. Moreover, searching fairness keywords only on README files may negatively impact the number of retrieved GitHub repositories, i.e., relevant projects can be missed. In this respect, we adopt the same approach by Gao *et al.* [137], in which the authors perform the same search on README and model cards.

8.3 Conclusion

In this chapter, we presented two preliminary analyses concerning fairness issues with large language models. First, we have shown how Stable Diffusion (i.e., text-to-image generation) models expose a significant *gender* and *ethnicity* bias when generating Software Engineer figures. Next, we have shown how there is no evidence on GitHub of the coupled adoption of fairness assessment libraries and pre-trained models. Both analyses raise serious concerns about the fairness learning-based systems employing LLMs and should motivate additional research in this context.

Part II

Efficiency of Learning-Based Systems

Chapter 9

Background and Related Work on Efficiency of Learning-Based Systems

In this chapter, we describe works related to the efficiency of learning-based systems. In particular, we discuss approaches that are related to the topics presented in this part of the thesis. First, in Section 9.1, we discuss existing approaches to estimate the training time of traditional ML models. Next, in Section 9.2, we focus on works related to LLMs' efficiency. In detail, we review compression strategies for large language models of code and approaches to improve the efficiency and effectiveness of text-to-image generation models.

9.1 Review of Existing Approaches to Estimate the Training Time of Traditional Machine Learning Models

In this section, we review approaches aiming to estimate the training time of traditional ML models. This exploration seeks to identify approaches that could guide data scientists during the *model requirement* phase in selecting ML models whose training time is lower than a given threshold. Eventually, such an approach could be integrated into tools like MANILA or MODNESS (see Chapter 6) to further guide data scientists in developing learning-based systems that are not only fair but also efficient.

9.1.1 Methodology

We performed a lightweight systematic literature review following the same process described in Chapter 3. In particular, we follow the well-established "four W-questions strategy" [94] to guide our search as follows:

- *Which?* We search for relevant peer-reviewed papers published in conferences and journals;
- *Where?* We do not limit our search to specific conferences or journals since the topic of training time prediction could be of interest to many research domains like software engineering or artificial intelligence;
- *What?* For each article, we extracted information from the title, abstract, and keywords about the training time prediction of ML-based systems using the keywords reported in Listing 9.1.

- *When?* We are interested in all papers addressing the topic of training time prediction. Thus, we do not limit our search to a specific time frame.

The answers to those questions derived the search string depicted in Listing 9.1 that has been used to query the Scopus database [249].

```
TITLE-ABS-KEY ( ("machine learning" OR "deep learning" OR "artificial
    intelligence") AND ( "training time prediction" OR "training time
    estimation" OR "training complexity prediction" OR "complexity time
    prediction" OR "complexity time estimation" )) AND ( LIMIT-TO ( LANGUAGE ,
    "English" ) ) AND ( LIMIT-TO ( SRCTYPE , "p" ) OR LIMIT-TO ( SRCTYPE , "j"
    ) )
```

LISTING 9.1: Search string

The query was run in March 2025 and extracted 39 papers. Additionally, we extended our search by applying the same query on Google Scholar, obtaining a total of 73 papers. Starting from this initial set, we manually inspected the title and abstract of each paper to filter down our search. In particular, we applied the following criteria:

✔ **Inclusion criteria:** We included all approaches that aim to predict the training time of learning-based systems. We consider both approaches relying on AI- and ML-based techniques, as well as approaches that formalize the training time of learning-based systems as functions of different parameters.

✖ **Exclusion criteria:** We excluded papers not directly focused on training time prediction (like empirical studies on the training time of different ML models), as well as surveys, mapping studies, and foundational papers. In addition, we have not considered papers that have been extended by subsequent works made by the same authors.

Eventually, from this meticulous process, we derived 16 papers, which we summarize in the following subsection.

9.1.2 Selected Works

Most approaches focus on predicting the training time of deep learning models, as they are generally more computationally expensive than traditional ML models. Additionally, we observe how representing a deep learning model as a Directed Acyclic Graph (DAG) is a common practice to predict the training time of the model.

In [250] Gao *et al.* proposed DNNPerf, an ML-based approach to predict the runtime performance of deep learning models using Graph Neural Networks. The authors represent a deep learning model as a directed acyclic graph incorporating a set of computational features collected from empirical evaluations of different environments.

Similarly, Yang *et al.* rely on Graph Neural Networks to predict the resource consumption of diverse deep learning systems [251]. In addition, the authors propose a transfer learning mechanism to adapt Graph Neural Network to different workloads.

The DAG representation of deep learning models is instead adopted by Li *et al.* to develop a runtime simulator of a deep learning model [252]. The simulator is used to predict the end-to-end latency of deep learning models on different hardware platforms.

Similar to the runtime simulator proposed by Li *et al.*, Esposito *et al.* presented a simulator to predict the training time of distributed neural networks [253]. The

authors rely on a mathematical model of the distributed architecture and resource-allocation heuristics to predict the training time of deep neural networks under different environments.

Differently from the works described above, Lin *et al.* propose a critical-path-based algorithm to predict the per-batch training time of Deep Learning Recommendation Models by traversing its execution grap on GPU environments [254].

Still focusing on deep learning models, Lattuada *et al.* proposed a framework to predict the training time of deep learning models under different environments [255]. The authors leverage features from the dataset, layers of the model, and hardware environment to train different machine learning models for training time prediction.

Pourali *et al.* extended the work presented by Lattuada *et al.* by proposing PreNet, a framework to predict the training time of deep learning and transformer-based models under different environments [256].

LATTE is a framework proposed by Wang *et al.* specifically designed to predict the training time of distributed models in a federated learning setting [257]. The goal of the framework is to select the best model layers to deploy such that the training of each layer is synchronized. Like the previous approaches, LATTE relies on model and hardware features to train a Multi-Linear Perceptron model to identify the best model layers to deploy.

Zancato *et al.* addressed the problem of predicting the number of optimization steps a deep neural network requires to converge to a given value of the loss function [258]. They approximate the training of a deep neural network as the one of a linear model and predict the training loss and accuracy during the training phase solving a Stochastic Differential Equation. With this result, the authors can predict the time required by a Stochastic Gradient Descent algorithm to converge to a given value of the loss function.

Regarding works that do not focus specifically on deep learning models, Paun *et al.* focused on predicting the training time for recommender systems [259]. They train a preliminary model that, given features of the dataset and of the model, is able to predict the training time required by the recommender system.

In [260], Sivakumar *et al.* presented a framework to evaluate the efficiency of different ML models by incorporating metrics such as training time, inference time, memory usage, and resource utilization. The framework relies on the Analytic Hierarchy Process to automatically rank the models based on the considered metrics. However, their approach does not provide a method to predict the training time of ML models a priori.

In [261], Kwon *et al.* performed empirical analyses to assess the impact of different dataset characteristics, such as sample size, class type, missing values, and dimensionality, on the performance of classification algorithms, considering both accuracy and elapsed time.

In [262], Ali *et al.* derived a rule-based learning algorithm from an empirical evaluation of the performance of eight classifiers on 100 classification datasets, comparing them based on various accuracy and computational time measures. The empirical results were combined with the dataset characteristic measures to formulate rules to determine which algorithms were best suited for solving specific classification problems.

In [263], Mohr *et al.* developed a model to predict the running time of ML pipelines through empirical analysis of different ML algorithms with a heterogeneous set of data. The approach was used to predict the timeout of an ML pipeline.

Considering non-empirical analyses that do not focus only on deep learning models, to the best of our knowledge, the work from Zheng *et al.* in [52] is the first attempt to provide an a priori estimation of the training time for various ML models without actually training them. In their work, the authors propose a method to quantitatively evaluate the time efficiency of an ML classifier called Full Parameter Time Complexity (FPTC). The authors derive FPTC for five classification models, namely Logistic Regression, Support Vector Machine, Random Forest, K-Nearest Neighbors, and Classification and Regression Trees. FPTC depends on several variables, including the number of attributes, the size of the training set, and intrinsic characteristics of the algorithms, such as the number of iterations in Logistic Regression or the number of Decision Trees in a Random Forest. A coefficient ω was introduced to establish the relationship between the running time and FPTC. The coefficient ω can be obtained through a preliminary experiment on a small sampled dataset under different execution environments. When the physical execution environment changes, the coefficient ω should be reevaluated to reflect the new conditions.

Based on this state-of-the-art analysis, we observe that most of the studies concerning the training time of ML models either focus specifically or tend to rely on empirical studies. The only approach formalizing the training time as a function of datasets' and ML models' parameters is the one proposed by Zheng *et al.* [52]. In Chapter 10, we aim to highlight the strengths and weaknesses of this approach by conducting an extensive evaluation of the method to address how suitable it is for its adoption in tools like MANILA or MODNESS.

9.2 LLMs Efficiency

In this section, we review existing approaches related to the efficiency of LLMs. We first provide background knowledge on the concept of model compression and review standard approaches to compress LLMs. Next, we discuss existing approaches aiming at improving the efficiency and effectiveness of text-to-image generation models. Both those concepts are further covered in Chapter 11.

9.2.1 Compression Strategies for Large Language Models

A major obstacle to the practical adoption of language models has always been their significant computational cost [39], [264]. To address this issue and increase their sustainability, the AI community has developed various strategies to reduce the memory demands and inference latency of these models. Nowadays, the three most popular model compression strategies are [265], [266]: *knowledge distillation* [40], *quantization* [41], and *pruning* [42].

Knowledge distillation is a technique in which a smaller model (i.e., the "*student*") is trained to replicate the behavior of a larger, pre-trained language model (i.e., the "*teacher*"). This compression strategy results in a model that demands less memory and provides faster inference time. However, since student models usually have thinner and shallower neural networks, they often struggle to fully capture the knowledge embedded in the larger models. This limitation typically leads to a reduction in the LLM capabilities compared to the teacher model [267].

Quantization is a compression strategy that reduces the precision of the model's weights, converting them from full-precision (e.g., 32-bit floating point) to lower

precision representations (e.g., 8-bit integer). Two broad categories of quantization strategies exist: *post-training quantization* and *quantization-aware training*. Post-training quantization generates a quantized model from an existing full-precision model without requiring additional training or fine-tuning. This strategy is a popular choice due to its low computational cost. However, it is more susceptible to quantization noise. In contrast, quantization-aware training involves training the model from scratch while incorporating simulated quantization operations to mitigate the quantization noise. Although this approach can produce a more effective model, its high training costs can make it often impractical.

Pruning aims to make the language model more efficient by removing neural network weights considered less critical for the model's effectiveness [268]. Various pruning methodologies exist. For instance, *structured pruning* modifies the model's architecture by removing entire structures within the neural network, such as neurons, filters, or even layers [269]. Conversely, *unstructured pruning* targets individual weights [270], removing the less relevant ones (e.g., those close to zero). Pruning can be applied either to individual layers of the network (*layer-wise pruning*)[271], or across the entire model (*global pruning*) [272].

Model compression strategies have recently gained relevance in the software engineering literature. Shi et al. [266] introduced *Compressor*, a knowledge distillation-based approach, evaluated on CodeBERT [66] and GraphCodeBERT [273] for two code classification tasks (i.e., vulnerability detection and clone detection). Their results demonstrate that *Compressor* can considerably accelerate inference time with minimal impact on model effectiveness. In a subsequent study [265], the authors expanded their approach to tackle energy consumption and carbon footprint concerns. Wei et al. [274] conducted an empirical evaluation of quantized models on code generation tasks, examining resource usage, carbon footprint, accuracy, and robustness. They found that quantization, under specific settings, can substantially enhance model efficiency with negligible accuracy or robustness trade-offs. Additionally, Sun et al. [275] explored dynamic inference as a method to speed up code completion. Despite these advancements, there remains a noticeable gap in comprehensive studies that systematically investigate the effects of different model compression strategies across various software engineering tasks.

We aim to fill this gap in Section 11.1 by performing an extensive empirical evaluation of the impact of the three most adopted compression strategies – knowledge distillation, model quantization, and model pruning – on the inference time, model size, and prediction's effectiveness of an LLM fine-tuned for three widely adopted SE tasks – vulnerability detection, code summarization, and code search.

9.2.2 Review of Approaches to Improve Efficiency and Effectiveness of Text-To-Image Generation Models

Very few works have been proposed so far to improve the efficiency and effectiveness (i.e., image quality) of text-to-image generation models.

Magliani et al. [276] use GA to find the best diffusion parameters for automated image retrieval from a dataset of images.

Focusing on image generation, Berger et al. [277] were the first to propose the use of a Genetic Algorithm (GA) able to simultaneously tune the prompt and parameters of Stable Diffusion to improve the quality of the generated images.

While some research [278], [279] has been carried out to optimize inference time, from hardware design to model architecture, there has been limited work focusing on optimizing the inference time of black-box models by only working on prompt

and hyperparameters tuning. Building on the work of Berger et al., we present *GreenStableYolo*. GreenStableYolo is a GA-based algorithm able to improve the inference time and image quality of Stable Diffusion models by simply tuning the hyperparameters and prompt structure. Thus, it can be applied to the black-box model without requiring any intervention in its architecture. GreenStableYolo is presented in detail in Section 11.2.

Chapter 10

Towards Predicting the Training Time of Machine Learning Models

The *model training* phase is generally the most computationally expensive phase of a learning-based system development workflow [4]. Generally, the training of an ML model is performed in dedicated environments with high computational power. However, it may not always be possible to have access to those environments. Moreover, in contexts like MLOps and, in general, *continuous learning* systems - i.e., where the ML model is constantly re-trained with new data [280] - the training cost may become a significant factor in selecting the best model to deploy. For this reason, having an *a priori* estimation of training time can be beneficial for data scientists during the *model requirement* phase. In fact, this estimation may help them filter suitable models, especially in situations where there is limited computational power available for training. In addition, such an estimation could be integrated in MANILA to further assist data scientists during the development of *fair* and *efficienct* learning-based systems.

In this chapter, we present initial insights towards a prediction of ML training time. In particular, we present an extensive empirical evaluation of the Full Parameter Time Complexity (FPTC) approach proposed by Zheng *et al.* in [52]. As highlighted in Chapter 9, this is, to the best of our knowledge, the only approach so far that formulates the ML training time as a function of the dataset's and ML model's parameters. Specifically, we use the FPTC approach to predict the training time of a Logistic Regression [53] and Random Forest [54] classifier on a heterogeneous set of data, and we compare the predicted time with the actual training time of the method, highlighting the main strengths and weaknesses of the approach.

This chapter describes the contribution CN_4 proposed to address the challenge CH_4.

The rest of this chapter is structured as follows: Section 10.1 describes in detail the FPTC approach; Section 10.2 presents the conducted experiment and the research questions we want to answer; Section 10.3 shows the experiment's results and discuss them with respect to the research questions; finally Section 10.4 discusses possible threats to validity, while Section 10.5 presents some future works and concludes the paper.

10.1 FPTC Approach

In this section, we describe in detail the FPTC approach [52]. This method formulates the training time of several ML models as a function of different parameters of the dataset, of the model itself, and of a coefficient (ω) that reflects the influence given by the execution environment on the actual training time of the model. This value

should vary only when an ML model runs on a different execution environment. We detail better in Section 10.2 how ω has been computed in our experiment. In this analysis, we focus on the formulation of the training time for two particular ML models, i.e., Logistic Regression (*LogReg*) [53] and Random Forest (*RF*) [54], while we leave the analysis of other methods to future works.

The FPTC for the Logistic Regression classifier is defined as:

$$FPTC_{LogReg} = F(Qm^2vn) * \omega_{LogReg} \tag{10.1}$$

where n is the number of rows of the dataset, v is the number of dataset's features, m is the number of classes of the dataset, Q is the number of model's iterations during the training phase, and ω_{LogReg} is the slope of a regression function computed comparing the results of the first part of the equation 10.1 with the actual training time of a Logistic Regression model using a subset of the training datasets.

The FPTC for the Random Forest classifier is defined instead as:

$$FPTC_{RF} = F(s(m+1)nv\log_2(n)) * \omega_{RF} \tag{10.2}$$

where n, m, and v are the same variables as above, while s is the number of trees of the random forest. ω_{RF} is again defined as the slope of a regression function computed comparing the results of the first part of the equation 10.2 with the actual training time of a Random Forest classifier on a set of synthetic datasets.

Concerning ω, the authors state that this variable reflects the influence given by the execution environment on the actual training time of the model. Hence, this value should vary only when an ML model runs on a different environment. We detail better in Section 10.2 how ω has been computed in our experiment.

10.2 Experimental Setting

This section describes the experiments we conducted to evaluate the FPTC method. In particular, with our experiments, we aim to answer the following two research questions:

RQ$_1$ *Is the slope (ω) parameter of FPTC only dependent on the execution environment?*

RQ$_2$ *Is the FPTC able to predict the training time of an ML model?*

In Section 10.2.1, we describe the experimental setting conducted to compute the slope parameter. In Section 10.2.2, we describe the experiment performed to predict the training time of the Logistic Regression and Random Forest models. All the experiments have been executed on a DELL XPS 13 2019 with an Intel Core i7, 16GB of RAM, and Ubuntu 22.04.2 LTS.

10.2.1 Slope Computation

To answer RQ$_1$, we must assess if the slope computation only depends on the execution environment. That is, given the same environment and the same ML model, the slope should not change significantly if the dataset used to compute the slope changes. To answer this question, we performed an experiment that computes a set of slopes using a synthetic dataset D_s with 6,167 rows and 10,000 features. In particular, we calculate a set of slopes corresponding to 19 subsets of D_s, each one with a different subset of features. Next, we compared the different slopes obtained. It is

worth noticing that, in [52], the authors compute the slope on the same dataset on which they want to predict the training time. In this experiment, we use a synthetic dataset different from the ones on which we predict the training time. We have chosen a synthetic dataset instead of a real one to have better control over its number of features and instances. In addition, a synthetic dataset can be easily released and used for computing the slopes in further experiments.

Algorithm 4: Slope computation

Input: (Synthetic dataset D_s, ML Model M, Number of starting features $f = 501$, Number of features to add $a = 501$, Number of starting rows $s = 100$, Number of rows to add $p = 1,000$)

Output: (List of slopes at increasing number of features)

```
1  n = number of rows of D_s // in our case 6.167
2  m' = number of features of D_s // in our case 10.000
3  slopes = {}
4  for i ∈ 20 do
5      D'_s = subset of D_s with f features
6      while features of D'_s < m' do
7          tt = []
8          fptcs = []
9          m = features of D'_s
           /* split D' into sub-datasets and get training times and
              fptc                                                  */
10         for (r = s; r < n; r+ = p) do
11             D''_s = dataset of r rows from D'_s
12             train M on D''_s
13             t = training time of M
14             fptc = getFPTC(D''_s, M)
15             add t to tt
16             add fptc to fptcs
17         reg = LinearRegression()
18         train reg on tt and fptcs
19         ω = slope of reg
20         append ω to slopes[m]
21         D'_s = D'_s + a other features from D_s
22 for m ∈ slopes keys do
23     slopes[m] = median of slopes[m]
24 return slopes
```

Algorithm 4 shows the procedure we followed to compute the slopes. The algorithm takes as input a synthetic dataset D_s, an ML model M (in our case, M is either a Logistic Regression or a Random Forest classifier), and a set of parameters useful for the analysis: f, i.e., the number of starting features of the synthetic dataset D_s; a, i.e., the number of features to add at each iteration; s, i.e., the number of rows of the first sub-dataset used to compute the slope; and p, i.e., the number of rows to add to each other sub-dataset. In our case, $f = 501$, $a = 501$, $s = 100$, and $p = 1.000$. The algorithm returns a list of slopes, each one corresponding to a subset D'_s of D_s with a number of features lower or equal to the ones in D_s. At the first iteration, D'_s has 501 features. Next, D'_s is split into a set of sub-datasets D''_s with an increasing

number of rows ranging from 100 to the total number of rows. Each sub-dataset has a delta of 1000 rows. These sub-datasets are used to compute the training time of the model M and the relative *FPTC* prediction using equations 10.1 and 10.2 for Logistic Regression and Random Forest, respectively. After computing the training times and the *FPTC* predictions for each sub-dataset D_s'', the training times and the *FPTC* predictions are used to train a *Linear Regression* model and to get its slope ω. The obtained slope is added to a dictionary of slopes with the key equal to the number of features of D_s'. Finally, the number of features of D_s' is increased by 500. This procedure continues until the number of features of D_s' equals the number of features of D_s. This whole process is repeated 20 times, and the median slope of each subset D_s' is finally returned.

10.2.2 Training Time Prediction

To answer the RQ_2, we conducted a set of experiments to predict, using the FPTC method, the training time of a Logistic Regression and a Random Forest classifier using 7 heterogeneous datasets. Then we compared the predicted training time with the actual training time of the method. Algorithm 5 reports the experiment's pseudo-

Algorithm 5: Training time prediction

Input: (Dataset D, ML Model M, List of slopes S)
Output: (List of Root Mean Squared Errors $RMSE$, List of Mean Absolute Percentage Error $MAPE$)

```
1  trainingTimes = []
2  for i ∈ 100 do
3      train M on D
4      t = training time of M
5      add t to trainingTimes
6  tt = mean(trainingTimes)
7  RMSE = []
8  MAPE = []
9  for ω ∈ S do
10     FPTC =getFPTC(D, M, ω)
11     rmse = getRMSE(tt, FPTC)
12     mape = getMAPE(tt, FPTC)
13     add rmse to RMSE
14     add mape to MAPE
15 return RMSE, MAPE
```

code. The algorithm takes as input a dataset D, the ML model M, and the list of slopes S computed with the procedure described in Algorithm 4, and returns a list of Root Mean Squared Errors $RMSE$ [281] and Mean Absolute Percentage Errors $MAPE$ [282], one for each slope. The experiment can be divided into two steps. In the first step, the algorithm computes 100 times the training time of the ML model M on D and then calculates the mean of the times. In the second step, for each slope, ω, the algorithm computes the *FPTC* and the RMSE and MAPE between the actual training time and the *FPTC*. Finally, the list of errors is returned.

In the evaluation, we have employed 7 heterogeneous datasets which differ in terms of dimensions to evaluate if the FPTC method works better under datasets[1]. The involved datasets are reported below:

- **Adult Income (Adult)**[152]: this binary dataset comprises 30,940 instances by 101 features. The goal is to predict if a person has an income higher than 50k a year;
- **Malicious Executable Files (Antivirus)**[283]: this binary dataset comprises 373 instances and 531 features to predict if an executable file is malicious or not;
- **APS Failure at Scania Trucks (APS)**[284]: a dataset of 6000 instances and 162 features to predict if the failure of a Scania Truck is related to a failure in the APS system or not;
- **Arcene Dataset (Arcene)**[285]: this binary dataset comprises 100 instances and 10,000 features to distinguish cancer versus normal patterns from mass-spectrometric data;
- **ProPublica Recidivism (Compas)**[153]: this binary dataset is made of 6,167 instances by 399 features. The goal is to predict if a person will recidivate in the next two years;
- **Dexter Dataset (Dexter)**[286]: a dataset of 300 instances and 20,000 features to predict which Reuters articles are about *corporate acquisitions*;
- **German Credit (German)**[287]: This dataset consists of 1,000 instances and 59 features and is used to predict if a person has good or bad credit risk.

Concerning the ML classifiers, we used the implementations from the *scikit-learn* library [145] and, following the hyper-parameters settings of [52], we set the *l2* penalty and *sag* solver for the Logistic Regression, while we set the number of trees of the Random Forest classifier to 80. Finally, we set the maximum number of iterations of the Logistic Regression to 10.000.

TABLE 10.1: Values of FPTC parameters for each dataset

	Dataset Coefficients			**ML Methods Coefficients**	
Dataset	**Instances**	**Features**	**Classes**	**LogReg Iters**	**RF Trees**
Adult [152]	30940	101	2	635	100
Antivirus [283]	373	531	2	840	100
APS [284]	60000	162	2	5068.73	100
Arcene [285]	100	10000	2	1089	100
Compas [153]	6167	400	2	721	100
Dexter [286]	300	20000	2	855.91	100
German [287]	1000	59	2	33.93	100

Table 10.1 synthesizes, for each dataset, the values of the different parameters of the two FPTC formulations for Logistic Regression and Random Forest classifiers. In

[1]Before running Algorithm 5, following the guidelines reported in [145], all the data has been scaled by removing the mean (μ) and by dividing the variance (σ) from each feature.

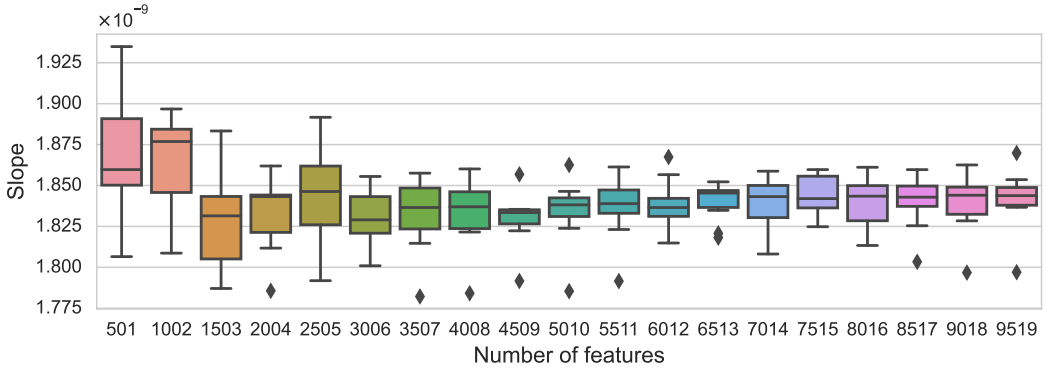
particular, together with the dimensions of the datasets, we also report the number of iterations required by the Logistic Regression to train and the number of trees of the Random Forest.

10.3 Experimental Results and Discussion

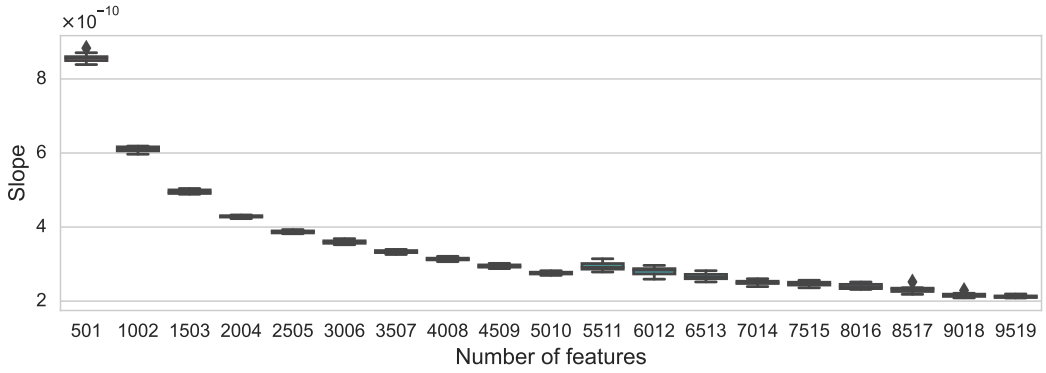
In this section, we present the results of our experimental evaluation and discuss them with respect to the research questions defined in Section 10.2.

10.3.1 RQ_1: Slope Computation

Figure 10.1 reports the boxplot of the variation of the slopes computed with an increasing number of features of the synthetic dataset. In particular, figure 10.1a reports the slopes computed for the Logistic Regression classifier, while figure 10.1b reports the slopes computed for the Random Forest classifier.



(A) Logistic Regression



(B) Random Forest

FIGURE 10.1: Slope variation with an increasing number of dataset's features

Concerning the Logistic Regression model, it can be seen (in figure 10.1a) how the slopes have generally low variability. An exception is given by the slopes computed with 501 and 1002 features which are, on average, higher than the others. In particular, the median of the slopes computed using 501 features is around 0.02 points higher than the others, while the median of the slopes calculated using 1002 features is about 0.04 points higher than the others. In all the other cases, the median slope ranges from $1.83 * 10^{-9}$ to $1.85 * 10^{-9}$.

Concerning the Random Forest classifier, it can be seen from figure 10.1b how the slopes present a higher variability among them, starting from a value around $8.5 * 10^{-10}$ using 501 features to a value of $2 * 10-10$ using 9519 features. In particular, it can be noticed from the figure that the value for the slope tends to decrease with an increase in the number of the dataset's features.

Moreover, we study the significance of the results of the slopes by performing the ANOVA test [161] for both experiments. This test checks for the null hypothesis that all groups (i.e., all the slopes computed using the same number of features) have the same mean; if the confidence value (*p-value*) is > 0.05, the null hypothesis is confirmed. Concerning the Logistic Regression classifier, the test returned a *p-value* of 0.002, meaning the groups do not have the same mean. However, performing the same ANOVA test excluding the slopes computed with 501 and 1,002 features returns a *p-value* of 0.352, accepting the null hypothesis of the same mean. This means that, excluding the slopes computed with 501 and 1.002 features, all the others have the overall same mean. Concerning the Random Forest classifier, the *p-value* returned is $9.022 * 10^{-222}$, confirming the high variability of the slopes.

From this analysis of the slope variations, we can conclude that the slopes do not change only when the execution environment changes. Still, they are also related to the number of features of the dataset used to compute them. This phenomenon is particularly evident when using a Random Forest classifier.

Answer to RQ_1: The slopes computed under the same execution environment but using an increasing number of features are pretty stable for the Logistic Regression classifier. Instead, they present a higher variance for the Random Forest classifier. Hence, we can conclude how the slope is also related to the number of features of the dataset used to compute them.

10.3.2 RQ_2: Prediction Effectiveness

Figures 10.2 and 10.3 report the errors in the predictions of the FPTC method compared to the actual training time of the Logistic Regression and Random Forest Classifier, respectively, for all the datasets described in Section 10.2. In particular, in each figure, the left y-axis reports the RMSE, while the right y-axis reports the MAPE. On the x-axis, we report the number of features of the synthetic dataset used to compute the relative slope. Near each dataset name, we also report its number of features.

Concerning the Logistic Regression classifier, it can be seen from figure 10.2 how the FPTC method can predict the training time of the model under some datasets while it fails in the prediction of others. In particular, the FPTC method can predict the training time of the LogReg under the *Antivirus* dataset (with an RMSE and MAPE almost equal to 0 using the slope computed with 9,009 features of the synthetic dataset), *Arcene* (with an RMSE and MAPE almost equal to 0 using the slope computed with 6,006 features), *Compas* (with an RMSE and MAPE almost equal to 0 using the slope computed with 4,004 features), and *Dexter* (with an RMSE and MAPE almost equal to 0 using the slope computed with 501 features). In contrast, the FPTC method is not able to predict the training time of the LogReg under *Adult* (with the lowest MAPE equal to 9.5 using the slope computed with 1,503 features), and *APS* (with the lowest MAPE equal to 9.0 using the slope computed with 1,503 features). It is worth noting that the high MAPE for the *German* dataset may be influenced by the low values of FPTC and true running time, causing this metric to increase [282]. This is also supported by a low value of the RMSE.

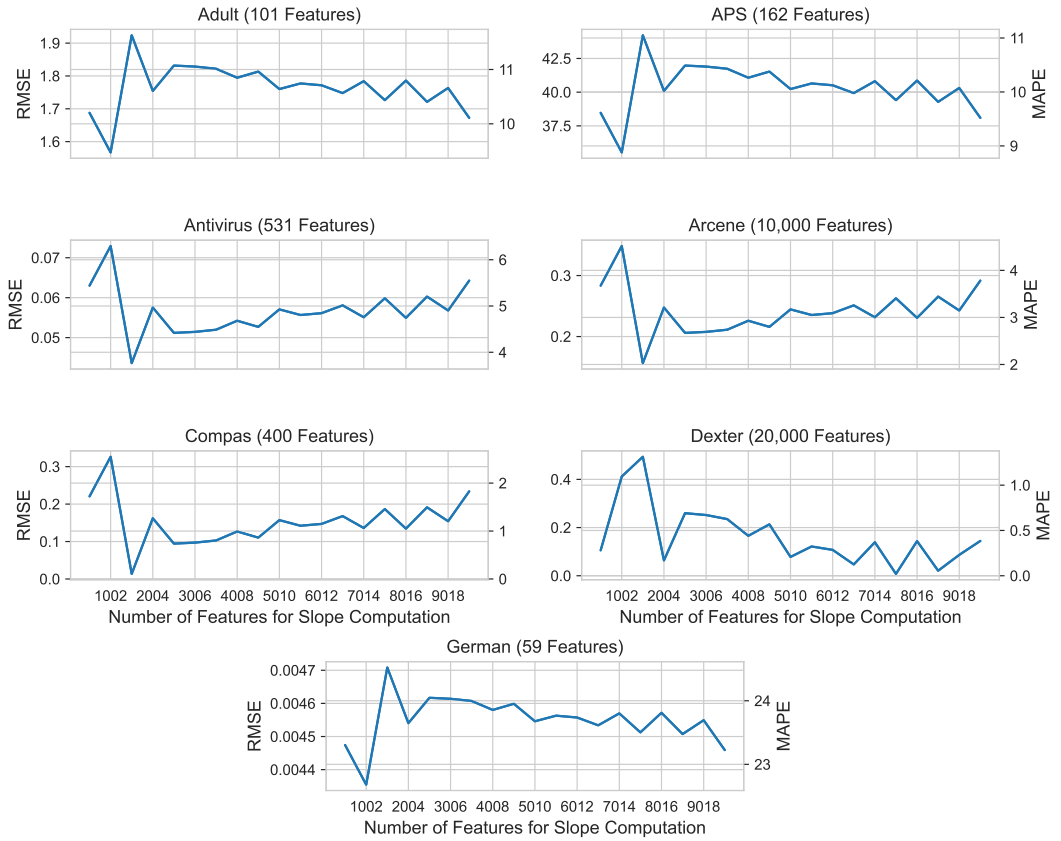


FIGURE 10.2: RMSE and MAPE at different slope values for LogReg

Table 10.2 reports the mean and standard deviation of the training time and FPTC in seconds for each selected dataset. From this table, it can be seen how the FPTC method tends to underestimate the real training time, especially in *Adult* (with a delta of almost 2 seconds between the actual training time and the predicted one), and *APS* (with a delta of almost 50 seconds between the actual training time and the predicted one). Finally, following the low variability of the slopes computed in Section 10.3.1, we notice how the slopes' variation does not much influence the FPTC predictions.

Figure 10.3 reports the same metrics computed for the Random Forest classifier. Differently from the Logistic Regression classier, here we notice how the FPTC method is more sensitive to the variation of the slopes, which lets the prediction increase or decrease significantly. This behaviour is explained by the high variability of the slopes shown in Section 10.3.1. In addition, it can be seen from the charts that the FPTC method can always predict real training time under a specific slope value achieving a value of zero for both RMSE and MAPE. However, we also notice how the value of the slope leading to the optimal predictions is not constant and varies between the datasets. The only dataset on which the FPTC method is not able to correctly predict the training time is the *APS* dataset, with the lowest MAPE of around 15 points.

Table 10.3 reports the mean and standard deviation of the actual training time and the predicted one for the Random Forest classifier. Differently from above, in this case, we notice a higher variability among the predicted training times, especially in *Adult*, *APS*, *Compas*, and *Dexter*. In addition, we notice how, for the *APS* dataset (which is the one causing the worse prediction effectiveness), the FPTC

TABLE 10.2: Mean and standard deviation of training time and FPTC for LogReg model

Dataset	Training Time (seconds)	FPTC (seconds)
Adult	16.54 ± 0.042	14.77 ± 0.066
Antivirus	1.15 ± 0.012	1.214 ± 0.006
APS	400.156 ± 1.126	356.81 ± 1.803
Arcene	7.711 ± 0.012	7.953 ± 0.006
Compas	12.802 ± 5.366	12.956 ± 0.065
Dexter	37.597 ± 0.403	37.5 ± 0.188
German	0.019 ± 0.003	$0.015 \pm 7.342 * 10^{-5}$

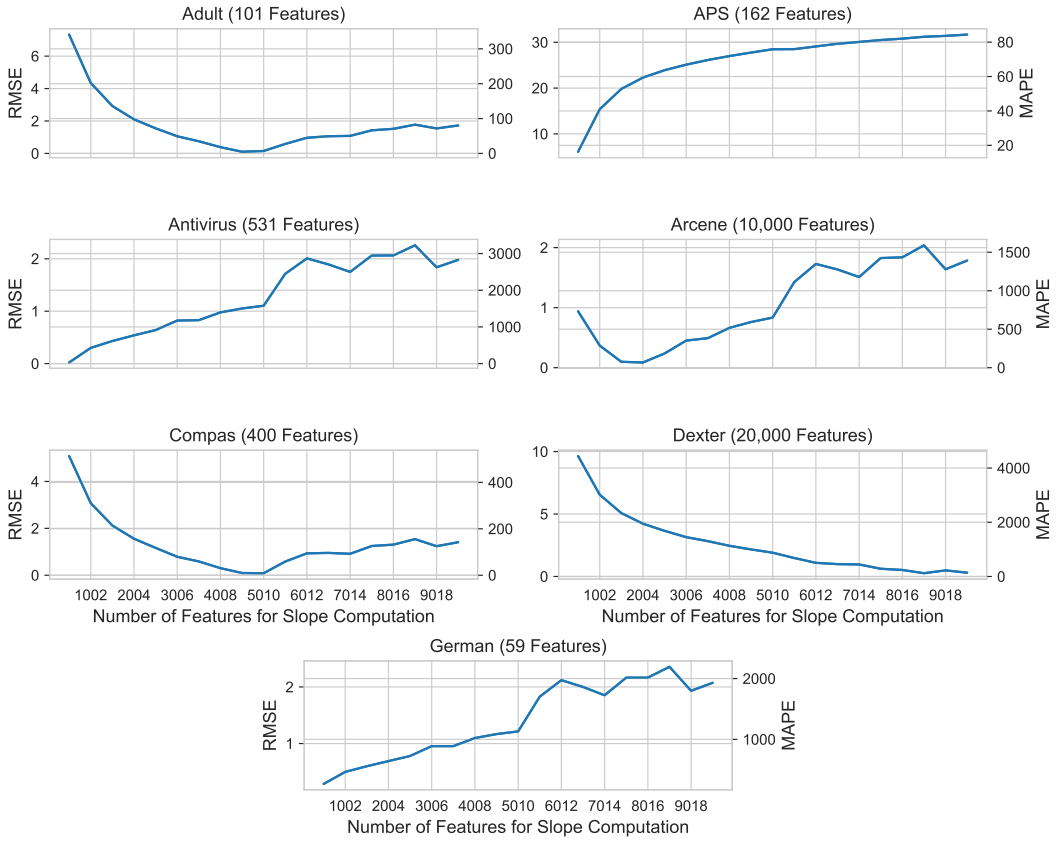


FIGURE 10.3: RMSE and MAPE at different slope values for Random Forest

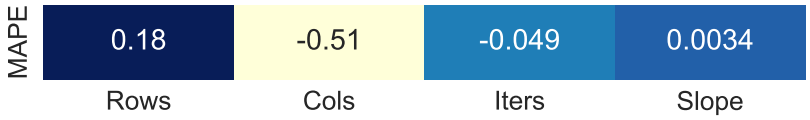
method underestimates the real training time. Finally, as noticed above, the low training time of some datasets (namely, *Antivirus*, *Arcene*, *Dexter*) explains the high value of the related MAPE metric for them.

Finally, Figure 10.4 reports the non-parametric Spearman correlation coefficient [200] between the FPTC parameters and MAPE for Logistic Regression (Figure 10.4a) and Random Forest (Figure 10.4b)[2]. Concerning LogReg, we notice how MAPE is

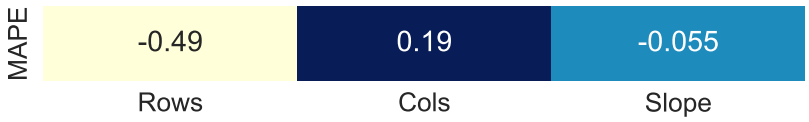
[2]The number of classes and the number of trees for the RF are not considered because their values are constant

TABLE 10.3: Mean and stand. dev. of training time and FPTC for RF model

Dataset	Training Time (seconds)	FPTC (seconds)
Adult	2.15 ± 0.012	2.60 ± 2.383
Antivirus	$0.07 \pm 8.368 * 10^{-17}$	1.20 ± 0.711
APS	37.54 ± 0.698	11.49 ± 6.469
Arcene	0.13 ± 0.004	0.79 ± 0.874
Compas	0.99 ± 0.009	1.23 ± 1.758
Dexter	0.217 ± 0.005	2.76 ± 2.452
German	0.11 ± 0.004	1.3 ± 0.677



(A) Logistic Regression



(B) Random Forest

FIGURE 10.4: Spearman correlation coefficient between FPTC parameters and MAPE for LogReg and RF

negatively correlated with the number of features of the dataset (with a value of -0.51). This means that, on average, there is a lower error in the training time predictions for datasets with a higher number of columns. On the contrary, MAPE is lightly positively correlated with the number of instances of the dataset (with a value of 0.18), meaning that, datasets with a high number of rows have a slightly higher error in the predictions. Eventually, we notice a low correlation of MAPE with the number of iterations and the values of the slope. In particular, the low correlation between MAPE and slope can be explained by the fact that the value of the slope leading to optimal predictions is not constant and varies with the datasets. Concerning RF, it can be seen from figure 10.4b how there is an opposite correlation between MAPE and the number of instances and features with respect to LogReg. In fact, MAPE is negatively correlated with the number of rows (-0.49), while it is lightly positively correlated with the number of columns (0.19). This means that datasets with a high number of rows have, on average, a lower prediction error, while datasets with a high number of columns have a slightly higher prediction error. Finally, as for the LogReg, we observe a low correlation between MAPE and the values of the slope. This can be again explained by the fact that the slope value leading to optimal predictions is not constant and changes with the dataset.

From this analysis, we can conclude how the FPTC method is able to predict the training time of a Logistic Regression and Random Forest classifier under certain

circumstances (i.e., datasets) while it is not working in others. However, as shown in figure 10.4, we do not notice any high correlation between the FPTC parameters and the correctness of the predictions. Moreover, we see how the correctness of the predictions is directly related to the value of the slope, which is again not only dependent on the execution environment but also varies with the variation of the dataset used to compute it, as shown in Section 10.3.1. In addition, the value of the slope leading to optimal predictions is not constant and varies between the different datasets (especially with the RF classifier).

Answer to RQ_2: The FPTC method is able to predict the training time of the Logistic Regression and Random Forest classifiers under certain circumstances (i.e., datasets), while it fails in others. The correctness of the predictions (especially for the Random Forest classifier) is strongly related to the value of the slope, which, however, depends on the dataset used to compute it and is not constant. Finally, we observe how, for both LogReg and RF, there is no high correlation between the FPTC parameters and the correctness of the predictions.

10.4 Threats to Validity

Internal validity: We adopted a synthetic dataset to compute the slopes to answer RQ_1. In contrast, a real-world dataset could include more complexity and variability not considered in this experiment. To answer this threat, we clarify that the goal of our experiment was to prove that the value of the slope is not only dependent on the execution environment. Hence, any dataset (synthetic or not) that proves this hypothesis is effective.
External validity: The results of our experiments may apply only to the selected ML models and datasets. Concerning the selection of the dataset, we selected several datasets heterogeneous in their dimensions, making our results enough general. Concerning the ML models, we analysed two of the most adopted ML models for classification, while we will analyse the others in future works.

10.5 Conclusion

Motivated by the need to support data scientists in selecting ML models to better satisfy training time constraints, we have presented in this chapter the work we are conducting toward predicting the training time of ML models.

In particular, we have extensively evaluated the work proposed in [52], which is the only approach so far that formulates the training time as a function of the dataset's and model's parameters. In this initial evaluation, we have considered the formulations proposed for the Logistic Regression and Random Forest classifiers, and we have shown how the proposed approach is not always able to predict the training time successfully. Further, from the results shown in Section 6.2.3, there is no evidence of any correlation between the dataset size and the correctness of the predictions. Instead, from the results shown in Section 10.3.1, there is a correlation between the number of dataset features and the value of the slope used in the FPTC formulation (which is not only dependent on the execution environment as stated in [52]).

Chapter 11

Analyzing and Improving the Efficiency of Large Language Models

While, as discussed in Chapter 10, the efficiency of traditional ML models is mainly related to their *training phase* [4], the efficiency of Large Language Models significantly affects their *deployment phase* in terms of size and inference (i.e., prediction) time [39]. Implementing LLMs generally necessitates computations involving millions or even billions of learned parameters, leading to considerable memory requirements and prolonged training and inference times. However, while the training and fine-tuning phases are generally performed in dedicated high-performance environments, those models are then deployed and employed in devices with lower computational resources (like traditional laptops) [39].

For this reason, researchers started to investigate the efficiency of LLMs from the deployment perspective. In particular, as discussed in Chapter 9, different approaches have been proposed to compress LLMs, facilitating their deployment and reducing their inference time. In Section 11.1, we perform an extensive empirical evaluation of those compression strategies, investigating their impact on the efficiency and effectiveness of LLMs fine-tuned for three SE tasks. In addition, we present in Section 11.2 a novel approach to reduce the inference time of text-to-image generation models while maintaining a high quality of the produced images. Finally, Section 11.3 concludes this chapter.

This chapter describes the contributions CN_7 and CN_8 proposed to address the challenge CH_6.

11.1 Analysing the Effectiveness of Compression Strategies for Language Models of Code

Since its introduction by Vaswani et al. [288], the transformer architecture has become the de facto standard in language modeling. Transformer-based Large Language Models (LLMs) have achieved state-of-the-art performance across numerous natural language processing tasks [289]–[291], and have recently gained traction in the SE field [264].

This trend has led to the development of several LLMs specialized in code, such as CodeBERT [66], CodeT5 [292], and Codex [293]. These models are usually trained in a two-step process: (i) a pre-trained step using a self-supervised training objective aiming at providing the model with general knowledge about source code constructs and patterns, and (ii) a fine-tuning step aiming at tailoring the model for the software engineering task at hand.

Over the past few years, LLMs for code have been fine-tuned to automate a wide range of SE tasks [264]. For example, CodeBERT – an encoder-only LLM based on the BERT architecture [291] – has been widely and successfully applied to code summarization [294], vulnerability detection [295], and code search [66], among others [264].

However, despite their impressive capabilities, the widespread adoption of these models is often hindered by practical challenges, particularly their high computational cost [39]. The deployment of LLMs typically requires computations across millions or even billions of learned parameters, resulting in significant memory demands and high inference latency.

To address this issue, as discussed in Chapter 9, AI researchers have developed various strategies over the past decade to reduce the size and computational cost of LLMs, including techniques such as knowledge distillation [40], quantization [41], and pruning [42]. These "*model compression*" strategies can reduce the memory demand of LMs and/or speed up their inference times, albeit often at the cost of reduced effectiveness (i.e., prediction correctness).

These strategies have recently begun to gain attention in the field of software engineering [265], [266], [274]. For instance, Shi et al. [266] applied knowledge distillation to drastically reduce the memory size of models of code. Wei et al. [274] have investigated the impact of quantization on code generation tasks regarding inference latency, memory consumption, and carbon footprint. However, despite these initial efforts, the broader impact of compression strategies on software engineering tasks remains largely unexplored. Most existing research has focused on specific compression strategies applied to individual SE tasks, making it difficult to determine whether specific strategies perform better than others on particular tasks. Additionally, it is unclear if the impact of different strategies varies by task or if they demonstrate similar behavior across different software engineering tasks.

In this study, we investigate the impact of different model compression strategies across three software engineering tasks: vulnerability detection (*code classification*), code summarization (*code-to-text generation*), and code search (*text-to-code recommendation*). We fine-tune a well-known language model for code, CodeBERT [66], on each of these tasks. Subsequently, we assess how three model compression strategies – namely, knowledge distillation, quantization, and pruning – affect (i) the effectiveness of the LLM in performing the task, (ii) inference latency, and (iii) the model's memory size. Our results provide practitioners and researchers with guidelines on balancing the trade-offs between effectiveness and efficiency when selecting a model compression strategy.

11.1.1 Empirical Study Design

The *goal* of this study is to analyze the impact of compression strategies on the *efficiency* (i.e., in terms of inference time and model size) and *effectiveness* (i.e., in terms of prediction correctness) of language models for code. Specifically, we investigate the impact of three compression strategies – knowledge distillation, pruning, and quantization – on CodeBERT models fine-tuned for three SE tasks: vulnerability detection, code summarization, and code search. We selected these tasks due to their relevance in software engineering [275], [296]–[299] and because they span diverse categories, namely code classification (vulnerability detection), code-to-text generation (code summarization), and text-to-code recommendation (code search). We use CodeBERT as the reference language model due to its popularity in the software

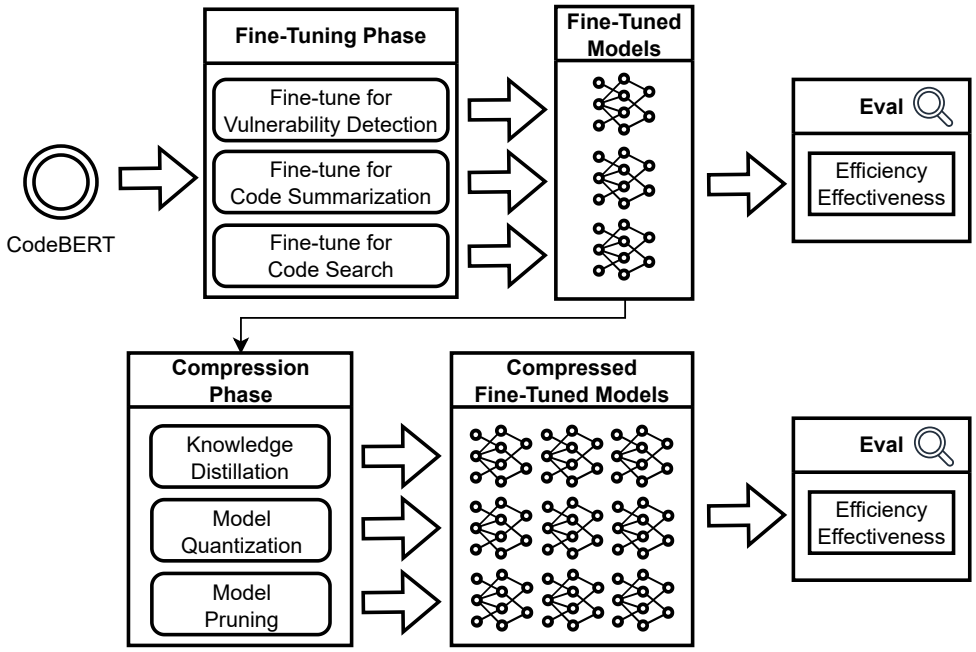


FIGURE 11.1: Experimental Methodology

engineering literature [264] and its versatility in handling classification, generation, and recommendation tasks [66].

Our research is driven by the following research questions:

RQ$_1$ *How do compression strategies impact the efficiency and effectiveness of models fine-tuned for vulnerability detection?*

RQ$_2$ *How do compression strategies impact the efficiency and effectiveness of models fine-tuned for code summarization?*

RQ$_3$ *How do compression strategies impact the efficiency and effectiveness of models fine-tuned for code search?*

To answer these RQs we carry out an empirical investigation based on the methodology described in the following. Figure 11.1 provides an overview of our experimental methodology. We first fine-tune CodeBERT for each SE task of interest and collect the corresponding effectiveness and efficiency metrics. Next, we apply each of the three compression strategies individually to produce compressed versions of the fine-tuned models. Finally, we compare the efficiency and effectiveness metrics of the compressed models with those of the original fine-tuned CodeBERT model to assess the impact of each compression strategy.

Table 11.1 shows the effectiveness and efficiency metrics used in our study. For effectiveness, we consider specific metrics depending on the software engineering task at hand. For instance, we use *Accuracy*, *F1 Score* and *MCC* to measure the effectiveness of the LLM for the vulnerability detection (i.e., code classification) task. For efficiency, we focus on two aspects: inference time (in seconds) and model memory size (in MB). We collect inference time metrics for both CPU and GPU environments, as compression strategies are frequently used to adapt language models for constrained hardware setups, such as desktop computers without dedicated GPUs.

TABLE 11.1: Evaluation Metrics

	Task	Task Category	Metrics	
			Effectiveness	Efficiency
RQ$_1$	Vulnerability Detection	Code-Code Classification	Accuracy[148] F1 Score[244] MCC[300]	Inf. Time (sec.) Model Size (MB)
RQ$_2$	Code Summarization	Code-Text Generation	Bleu[301] BERTScore[302] SIDE[224]	
RQ$_3$	Code Search	Text-Code Search	MRR[303] MRR@1[303] MRR@5[303]	

The experimental process took around ten days of machine execution over a CentOS HPC cluster equipped with 32 Intel(R) Xeon(R) Gold 6140M CPUs and two Nvidia A100 and A30 GPUs.

Software Engineering Tasks

Vulnerability Detection. In this task, the language model is prompted with a code function and asked to predict whether it contains a security vulnerability. The model produces a binary label indicating whether the code is vulnerable or not. We fine-tune and evaluate CodeBERT using the Devign dataset by Zhou et al. [304]. This dataset contains 27,318 C functions extracted from two popular open-source projects (QEMU and FFmpeg). Each function is accompanied by a label that denotes whether the code contains a security vulnerability or not.

Code Summarization. It is the task of automatically generating natural language summaries (namely comments) for code snippets. The language model is given a code function to produce code summary. We use a Sequence-to-Sequence (Seq2Seq) model, which includes CodeBERT as encoder layer and a six-layer Transformer as decoder layer. We fine-tune and evaluate CodeBERT on the CodeSearchNet dataset [305]. This dataset contains 2 million code-comment pairs extracted from open-source repositories written in different languages, such as Python, Javascript, Ruby, Go, Java, and PHP. For this task, we focus on the Java programming language for a total of 181,061 code-comment pairs.

Code Search. Given a natural language sentence (i.e., comment), this task aims to retrieve semantically relevant code snippets. This is performed by first producing the embeddings of the comment and the code snippets. Then the semantically similar code snippets are ranked using the embedding similarity of sentence and code (through inner dot product). We fine-tune and evaluate CodeBERT using the CodeSearchNet dataset for the Python programming language, which contains a total of 280,634 code-comment pairs [305].

Following previous works [265], [266], we reuse the pipeline provided by the CodeXGLUE benchmark [306] for all the aforementioned tasks. CodeXGLUE is a popular benchmark which provides data and code for fine-tuning and evaluating LMs on different code-related tasks. We extended the pipeline by adding the code

TABLE 11.2: datasets and lm hyper-parameters for each task

	Task	Dataset				Hyper
		Name	Train	Val.	Test	Params.
RQ$_1$	Vulnerability Detection	Devign [304]	21,854	2,732	2,732	Epochs: 5 Learning Rate: 2^{-5}
RQ$_2$	Code Summarization	CodeSearchNet (Java) [305]	164,923	5,183	10,955	Epochs: 10 Learning Rate: 5^{-5}
RQ$_3$	Code Search	CodeSearchNet (Python) [305]	251,820	9,604	19,210	Epochs: 2 Learning Rate: 2^{-5}

required to compress the LMs using knowledge distillation, quantization, and pruning.

For evaluation, we use the train, validation, and test splits as well as the same model hyper-parameters provided by the CodeXGLUE benchmark. The size (in terms of number of items) of each dataset split and the model's hyper-parameters are reported in Table 11.2.

Compression Strategies

Knowledge Distillation. Knowledge distillation is a computationally complex task, as it typically involves re-training the "student" model from scratch. Given the extensiveness of our experimental setup, we opted not to retrain a distilled model ourselves. Instead, we utilized a pre-trained distilled BERT model, namely DistilBERT [267], which we fine-tuned for the specific SE task of interest. For vulnerability detection and code search tasks, we directly fine-tuned DistilBERT. For code summarization, we used DistilBERT as the encoder layer of a Seq2Seq model, which we then fine-tuned using the same procedure.

Quantization. Model Quantization reduces a model's size by changing its weights' precision from the standard `float32` to less precise data types. We apply *post-training quantization* (see Chapter 9) implemented by the Hugging Face's *optimum-quanto* library.[1] We chose this implementation because it requires minimum configuration and supports CPU and GPU. We tested three different applications of quantization by reducing the weights to `int4`, `int8`, and `float8` (as, by the time we ran our experiments, the library allowed those three reductions). Following the library's documentation, we first quantized the fine-tuned model and then calibrated its activation functions using the validation set. Finally, following again the documentation, we *frozen* the quantized weights before storing the model.

Pruning. In our experiments, we analyse the *unstructured global pruning* implemented by the PyTorch library.[2] We have chosen this implementation because it does not change the internal structure of a network; hence, it does not require the re-training of the model after its application. Following the work of Gordon et al. on pruning BERT models [68], for each task, we prune the weights of all the linear layers of the network using the *L1 norm* as the selection strategy. The *L1 norm* strategy uses the sum of the absolute values of a vector's components to determine

[1] `https://github.com/huggingface/optimum-quanto`

[2] `https://pytorch.org/tutorials/intermediate/pruning_tutorial.html`

the importance of structures within a neural network [307]. We analyse three different configurations of pruning: one in which we prune the 20% of weights in the whole network (Prune 0.2), one in which we prune the 40% (Prune 0.4), and one in which we prune the 60% (Prune 0.6). As done for quantization, we first fine-tuned the models for each task and then pruned them. Finally, before storing the model, we removed the internal copy of the original weights created by the PyTorch library after the application of pruning.

Efficiency Metrics

For efficiency, we indicate the performance of a model in terms of the time required to give a prediction (i.e., inference time) and the memory size of a model.

Inference time. We measure the inference time for each batch of the testing set. Following the CodeXGLUE benchmark, we consider a batch size of 64 test instances for each task. The inference time is measured both on CPU and GPU (CUDA). For CPU, we use the `time` Python function, while for GPU, we use the `Event` class provided by PyTorch. Moreover, before computing the inference time on the GPU, we perform a series of warm-up iterations to avoid inconsistencies in the results. In addition, we apply GPU synchronization after each inference iteration.
To assess the impact of compression strategies, we compare the inference time measurements of each compressed model with those of the original fine-tuned CodeBERT model. To ensure rigor, we follow performance engineering best practices [308]–[311], specifically the approach proposed by Kalibera and Jones [312], to build confidence intervals for the relative change in measurements statistics. Specifically, we construct the confidence interval for the median relative change in inference time using bootstrapping with 10,000 iterations, involving random resampling with replacement [313]. The main advantage of this technique, compared to others such as the Wilcoxon test [314], is that it provides a clear and rigorous account of the inference time change and the associated uncertainty [312], [313], [315]. For example, this method allows us to state that a compressed model is faster than the original CodeBERT model by -30%$\pm$2% with 95% confidence. We consider a difference to be statistically significant if the confidence interval is not greater than the percentage change.

Model size. To measure the model size, we save the model state in memory using the `save` function provided by PyTorch and then calculate its size using the `getsize` Python function. The value the function returns is converted to megabytes (MB). Although we performed this process on both CPU and GPU, as expected, the models' size remained unchanged between the two environments. Therefore, we do not differentiate between CPU and GPU when reporting the models' size in Section 11.1.2.

Effectiveness Metrics.

For effectiveness, we assess how good the predictions of a model are for a specific task. Given the heterogeneity of tasks involved in our evaluation, we used different metrics depending on the SE task being analyzed (see Table 11.1).

Vulnerability Detection. We use the *Matthews Correlation Coefficient (MCC)* as it considers all quadrants of the classification matrix (i.e., it gives a comprehensive overview of the model's performance). It has been shown to be a reliable measure

when handling imbalanced data, as is often the case for vulnerability prediction [239], [300], [316], [317]. MCC is defined as a correlation factor between the true and predicted labels. It ranges from -1 to 1, where -1 means the model gives opposite predictions, 0 means random predictions, and 1 means perfect predictions. In our study, we also report on *F1 Score* [244] and *Accuracy* [148] for compatibility with respect to previous work. The F1 Score is defined as the harmonic mean between Precision and Recall. Accuracy is defined as the number of correct predictions over the whole predictions of a model. Both F1 Score and Accuracy values range from 0 to 1, where 1 is the best value. Although the use of Accuracy is deprecated for problems suffering from data imbalance [239], we include this measure in our analysis for completeness as it is the metric employed by the CodeXGLUE benchmark. Still, we discourage its use in practice as done in previous work [239].

Code Summarization. We employ three metrics that assess different aspects of the quality of a generated text [318]. Bleu is the metric employed in the CodeXGLUE benchmark and is a standard metric adopted in natural language translation and, generally, text generation tasks [301]. It computes how similar a generated text is with respect to a reference baseline by comparing overlapping n-grams (contiguous sequences of n words) between the generated and reference texts. It is a metric of *summary-summary text similarity*, but has been criticised for not considering the semantic similarity between two texts [274], [318]. For this reason, we extended the evaluation by including two additional metrics. BERTScore evaluates the quality of a generated text by comparing the similarity between the BERT embeddings of the generated text and the reference baseline [302]. It is a metric of *summary-summary semantic similarity* [318]. Finally, SIDE is a metric based on contrastive learning that measures how good a generated explanation is for a given code snippet without considering a reference baseline [224]. It is specific for code summarization tasks and is a metric of *summary-code semantic similarity* [318]. All metrics range between 0 and 1, where 1 is the optimum value.

Code Search. We employ three versions of the Mean Reciprocal Rank (MRR) score [303]. We adopt this metric because it is used in the CodeXGLUE benchmark and is by far the most commonly adopted metric in code search [319]. MRR measures the average of the reciprocal ranks of the correct results for a set of code comments. The reciprocal rank for a single code comment is defined as the inverse of the rank position where the correct corresponding code appears in the list of retrieved results. In addition, we include two variations of MRR: MRR@1, which measures the proportion of code comments where the correct code appears in the first position, and MRR@5, which calculates the mean reciprocal rank based on the top five results. These metrics all range from 0 to 1, with 1 representing the optimal score.

11.1.2 Empirical Study Results

In this section, we report the result of our empirical evaluation. For each RQ, we discuss the impact of each compression strategy relative to the model's efficiency and effectiveness, as well as the trade-off between these two aspects.

Table 11.3 shows the results for each RQ. On each sub-table, the first row reports the results of the plain CodeBERT model (i.e., without compression), while the remaining rows show the percentage variations provided by each compression

TABLE 11.3: RQs 1-3: Efficiency and effectiveness of original and compressed code models for each of the SE tasks.

(A) RQ_1: Vulnerability Detection

Compression Method	Efficiency			Effectiveness		
	CPU Inf. Time	GPU Inf. Time	Model Size	Accuracy	F1	MCC
None	15.902 (sec)	0.011 (sec)	499 (MB)	0.630	0.541	0.247
Know. Distil.	-39.8% ± 2.7%	**-47.7% ± 0.6%**	-48.8%	-2.2%	**+3.1%**	-10.1%
Pruning (0.2)	+15.5% ± 6.0%	+9.1% ± 2.0%	0.0%	-4.4%	-41.0%	-11.3%
Pruning (0.4)	+18.8% ± 7.2%	+6.2% ± 1.7%	0.0%	-7.3%	-61.0%	-20.6%
Pruning (0.6)	**-67.9% ± 1.4%**	+7.3% ± 1.6%	0.0%	-5.9%	-49.2%	-18.2%
Quantization (float8)	+41.9% ± 16.3%	+98.3% ± 3.2%	-51.4%	**0.0%**	-1.1%	**+0.4%**
Quantization (int8)	+102.2% ± 14.4%	+107.4% ± 5.1%	-51.4%	-0.5%	-0.9%	-2.4%
Quantization (int4)	+133.5% ± 26.3%	+201.6% ± 4.8%	**-59.3%**	-1.6%	-4.4%	-8.5%

(B) RQ_2: Code Summarization

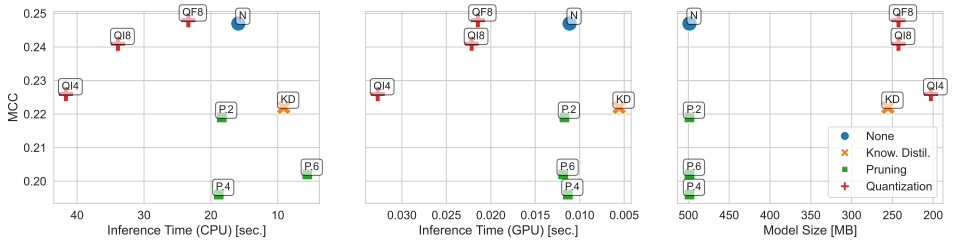
Compression Method	Efficiency			Effectiveness		
	CPU Inf. Time	GPU Inf. Time	Model Size	Bleu	BERTScore	SIDE
None	157.369 (sec)	23.692 (sec)	707 (MB)	18.791	0.888	0.871
Know. Distil.	-39.8% ± 7.8%	**-2.2% ± 4.9%***	-33.0%	-42.3%	-6.1%	-70.6%
Pruning (0.2)	**-45.3% ± 4.9%**	+9.8% ± 1.9%	0.0%	-4.3%	-0.1%	**+0.4%**
Pruning (0.4)	+24.7% ± 19.8%	+121.9% ± 13.1%	0.0%	-66.6%	-17.7%	-42.4%
Pruning (0.6)	+183.5% ± 41.9%	+419.3% ± 29.5%	0.0%	-93.4%	-58.4%	-93.0%
Quantization (float8)	-20.3% ± 7.9%	+14.0% ± 2.2%	-42.0%	**+0.4%**	**0.0%**	+0.1%
Quantization (int8)	-27.2% ± 6.7%	+6.2% ± 2.3%	-42.0%	-0.3%	**0.0%**	+0.0%
Quantization (int4)	-17.9% ± 7.0%	+29.1% ± 2.5%	**-51.9%**	-2.0%	-0.1%	+0.2%

(C) RQ_3: Code Search

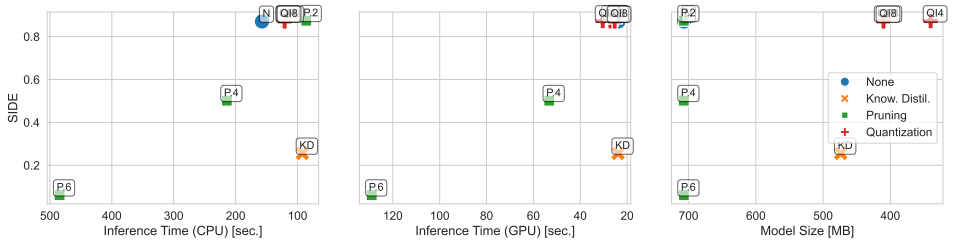
Compression Method	Efficiency			Effectiveness		
	CPU Inf. Time	GPU Inf. Time	Model Size	MRR	MRR@1	MRR@5
None	6.047 (sec)	0.010 (sec)	499 (MB)	0.329	0.242	0.310
Know. Distil.	**-84.7% ± 0.7%**	**-29.2% ± 0.5%**	-48.7%	-52.4%	-57.7%	-54.3%
Pruning (0.2)	+5.5% ± 1.0%	+11.1% ± 0.7%	0.0%	-3.2%	-3.4%	-3.4%
Pruning (0.4)	+16.1% ± 2.2%	+6.4% ± 1.4%	0.0%	-52.1%	-57.3%	-54.3%
Pruning (0.6)	+3.1% ± 3.5%*	+19.8% ± 3.3%	0.0%	-99.6%	-99.9%	-99.8%
Quantization (float8)	+34.2% ± 2.9%	+113.2% ± 1.8%	-51.4%	-0.2%	**0.0%**	-0.3%
Quantization (int8)	+42.2% ± 3.8%	+105.9% ± 1.1%	-51.4%	**0.0%**	-0.1%	**+0.1%**
Quantization (int4)	+61.8% ± 4.8%	+209.6% ± 2.0%	**-59.3%**	-6.3%	-7.6%	-6.7%

strategy.[3] For inference time, we also report the confidence interval for the change using the Kalibera and Jones approach [313] (see Section 11.1.1 for details). Non-statistically significant changes are marked with an asterisk (*). For each considered efficiency or effectiveness metric, the best values are highlighted in **bold**, while the worst values are underlined. We also present scatter plots showing the trade-off between effectiveness (y-axis) and efficiency (x-axis) for each RQ in Figure 11.2.

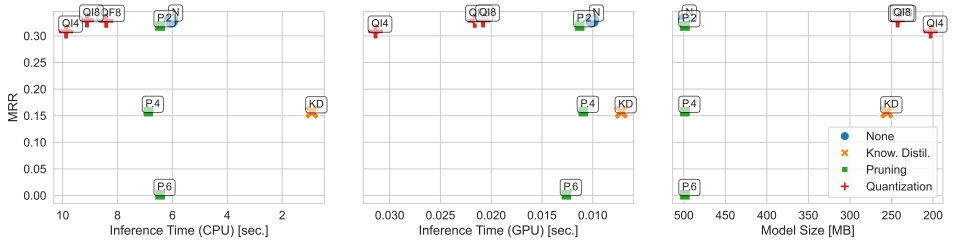
[3]For inference time, the first row reports the median of measurements across batches of the plain CodeBERT model. The remaining rows show the percentage variations in the median inference time provided by each compression strategy, computed using the Kalibera and Jones approach [313].



(A) RQ_1: Vulnerability Detection



(B) RQ_2: Code Summarization



(C) RQ_3: Code Search

FIGURE 11.2: Trade-off between effectiveness (y-axis) and efficiency (x-axis) metrics for each of the SE tasks.

RQ_1 Results - Vulnerability Detection

We analyze the impact of compression strategies from various aspects: inference time, model size, and vulnerability detection effectiveness. Additionally, we investigate the trade-offs involved among these metrics.

Inference Time. As reported in Table 11.3a, all compression strategies change the inference time of the vulnerability detection model with statistical significance. However, the impact varies widely depending on the specific compression strategy and the hardware environment. On the CPU, for example, the most aggressive configuration of pruning (0.6) leads to the highest speed-up across all the strategies, with an inference time reduction of -67.9% compared to the plain CodeBERT model. Interestingly, less aggressive forms of pruning (0.2 and 0.4) lead to the opposite outcome, with an inference slow-down of $+15.5\%$ and $+18.8\%$, respectively. All pruning configurations also negatively affect inference time in GPU environments, with slowdowns of up to $+9.1\%$. A possible explanation for this behavior could be identified in the lower ability of GPU to handle sparse-matrix multiplications [271]. These results of model pruning suggests that this strategy requires specific hardware and configurations to improve the inference time of vulnerability detection models; however, when these conditions are met, the benefits can be substantial.

From Table 11.3a, we also observe that quantization negatively impacts inference time in all configurations, resulting in the highest slowdowns across all compression

strategies in both CPU and GPU environments. `int4` quantization, in particular, causes the most significant slowdowns, with increases of $+133.5\%$ on the CPU and $+201.6\%$ on the GPU. These negative results could be explained by the fact that CPUs and GPUs are heavily optimized for full-precision floating-point operations (e.g., 32-bit), hence quantized models may not fully take advantage of these optimizations and experience an inference slowdown [67], [266], [320], [321].

Knowledge Distillation, on the other hand, consistently and significantly improves inference time across both CPU and GPU environments. It ranks as the second-best strategy (after Pruning 0.6) on the CPU, with a reduction of -39.8% in inference time, and as the best strategy on the GPU, with a speed-up of -47.7%.

Model Size. As can be observed from Table 11.3a, all the compression strategies, with the exception of pruning, reduce the size the vulnerability detection model. `int4` quantization provides the highest model's size reduction (-59.3%), followed by the other two quantization configurations (-51.4%). Those results align with the expected quantization behaviour, where a lower bits' precision implies a lower model size. Knowledge Distillation reduces the original model size by almost half (-48.8%). This result aligns with the smaller architecture of the distilled model compared with the original fine-tuned CodeBERT model. In contrast, pruning does not affect the model size. This behavior can be explained by the unstructured nature of the pruning strategy employed, where pruning impacts only the weight values by setting them to zero without modifying the network structure itself [266].

Effectiveness. Notably, we find that quantization has very limited impact on the model's effectiveness (see Table 11.3a). `float8` quantization marginally changes the effectiveness, in terms of Accuracy, F1-score, and MCC (0.0%, -1.1%, and $+0.4\%$, respectively). Similar results hold for `int8` quantization, while `int4` quantization provides a slightly higher degradation, especially for F1 (-4.4%) and MCC (-8.5%).

Knowledge distillation performs slightly worse than quantization, particularly in terms of MCC (-10.1%) and Accuracy (-2.2%). However, we observe an improvement in the F1-score ($+3.1\%$). This means that the distilled model has a higher tendency to predict vulnerabilities compared to the original fine-tuned CodeBERT model, which can decrease the number of true negative instances (i.e., increase the number of false positive instances). This leads to an improvement in recall, while only marginally affecting precision, which overall positively impacts the F1-score. Nonetheless, the reduction in MCC suggests a lower correlation between the predicted vulnerabilities and the actual ones.

Pruning has the most significant detrimental impact on vulnerability detection effectiveness. All the pruning configurations considerably reduce model effectiveness, with MCC changes ranging from -11.3% to -20.6%. Among all compression strategies, Pruning 0.4 performs the worst across all three metrics, showing effectiveness losses of -7.3%, -61%, and -20.6% for Accuracy, F1-score, and MCC, respectively. Interestingly, Pruning 0.6 performs slightly better than Pruning 0.4, despite prior research suggesting more severe effectiveness degradation with higher pruning levels [68].

Trade-offs. Figure 11.2a shows the effectiveness-efficiency trade-off provided by each compression strategy. For each sub-plot, the upper-right solutions are the best. We use MCC as the effectiveness metric since, as explained in Section 11.1.1, it is the most comprehensive metric for classification [239].

From the far-right subplot, we observe that quantization achieves the best model size reduction while maintaining effectiveness levels comparable to the baseline. However, as shown in the first two subplots, this gain comes at the cost of increased inference time on both CPU and GPU. From Figure 11.2a, we observe that pruning is consistently outperformed by other strategies across all efficiency and effectiveness metrics. The only exception is Pruning 0.6, which offers the fastest inference time but at the expense of a comparatively low effectiveness. Knowledge distillation is, on the other hand, the only compression strategy that improves efficiency across all dimensions (i.e., CPU and GPU inference time, and model size). At the same time, this strategy results in a comparatively moderate effectiveness degradation–greater than quantization but less than pruning. Overall, this makes knowledge distillation the strategy that offers the most balanced trade-off between efficiency gains and effectiveness degradation.

Answer to RQ_1: Quantization is the most effective strategy for reducing memory size in vulnerability detection models (up to -59.3%), with minimal impact on model effectiveness (-8.5% MCC in the worst case). However, it can significantly increase inference time, by as much as $+201.6\%$. Pruning, on the other hand, shows no efficiency gains, with the exception of the 0.6 configuration that results in notable speed-up on CPU (up to -67.9%), but at the cost of a -18.2% effectiveness degradation in MCC. Finally, Knowledge Distillation improves both inference time (up to -47.7%) and model size (up to -48.8%), while moderately impacting vulnerability detection effectiveness, with a reduction of -10.1% in MCC.

RQ_2 Results - Code Summarization

Inference Time. As shown in Table 11.3b, Pruning 0.2 is the most effective approach for reducing inference time on the CPU, achieving a -45.3% reduction compared to the plain CodeBERT model. Counter intuitively, we observe a negative correlation between the percentage of weights pruned and the inference time reduction. For example, Pruning 0.4 performs worse than Pruning 0.2, increasing inference time by $+24.7\%$, while Pruning 0.6 shows the worst behavior, increasing inference time by $+183.5\%$ on the CPU. On the GPU, we observe an even more pronounced worsening trend, as inference times increase by $+9.8\%$, $+121\%$, and $+419.3\%$ for Pruning 0.2, 0.4, and 0.6, respectively. A possible explanation for this behavior could be the lower hardware's ability to handle sparse matrix multiplications [322]. Since generation tasks require multiple network forward steps, a higher sparsity of the weights could increase the overall inference time.

Quantization strategies can all reduce the inference time on the CPU, with `int8` quantization being the best configuration (-27.2%). This behavior differs from what we observed for vulnerability detection and could be explained by the higher complexity of the underlying task, which may benefit more from reduced weight precision [67]. On the other hand, on the GPU, quantized models are overall slower than the plain CodeBERT model, with inference time slow-downs ranging from $+6.2\%$ to $+29.1\%$.

Knowledge Distillation is the only strategy capable of reducing the inference time of code summarization models across both CPU and GPU environments. On the CPU, it is the second-best compression strategy for inference time reduction (-39.8%). On the GPU, it is the only strategy to achieve an inference time reduction, with a decrease of (-2.2%). However, this reduction is not statistically significant,

as the confidence interval for the relative change includes zero (see Section 11.1.1 for details).

Model Size. From Table 11.3b, we observe how, like in vulnerability detection, `int4` quantization is the best strategy for reducing the model's size (-51.9%), followed by the other two quantization configurations. Again, these results align with the expected quantization behavior, where the lower the precision, the lower the size. Knowledge Distillation can also reduce the model size (by -33%), while the adopted pruning strategies confirm that they do not lead to any change.

Effectiveness. From Table 11.3c we observe that all quantization strategies provide only marginal change in the model's effectiveness, with `float8` and `int8` quantization having almost comparable metrics to the baseline, and `int4` quantization showing limited degradations of -2%, -0.1%, -0.2% for Bleu, BERTScore and SIDE, respectively.

For pruning, we observe that the impact grows as pruning becomes more aggressive. Specifically, Pruning 0.2 shows marginal impact, with a slight degradation of -4.3% in summary-summary text similarity (BLEU). On the other hand, Pruning 0.4 and 0.6 substantially reduce the model's effectiveness across all dimensions. For instance, in terms of summary-code semantic similarity (SIDE), Pruning 0.4 and 0.6 lead to effectiveness decreases of -42.4% and -93%, respectively. This behavior is in line with previous research showing how the effectiveness of a BERT model starts to decrease if the amount of pruned weights is $\geq 40\%$ [68].

Knowledge Distillation also shows a significant impact, particularly in terms of summary-summary text similarity (-42.3% in Bleu) and summary-code semantic similarity (-70.6% in SIDE). This behavior is consistent with previous studies, which highlight that distilled models are less effective for tasks beyond classification [267].

Trade-offs. Figure 11.2b illustrates the trade-off between effectiveness and efficiency. We use SIDE as the effectiveness metric, as it is specifically designed for code summarization tasks [224]. We observe that Knowledge Distillation overall improves the inference time and size of the model, but it significantly degrades effectiveness. In contrast, quantization generally performs well across all efficiency metrics, with only a slight slow-down in inference time on the GPU. Moreover, this strategy has only a marginal impact on the model's effectiveness, with values comparable with the baseline. As such, quantization appears to offer the best balance between efficiency and effectiveness as a compression strategy. Pruned models are generally outperformed by other compressed models in terms of both efficiency and effectiveness. The only exception is Pruning 0.2, which provides the best trade-off between inference time and effectiveness on the CPU. However, using pruning does not improve model size.

Answer to RQ_2: Quantization strategies offer the best efficiency-effectiveness trade-off among compression techniques, with inference time speed-up of up to -27.2%, model size reduction of up to -51.9%, and minimal effectiveness degradation of up to -0.2% in SIDE. Pruning generally underperforms compared to other strategies, but under specific configurations, such as Pruning 0.2, it achieves the best inference time improvements on the CPU (-45.3%). While Knowledge Distillation improves all efficiency metrics (with up to -39.8% in inference time and -33% in model size), it causes significant losses in effectiveness, with reductions of -70.6% in SIDE.

RQ_3 Results - Code Search

Inference Time. Table 11.3c reports how Knowledge Distillation is the strategy that better reduces the inference time on both CPU and GPU, with improvements of -84.7% and -29.2%, respectively. We observe that all other compression strategies have a negative impact on inference time. Pruning slows down inference on both CPU and GPU, with an increase in inference time ranging from $+3.1\%$ to $+19.8\%$. A possible explanation for this behavior with pruning could be the complexity of the code search task, which typically requires multiple comparisons between a code comment and the code snippets. In that, the sparsity of matrices could increase inference time, especially if the hardware is not well-optimized for handling sparse matrices [271]. Similarly to vulnerability detection, we observe that quantization consistently increases the inference time of code search on both CPU and GPU. The magnitude of the increase is lower on CPU, ranging from $+34.2\%$ to $+61.8\%$, and higher on GPU, with variations from $+113.2\%$ to $+209.6\%$. The negative behavior of quantization strategies could be (once again) explained by the lack of optimization in CPU and GPU kernels for low-precision bit operations [67], [266], [320], [321].

Model Size. Since the model used for this task is the same as the one used for vulnerability detection (i.e., CodeBERT), the results regarding the model size are identical. Hence, `int4` quantization emerges as the best strategy, while the unstructured nature of pruning does not imply any change.

Effectiveness. Table 11.3c shows how `int8` and `float8` quantization provide no significant change compared with the baseline. A slightly higher degradation is instead observed with `int4` quantization (-6.3% in MRR). All pruning strategies provide a degradation in the model's effectiveness, with reductions in MRR ranging from -3.2% to -99.6%. We observe a correlation between the increase in effectiveness loss and the percentage of pruned weights. Pruning 0.6 strategy proves to be the worst compression strategy, with an MRR loss of -99.6%. Finally, we also observe a significant degradation in effectiveness for Knowledge Distillation, with a -52.1% loss in MRR. This result is consistent with previous research, which has shown that distilled models often exhibit lower effectiveness on tasks different from classification [267].

Trade-offs. Figure 11.2c shows the effectiveness-efficiency trade-off. We adopt the general MRR score as the effectiveness metric. We observe that quantization strategies are preferred to reduce the model size while not impacting its effectiveness; however, they negatively influence the inference time. Knowledge Distillation achieves significant reductions in both inference time and model size, but it also drastically

reduces the model's effectiveness, with a loss of -52.1% in MRR. Finally, pruning strategies do not achieve positive results in any of the efficiency and effectiveness metrics analyzed.

> **Answer to RQ_3:** Quantization strategies drastically reduce the size of code search models (up to -59.3%) with only a marginal impact on effectiveness (up to -6.3% in MRR). However, they can significantly slow down inference time, with an increase of up to $+61.8\%$ on CPU and $+209.6\%$ on GPU. Knowledge Distillation reduces both model size (-48.7%) and inference time (-84.7% on CPU and -29.2% on GPU), but it comes at the cost of a considerable loss in effectiveness (-52.4% in MRR). Pruning, however, does not show improvements in any of the efficiency metrics analyzed.

11.1.3 Discussion

In the following, we discuss the overall behaviour of the analysed compression strategies and report insights for practitioners and researchers derived from our empirical evaluation.

Performance of LLM Compression Strategies

Knowledge Distillation. We found that Knowledge Distillation is the only compression strategy capable of improving both inference time and model size across all the software engineering tasks we considered. However, this strategy can lead to a significant loss in effectiveness. This loss in effectiveness is particularly pronounced in tasks like code search and summarization, while it is milder in vulnerability detection. These findings align with previous research, which suggests that distilled models tend to be less effective for tasks other than classification [267]. Indeed, to date, knowledge distillation has predominantly been applied to code classification tasks, such as vulnerability detection and clone detection [265], [266].

Model Quantization. We found that quantization drastically reduces the size of models across all tasks while maintaining relatively high effectiveness. However, this improvement often comes at the cost of increased inference time. On GPU environments, quantization can double or even triple the inference time for tasks like vulnerability detection and code search. In contrast, the slowdown in inference time is much less pronounced for the code summarization task. Interestingly, in CPU environments, quantization can even improve the inference time for code summarization. However, for vulnerability detection and code search, it still introduces notable slowdowns. Based on these findings, we hypothesize that quantization performs better in terms of efficiency when the task is complex—requiring multiple forward passes through the model—such as in code generation tasks. These results align with previous studies on the application of quantization for code generation tasks [274].

Model Pruning. We found that while pruning offers limited overall benefits in terms of efficiency, it can lead to significant speed-ups in inference time under specific combinations of configurations and environments. For instance, Pruning 0.2 provides the highest inference speed-up for code summarization on CPU among all compression strategies, with only a marginal reduction in effectiveness. Similarly,

in the vulnerability detection task, Pruning 0.6 proves to be the most effective strategy for improving inference time on CPU, though it comes with a moderate loss in effectiveness. These findings suggest that, when properly configured, pruning can deliver substantial inference speed-ups, particularly in CPU environments.

Insights

Insights for Practitioners. Our results indicate that the impact of different compression strategies can vary significantly depending on the SE task and the underlying execution environment, often involving important efficiency-effectiveness trade-offs. Hence, practitioners should carefully select the compression strategy based on their requirements and the underlying task as follows:

- If the practitioner's priority is to improve both inference time and model size, regardless of the task or underlying execution environment, Knowledge Distillation is the preferred choice. Indeed, it is the only compression strategy that delivers improvements across all efficiency aspects in both CPU and GPU environments. However, practitioners should be aware of potential negative impacts on effectiveness, particularly in tasks other than code classification (e.g., code summarization and code search). Additionally, it is important to note that Knowledge Distillation is the only compression strategy that requires retraining a model from scratch, which may be undesirable if practitioners lack sufficient computational resources.
- If the priority is to reduce model size without significantly degrading effectiveness, quantization is the natural choice, regardless of the SE task. Nevertheless, practitioners should be mindful of the potential side effects on inference time, which can vary greatly depending on the task and environment, and often result in significant slowdowns. In some specific cases, however, quantization can even improve inference time, such as code summarization on CPU.
- If the practitioner's goal is to reduce inference time in GPU environments, the only viable choice is Knowledge Distillation. However, as previously mentioned, this approach can drastically affect model effectiveness, particularly for tasks like code summarization and code search.
- If the priority is to improve inference time on CPU, the practitioner could once again consider Knowledge Distillation. They may also consider pruning, especially for tasks such as vulnerability detection and code summarization. However, pruning must be carefully configured to provide benefits. For instance, a less aggressive form of pruning (0.2) proved beneficial in reducing inference time for vulnerability detection, while a more aggressive form (0.6) resulted in the highest inference speed-up for code summarization. For code search, Knowledge Distillation remains the only viable option for reducing inference time on CPU, though it drastically reduces the model's effectiveness. In fact, we did not find any compression strategy that improves inference time for code search without significant losses in effectiveness.

Insights for Researchers. Our results show that the behaviour of compression strategies greatly varies depending on the context. However, when these strategies are carefully selected for the underlying task and environment, they can significantly enhance efficiency aspects with a marginal impact on effectiveness. For instance,

we found that specific quantization configurations can enhance both CPU inference time and model size without influencing the effectiveness of code summarization. These findings highlight the potential for developing approaches that automatically select the optimal compression strategy based on the underlying task, execution environment, and efficiency/effectiveness requirements. Moreover, we observed that the efficacy of compression strategies can highly depend on their specific configuration. This behavior is particularly evident in the pruning strategy, where the amount of pruned weights significantly influence its impact on inference time. We encourage future research aimed at developing approaches to automatically identify the optimal compression configuration (e.g., amount of weights to prune), based on the SE task and execution environment.

11.1.4 Threats to Validity

Internal Validity: Execution time measurements are typically subject to variability, which can hinder rigorous evaluation [323]. To address this issue, we followed best practices from performance engineering to increase the reliability of our evaluation [308]–[313] (see Section 11.1.1). Furthermore, the outcomes may be influenced by potential errors in the implementation of the baseline models and compression strategies. To address this concern, we utilized a widely adopted benchmark (CodeXGLUE) for training and testing the baseline models and employed compression strategies implementations from widely used and maintained libraries.

Construct Validity: As explained in Section 11.1.1, we employed multiple metrics for each task to assess a model's effectiveness. This has been done to account for possible threats concerning adopting specific metrics (like Accuracy [239] or Bleu [318]). Concerning efficiency metrics, we relied on standard approaches and statistics to measure inference time and model size.

External Validity: The major threats of our work concern its generalizability. The results obtained are limited to the models and tasks analysed and the environment in which the experiments were run. In addition, the results concerning Knowledge Distillation are specific to the DistilBERT LLM and may not hold for other distilled models. Future work can extend our analysis with other LMs and code-related tasks and by analysing other Knowledge Distillation techniques. To this end we strove to describe the methodology we followed as clearly as possible and have made our code and scripts publicly available [69].

11.2 Improving Inference Time and Image Quality of Image Generation Models

As highlighted in Section 8.1, text-to-image generation models have garnered significant attention due to their potential to bridge the gap between textual descriptions and visual representations [324]. However, achieving high quality of the generated images involves fine-tuning various aspects of a generative model, such as the number of inference steps or positive and negative prompts [277]. At the same time, like all LLMs, image generation models are energy and resource-demanding and present a significant inference time [16], [325].

Berger et al. [277] proposed a search-based approach, dubbed StableYolo, to optimize the image quality of Stable Diffusion by assessing image quality using the Yolo pre-trained model for image-captioning[326]. However, their approach does not

take into account the issue of inference time, which is a cornerstone for both ensuring user experience and minimizing the energy consumption of generative models. Moreover, the compression strategies evaluated in Section 11.1 have been shown to expose efficiency and effectiveness trade-offs and may behave inconsistently based on the underlying task.

To address this gap, we extend the work of Berger et al. by presenting *GreenStableYolo* [72], a novel approach that addresses the challenge of optimizing the trade-off between inference time and image quality using a search-based multi-objective optimization method, namely Non-dominated Sorting Genetic Algorithm (NSGA-II) [71]. More in detail, GreenStableYolo searches for the optimal configuration of hyperparameters and prompt structure to reduce the inference time while maintaining a high quality of the generated images. Being a post-processing method, it can be applied to a black-box model without altering its internal architecture. Thus, it can be employed during the *model deployment* phase to reduce the image generation time.

We provide initial empirical evidence that by using GreenStableYolo, Stable Diffusion models achieve a satisfactory equilibrium between inference time and image quality, making it suitable for real-world applications where both factors play a crucial role.

11.2.1 Methodology

GreenStableYolo is a novel multi-objective search-based approach that, given a text prompt for image generation, searches for the optimal parameters that can strike a trade-off between:

1. *Inference time*, which is measured by the GPU time taken for the execution of the `StableDiffusionPipeline`;
2. *Image quality*, which is determined by performing object recognition with Yolo, then selecting objects that match the input prompt, and computing their average probabilities [277].

NSGA-II Optimization Algorithm

To simultaneously enhance image quality and reduce inference time, we leverage NSGA-II, a well-known and efficient multi-objective evolutionary algorithm [327], [328]. Specifically, NSGA-II works as follows:

1. Initialize a population with N individuals;
2. Perform crossover and mutation operations, generating an offspring population denoted as P_o;
3. Reassemble the parent population P_{t-1} and P_o into a temporary population with the size of $2N$, and formulate individuals into i non-inferior frontier through fast non-dominating sorting;
4. Select N individuals from the temporary population to form the next population for the t^{th} iteration, denoted as P_t.
5. Repeat steps (2)-(4) until the termination condition is met;
6. The algorithm ends up and returns the current Pareto-Optimal set.

Selected Parameters

To make a straightforward comparison with StableYolo, we adopt the same settings as used by Berger et al. [277]. Specifically, the following parameters and prompts are tuned and searched with NSGA-II:

- **Inference steps** (1 to 100): the AI's image generation iterations;
- **Guidance scale** (1 to 20): the impact of the prompt on image generation;
- **Guidance rescale** (0 to 1): rescales the guidance factor to prevent over-fitting;
- **Seed** (1 to 512): randomization seed;
- **Positive prompt**: used to describe images and improve their details, e.g., *"photograph"*, *"color"*, and *"ultra real"*;
- **Negative prompt**: avoided description during image generation, e.g., *"sketch"*, *"cropped"*, and *"low quality"*.

11.2.2 Evaluation

To evaluate our proposal, we address the following research questions (RQs):

RQ_1: *To what extent can GreenStableYolo improve image quality and inference time compared with StableYolo?*

RQ_2: *How do parameters/prompts of Stable Diffusion influence the inference time for image generation?*

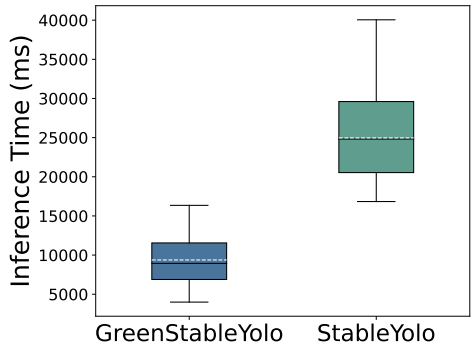
RQ_3: *How do parameters/prompts of Stable Diffusion influence the quality of the generated images?*

Experimental Setup

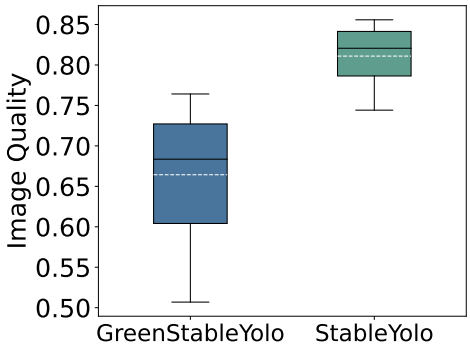
To ensure a fair evaluation of the optimization effectiveness, we employed the same hyperparameter setup as StableYolo for NSGA-II, e.g., the population size was set to 25, the number of generations was set to 50, and both the mutation rate and crossover rate were set to 0.2. We selected the weights of 0.001 for image quality and -1000 for inference time based on empirical investigation of different weight combinations. In addition, we used Stable Diffusion version v2 and Yolo version v8. To assess variability, we evaluated each model 15 times using different random seeds, focusing solely on the prompt "two people and a bus" due to time constraints. Any future studies can explore additional prompts. All experiments were conducted on a virtual machine hosted on Google Colaboratory, with an NVIDIA Tesla T4 GPU with 16 GB of RAM.

RQ1 Results

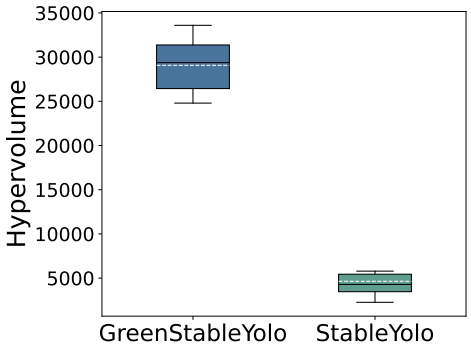
Figure 11.3 presents the performance comparisons between GreenStableYolo and StableYolo. Specifically, Figure 11.3a reveals that GreenStableYolo achieves an average inference time of 9.4 seconds with an interquartile range (IQR) of 4.7 seconds. Conversely, StableYolo exhibits an average inference time of 25.0 seconds, which is 1.66 times higher than GreenStableYolo, with an IQR of 9.1 seconds. That is, GreenStableYolo generates images much faster.



(A) Inference time



(B) Yolo quality measure



(C) Hypervolume

FIGURE 11.3: Comparison of GreenStableYolo and StableYolo on 15 independent runs

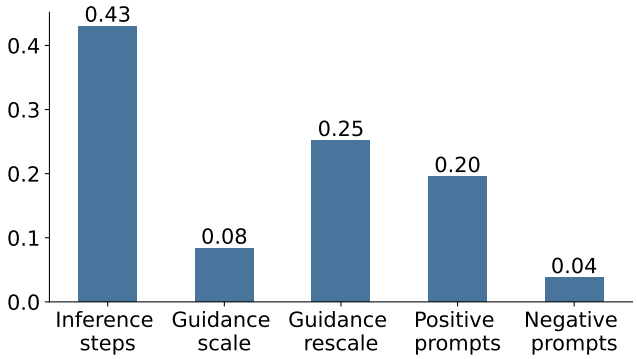
This improvement in inference time comes at a slight cost to image quality. As illustrated in Figure 11.3b, GreenStableYolo experiences approximately an average degradation of 0.18 points in image quality. We also compute the hypervolume [329] for both models for a more comprehensive comparison[4]. Figure 11.3c presents the hypervolume values with the reference point set as (1, 50000), where GreenStableYolo achieves an average hypervolume of 29074.11, surpassing StableYolo's score of 4642.17 by 5.26 times.

> **Answer to RQ$_1$:** GreenStableYolo significantly improves the StableYolo baseline in this two-objective optimization problem for text-to-image generation.

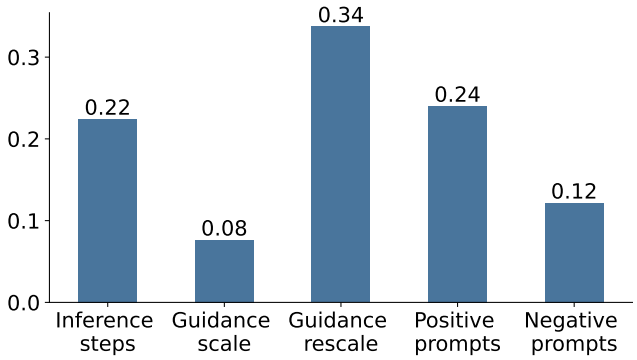
Results of RQ2 and RQ3

To investigate RQs2–3, we followed previous work [330] and built two Random Forest regression models using `scikit-learn`. The features of these models include the number of iteration steps, guidance scale, guidance rescale, positive prompts, and negative prompts (excluding the random seed). The target variables are inference time and image quality score, respectively. We use the `RandomizedSearchCV` function from `scikit-learn` to find the optimal hyperparameters during model training. The `feature_importances_` function is then used to compute the importance of each

[4]Hypervolume is a fundamental metric used in multi-objective optimization problems that indicates the dominance of a solution in the objective space.



(A) RQ2: MDI w.r.t. inference time



(B) RQ3: MDI w.r.t. image quality

FIGURE 11.4: Parameters and prompts importance based on the mean decrease in impurity

parameter and prompt based on the Mean Decrease Impurity (MDI), a.k.a. as Gini importance. To ensure reliability, we repeat this process 10 times.

Figures 11.4a and 11.4b present the calculated importance of parameters and prompts based on the mean decrease in impurity, with respect to inference time and image quality scores, respectively. As shown in Figure 11.4a, the number of *inference steps* emerges as a significant factor affecting inference time. This is expected, as more steps involve more computations, thereby resulting in higher inference time. Meanwhile, for image quality (Figure 11.4b), parameters like *guidance rescale* and *positive prompts* play a relatively more critical role.

Answers to RQ_2 and RQ_3: The number of inference steps is the most crucial parameter affecting inference time, while guidance rescale and positive prompts are more critical for image quality.

The results of our evaluation highlight the importance of identifying optimal parameter combinations during model inference to balance computational efficiency and output quality.

11.2.3 Threats to Validity

The limited exploration of prompts, the randomness in the optimization process, and the specific configuration for NSGA-II may introduce internal threats. Besides,

external threats may include the choice of the GenAI model, the noise when measuring the inference time, and the evaluation of image quality based on object recognition using Yolo.

11.3 Conclusion

In this chapter, we addressed the issue of the efficiency of learning-based systems employing LLMs. We first performed an empirical evaluation of the impact that the adoption of three LLM compression strategies (i.e., Knowledge Distillation, Quantization, and Pruning) could have on the inference time, memory size, and effectiveness of models fine-tuned for three widely adopted SE tasks – vulnerability prediction, code summarization, and code search. Results show how each strategy provides some trade-offs among the three analyzed dimensions and how the proper compression method should be chosen based on the underlying task and the practitioner's priorities.

Following this evaluation, we introduced GreenStableYolo, the first approach leveraging NSGA-II to strike an optimal trade-off between inference time and image quality for Stable Diffusion models. Experimental comparisons with the StableYolo baseline demonstrate that GreenStableYolo achieves significantly reduced inference time while maintaining a relatively high image quality.

Part III

Conclusion

Chapter 12

Conclusion

The quality-based development of learning-based systems is a complex task, and different challenges arise in the various phases of the development pipeline. In this thesis, we proposed a set of contributions to address some of the most peculiar challenges by easing and standardizing the quality-based development of learning-based systems. In particular, we focused on the *fairness* and *effectiveness* of learning-based systems and presented a set of contributions spanning through almost all the steps of a learning-based system development workflow.

In the following, we summarize the different contributions by placing them in the various phases of the workflow depicted in Figure 1.1:

- **Model Requirements.** Concerning the *model requirement* phase, the first contribution relates to introducing two low-code approaches, MANILA and MODNESS, to assist data scientists and researchers in developing fair learning-based systems. Both approaches contribute to different phases of a learning-based system development workflow. In this specific phase, these approaches can assist data scientists and domain experts in specifying fairness analyses at a high level and selecting compatible combinations of machine learning models and fairness-enhancing methods for evaluation. A second contribution relates to an initial exploration of approaches to assist data scientists in selecting efficient ML models from a training time perspective. In particular, we analyzed the FPTC approach and highlighted its strengths and weaknesses.

- **Feature Engineering.** For the *feauture engineering* phase, we presented the *Debiaser for Multiple Variables (DEMV)*. DEMV is a pre-processing algorithm able to mitigate the bias of a dataset both in binary and multi-class classification tasks. In addition, we presented an empirical study on the effectiveness of dataset's *bias symptoms* to early predict algorithmic bias before training an ML model.

- **Model Training.** By automatically generating the Python code to train the selected ML models and fairness-enhancing method combinations, MANILA supports data scientists also in the *model training* phase. In addition, MANILA can automatically execute the code in the web infrastructure.

- **Model Evaluation.** Both MANILA and MODNESS assist data scientists in the *model evaluation* phase by automatically computing the metrics specified during the *model requirements* phase. Additionally, we performed a first investigation on how the LLM fairness assessment process is performed in GitHub projects. Surprisingly, we did not observe any project employing LLMs and fairness assessment libraries together. This result raises concerns about the fairness assessment of LLMs.

- **Model Deployment.** We performed an extensive empirical study on the impact of LLM compression strategies on the efficiency and effectiveness of LLMs fine-tuned for SE tasks. Following this empirical evaluation, we provided a set of recommendations for practitioners and researchers to select the best compression strategy to employ during the *model deployment* phase. In addition, we presented a novel search-based approach named GreenStableYolo. This method searches for the best set of hyperparameters and prompt structures to improve the efficiency of text-to-image generation models while keeping the generated images of high quality. Thus, it can be employed during *model deployment* to identify the best model setting.
- **Model Monitoring.** We presented a comprehensive analysis of the *gender* and *ethnicity* bias exposed by Stable Diffusion text-to-image generation models when generating images for SE tasks. The results raise serious concerns about the bias exposed by those models. Moreover, we provide practitioners and researchers with recommendations to mitigate the bias exposed by those models when using them.

Although the contributions cover many phases of the pipeline, challenges remain. In particular, while approaches like MANILA, MODNESS, and the identified bias symptoms are presented separately, they can be integrated to automate and enhance the development of fair learning-based systems. Additionally, we envision extending the early detection of bias to the very beginning of the learning-based system development workflow, specifically during the model requirements phase. Finally, the empirical study on the bias exposed by image-generation models opens the field to additional research needed to mitigate the bias embedded in these models. At the same time, the empirical study on LLM compression methods underscores the importance of assisting practitioners in choosing the most suitable compression strategy according to the specific task and model architecture. In the following section, we detail future work proposed to address those current limitations.

12.1 Future Work

Future work of this thesis lies in additional contributions to the development of *fair* and *efficient* learning-based systems. More in detail:

- **Automate the development of fair learning-based systems.** As highlighted in Chapters 5 and 6, MANILA and MODNESS could be integrated in the future to guide data scientists through the development of fair learning-based systems, allowing a high degree of expressiveness. More in detail, we plan to extend the MODNESS metamodel by including the features and constraints specified in MANILA's ExtFM. Moreover, we plan to integrate the study on bias symptoms to suggest possible variables leading to high bias in the system. Furthermore, we will explore methods for early detection of ML model training times to support the constraint of training efficiency. Eventually, we envision a web-based application providing a low-code infrastructure to: *i)* provide a high-level definition of bias for a given domain; *ii)* highlighting variables in the dataset possibly leading to high bias in the system; *iii)* guide them in the selection of ML models and fairness-enhancing methods to benchmark; *iv)* assist them in selecting fairness metrics more compliant with the high-level definition of bias; *v)* generate the complete implementation of the fairness benchmarking workflow.

- **Early bias detection and mitigation from model requirements.** To additionally support data scientists in developing fair learning-based systems, we envision extending the abovementioned framework to automatically detect bias issues starting from model requirements. In particular, similarly to previous work [119], [331], we can rely on Natural Language Processing or AI-based approaches to extract all the topics related to fairness assessment from a software requirement (like sensitive variables, positive outcomes, or privileged and unprivileged groups). These topics can be employed to assist domain experts in the high-level definition of bias. In particular, we plan to create a repository of high-level bias definition templates using the MODNESS DSL. These templates will be tagged with specific fairness topics. Next, based on the fairness topics extracted from a fairness requirement, this repository will be queried to select more suitable templates for a given requirement.

- **Automatic identification of LLM compression strategies.** The effectiveness of LLM compression strategies, as highlighted in Section 11.1, is closely related to the specific task and the model architecture used. Therefore, we propose an automated approach to identify the compression strategy that strikes the best balance between high prediction accuracy, reduced inference time, and smaller model size. More in particular, we envision a search-based approach that explores different compression method configurations to identify the optimal one. The search-based algorithm will be similar to a search-based strategy proposed in a previous work that explores different ML model configurations to create the best ensemble model for defect prediction [332].

- **Energy and fairness improvement of Text-To-Image generation models.** As shown in Section 8.1, Stable Diffusion models exhibit a strong *gender* and *ethnicity* bias when generating figures for specific tasks. Thus, we plan to extend GreenStableYolo to improve the fairness of Stable Diffusion models by working on its hyperparameters and prompt structure. In detail, we will include in the fitness function the *gender* and *ethnicity* bias metrics from Chapter 8 to guide the search toward solutions that not only minimize inference time but also reduce bias related to gender and ethnicity, all while ensuring the quality of the generated images remains high. Additionally, following the same approach, we plan to extend the efficiency improvement by also optimizing CPU and GPU consumption of Stable Diffusion models relying on dedicated libraries for its measurement, like Codecarbon[1].

- **Trade-off analysis on fairness and efficiency.** Finally, while the trade-off between fairness and prediction's effectiveness has been widely studied [34], [35], [80], there is still a lack of research focusing on the trade-off between fairness and efficiency in learning-based systems. To address this gap, we plan to thoroughly investigate the trade-offs between fairness and efficiency in learning-based systems. Specifically, we aim to explore how fairness-enhancing methods impact the efficiency of learning-based systems, particularly regarding training and inference times, as well as energy consumption. From this analysis, we aim to propose a search-based strategy to automatically identify the optimal combination of the ML model and fairness-enhancing method able to achieve the best fairness, efficiency and effectiveness trade-offs.

[1] `https://codecarbon.io/`

Appendix A

Additional DEMV Evaluations

A.1 Detailed results of generative strategies' comparison

In the following, we report the detailed results of the evaluation of DEMV's generative strategies. For each dataset and for each method, we report the mean and standard deviation of all metrics. In addition, we report the mean and standard deviation of the H-Mean computed from the obtained values. Finally, we also report the overall means and standard deviations of all the values obtained by each method in each experiment. For each dataset, we highlight in boldface the best value of each metric

In particular, table A.1 shows the results for binary datasets, while table A.2 describes the results for multi-class datasets.

A.2 Detailed results for binary classification

In the following, we report the charts and the detailed results for binary classification. Concerning the experiment with one sensitive variable we report the mean of the measures of both experiments taking each sensitive variable singularly.

Figure A.1 reports the means and standard deviations obtained in all three experiments. As noticed above, EG is the method performing better when one or two sensitive variables are involved, while it is not able to manage groups identified by three sensitive variables.

Tables A.3, A.4, and A.5 reports the detailed results for each dataset. For each dataset, we highlight in boldface the best value of each metric whose differences are statistically significant. As mentioned above, we like to remark that when dealing

TABLE A.1: Evaluation results of generative strategies for binary datasets

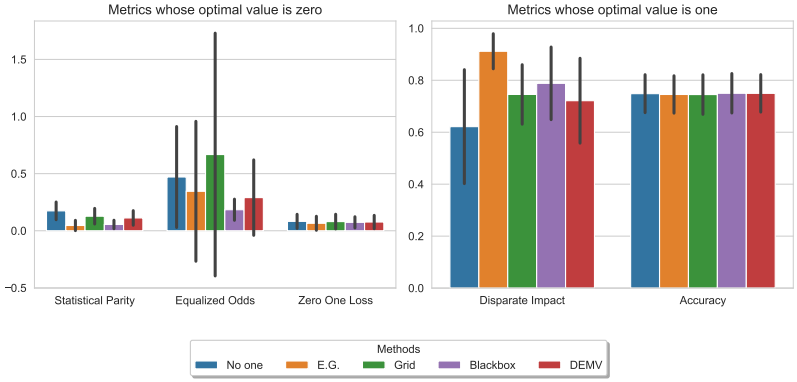
Data	Strategy	SP	AO	ZO Loss	DI	Acc	H-Mean
Adult	DEMV Uniform	**0.126 ± 0.03**	0.26 ± 0.124	**0.144 ± 0.015**	0.373 ± 0.137	**0.834 ± 0.005**	0.635 ± 0.117
	DEMV SMOTE	0.138 ± 0.014	**0.22 ± 0.149**	0.153 ± 0.011	0.242 ± 0.075	**0.834 ± 0.005**	0.543 ± 0.078
	DEMV ADASYN	**0.126 ± 0.03**	0.259 ± 0.123	0.145 ± 0.015	**0.374 ± 0.137**	**0.834 ± 0.005**	**0.636 ± 0.117**
Compas	DEMV Uniform	0.161 ± 0.063	0.29 ± 0.202	**0.124 ± 0.043**	0.773 ± 0.08	0.664 ± 0.016	0.75 ± 0.099
	DEMV SMOTE	**0.15 ± 0.043**	**0.266 ± 0.154**	0.133 ± 0.051	**0.79 ± 0.056**	**0.665 ± 0.016**	**0.767 ± 0.061**
	DEMV ADASYN	0.16 ± 0.063	0.288 ± 0.203	**0.124 ± 0.043**	0.774 ± 0.081	0.664 ± 0.016	0.75 ± 0.099
German	DEMV Uniform	**0.18 ± 0.134**	0.644 ± 0.342	0.278 ± 0.119	**0.772 ± 0.163**	**0.748 ± 0.038**	0.616 ± 0.157
	DEMV SMOTE	0.183 ± 0.138	**0.625 ± 0.381**	**0.276 ± 0.121**	0.771 ± 0.172	**0.748 ± 0.039**	**0.636 ± 0.162**
	DEMV ADASYN	0.181 ± 0.134	0.649 ± 0.353	**0.276 ± 0.12**	0.771 ± 0.163	0.747 ± 0.039	0.623 ± 0.146
Mean	DEMV Uniform	**0.156 ± 0.027**	0.398 ± 0.214	**0.182 ± 0.084**	0.639 ± 0.231	**0.749 ± 0.085**	0.667 ± 0.073
	DEMV SMOTE	0.157 ± 0.023	0.37 ± 0.222	0.187 ± 0.077	0.601 ± 0.311	**0.749 ± 0.085**	0.649 ± 0.113
	DEMV ADASYN	**0.156 ± 0.028**	**0.399 ± 0.217**	**0.182 ± 0.082**	**0.64 ± 0.23**	0.748 ± 0.085	**0.67 ± 0.07**

TABLE A.2: Evaluation results of generative strategies for multi-class datasets

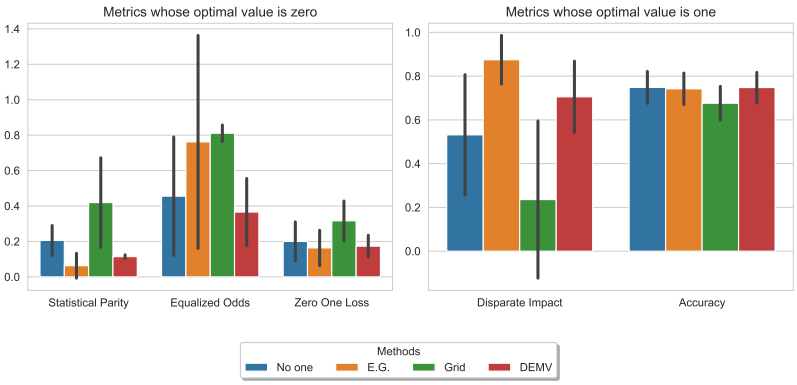
Data	Strategy	SP	AO	ZO Loss	DI	Acc	H-Mean
CMC	DEMV Uniform	0.056 ± 0.029	0.206 ± 0.191	**0.233 ± 0.102**	0.663 ± 0.157	0.512 ± 0.038	0.694 ± 0.074
	DEMV SMOTE	**0.048 ± 0.033**	**0.195 ± 0.138**	0.273 ± 0.098	**0.722 ± 0.138**	0.51 ± 0.038	**0.704 ± 0.051**
	DEMV ADASYN	0.054 ± 0.03	0.213 ± 0.172	0.255 ± 0.101	0.68 ± 0.168	**0.514 ± 0.038**	0.693 ± 0.07
Crime	DEMV Uniform	**0.202 ± 0.049**	0.32 ± 0.138	**0.164 ± 0.044**	**0.365 ± 0.143**	0.441 ± 0.028	**0.542 ± 0.076**
	DEMV SMOTE	0.242 ± 0.048	**0.309 ± 0.146**	0.181 ± 0.039	0.292 ± 0.124	**0.456 ± 0.03**	0.501 ± 0.08
	DEMV ADASYN	0.215 ± 0.051	0.316 ± 0.162	0.175 ± 0.054	0.271 ± 0.151	0.454 ± 0.033	0.476 ± 0.122
Drug	DEMV Uniform	**0.148 ± 0.047**	**0.17 ± 0.072**	0.337 ± 0.109	0.486 ± 0.13	0.675 ± 0.025	0.662 ± 0.062
	DEMV SMOTE	0.185 ± 0.056	0.184 ± 0.058	**0.328 ± 0.108**	0.41 ± 0.121	**0.68 ± 0.029**	0.624 ± 0.069
	DEMV ADASYN	0.134 ± 0.054	0.198 ± 0.085	0.345 ± 0.106	**0.519 ± 0.121**	0.671 ± 0.025	**0.67 ± 0.059**
Law	DEMV Uniform	**0.041 ± 0.028**	0.145 ± 0.063	0.159 ± 0.022	**0.887 ± 0.077**	0.512 ± 0.011	**0.77 ± 0.019**
	DEMV SMOTE	0.095 ± 0.031	0.144 ± 0.058	0.172 ± 0.021	0.741 ± 0.087	**0.515 ± 0.01**	0.737 ± 0.023
	DEMV ADASYN	0.044 ± 0.031	**0.14 ± 0.055**	**0.153 ± 0.014**	0.883 ± 0.08	0.511 ± 0.012	**0.77 ± 0.02**
Park	DEMV Uniform	0.062 ± 0.048	0.073 ± 0.046	**0.211 ± 0.047**	0.809 ± 0.136	0.493 ± 0.024	**0.746 ± 0.042**
	DEMV SMOTE	0.067 ± 0.049	0.085 ± 0.044	0.22 ± 0.048	0.796 ± 0.136	**0.496 ± 0.024**	0.742 ± 0.048
	DEMV ADASYN	**0.057 ± 0.032**	**0.071 ± 0.043**	**0.211 ± 0.052**	**0.812 ± 0.1**	0.478 ± 0.023	0.742 ± 0.036
Wine	DEMV Uniform	0.106 ± 0.038	0.478 ± 0.264	**0.078 ± 0.03**	0.858 ± 0.047	0.519 ± 0.018	0.646 ± 0.177
	DEMV SMOTE	0.174 ± 0.052	0.787 ± 0.369	0.138 ± 0.046	0.772 ± 0.062	**0.538 ± 0.016**	0.566 ± 0.154
	DEMV ADASYN	**0.096 ± 0.039**	**0.34 ± 0.215**	0.083 ± 0.028	**0.864 ± 0.053**	0.515 ± 0.018	**0.722 ± 0.073**
Mean	DEMV Uniform	0.102 ± 0.063	0.232 ± 0.145	**0.197 ± 0.087**	**0.678 ± 0.214**	**0.525 ± 0.079**	0.677 ± 0.081
	DEMV SMOTE	0.135 ± 0.077	0.284 ± 0.257	0.219 ± 0.071	0.622 ± 0.215	0.533 ± 0.077	0.646 ± 0.099
	DEMV ADASYN	**0.1 ± 0.066**	**0.213 ± 0.102**	0.204 ± 0.09	0.672 ± 0.239	0.524 ± 0.076	**0.679 ± 0.105**

TABLE A.3: Evaluation results for all binary datasets and methods with one sensitive variables

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
Adult	No one	0.139 ± 0.017	**0.104 ± 0.063**	0.094 ± 0.033	0.34 ± 0.062	0.835 ± 0.007	0.659 ± 0.054
	Blackbox	0.061 ± 0.021	0.253 ± 0.065	0.094 ± 0.033	0.659 ± 0.045	0.835 ± 0.007	0.802 ± 0.032
	EG	**0.02 ± 0.015**	0.238 ± 0.072	0.078 ± 0.035	**0.892 ± 0.079**	0.827 ± 0.008	**0.868 ± 0.026**
	Grid	0.066 ± 0.021	0.164 ± 0.074	0.09 ± 0.033	0.662 ± 0.11	0.833 ± 0.005	0.818 ± 0.039
	DEMV	0.094 ± 0.023	0.135 ± 0.062	0.093 ± 0.03	0.57 ± 0.098	0.834 ± 0.006	0.787 ± 0.044
Compas	No one	0.217 ± 0.067	0.759 ± 0.499	0.041 ± 0.033	0.727 ± 0.066	0.67 ± 0.019	0.61 ± 0.216
	Blackbox	0.064 ± 0.048	**0.112 ± 0.05**	0.041 ± 0.033	0.831 ± 0.114	0.67 ± 0.019	0.84 ± 0.037
	EG	**0.035 ± 0.027**	0.158 ± 0.08	0.035 ± 0.028	**0.947 ± 0.039**	0.662 ± 0.015	**0.857 ± 0.022**
	Grid	0.183 ± 0.044	0.442 ± 0.172	0.051 ± 0.036	0.731 ± 0.07	0.657 ± 0.023	0.706 ± 0.081
	DEMV	0.117 ± 0.053	0.212 ± 0.154	0.037 ± 0.028	0.832 ± 0.068	0.665 ± 0.018	0.807 ± 0.066
German	No one	0.166 ± 0.105	0.549 ± 0.359	0.112 ± 0.084	0.798 ± 0.125	0.741 ± 0.045	0.676 ± 0.19
	Blackbox	**0.025 ± 0.026**	**0.191 ± 0.108**	0.099 ± 0.069	**0.963 ± 0.038**	0.741 ± 0.028	**0.864 ± 0.025**
	EG	0.084 ± 0.056	0.641 ± 1.016	0.085 ± 0.091	0.897 ± 0.068	0.746 ± 0.042	0.78 ± 0.139
	Grid	0.133 ± 0.072	1.395 ± 1.629	0.1 ± 0.096	0.843 ± 0.083	0.746 ± 0.038	0.76 ± 0.175
	DEMV	0.119 ± 0.088	0.563 ± 0.419	0.098 ± 0.077	0.851 ± 0.102	0.748 ± 0.039	0.737 ± 0.112
Mean	No one	0.174 ± 0.04	0.471 ± 0.334	0.082 ± 0.037	0.622 ± 0.247	0.749 ± 0.083	0.648 ± 0.034
	Blackbox	0.05 ± 0.022	**0.185 ± 0.071**	0.078 ± 0.032	0.818 ± 0.152	0.749 ± 0.083	**0.835 ± 0.031**
	EG	**0.046 ± 0.033**	0.346 ± 0.259	0.066 ± 0.027	**0.912 ± 0.03**	0.745 ± 0.083	**0.835 ± 0.048**
	Grid	0.127 ± 0.059	0.667 ± 0.646	0.08 ± 0.026	0.745 ± 0.091	0.745 ± 0.088	0.761 ± 0.056
	DEMV	0.11 ± 0.014	0.303 ± 0.228	0.076 ± 0.034	0.751 ± 0.157	0.749 ± 0.085	0.777 ± 0.036



(A) Application with one sensitive variable



(B) Application with two sensitive variables

(C) Application with three sensitive variables

FIGURE A.1: Comparison of DEMV with the baselines in binary classification

with a binary dataset with one single sensitive variable, DEMV coincides with the *Sampling* method of [86].

Figure A.2 reports instead the overall mean and standard deviation of all the metrics computed in the experiments involving more complex classifiers. It can be seen how, differently from the experiments involving a Logistic Regression model, DEMV overcomes the other baselines in all the experiments, with the only exception of EO with SVM in which the best method is EG. As already said, we like to remark that EG and Grid can not be applied with a Neural Network model.

Finally, tables A.6, A.7, and A.8 reports the detailed results for each dataset in the experiments involving, respectively Gradient Boosting, SVM and Neural Network. For each dataset, we highlight in boldface the best value of each metric whose

TABLE A.4: Evaluation results for all binary datasets and methods with two sensitive variables

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
Adult	No one	0.17 ± 0.017	**0.17 ± 0.136**	0.156 ± 0.01	0.174 ± 0.071	**0.835 ± 0.007**	0.455 ± 0.107
	EG	**0.021 ± 0.012**	0.396 ± 0.101	**0.117 ± 0.019**	**0.871 ± 0.08**	0.82 ± 0.005	**0.805 ± 0.046**
	Grid	0.366 ± 0.007	0.52 ± 0.014	0.237 ± 0.012	0.0 ± 0.0	0.771 ± 0.005	0.0 ± 0.0
	DEMV	0.1 ± 0.021	0.284 ± 0.112	0.141 ± 0.015	0.475 ± 0.109	0.832 ± 0.004	0.706 ± 0.072
Compas	No one	0.241 ± 0.038	0.55 ± 0.212	0.127 ± 0.047	0.678 ± 0.045	**0.67 ± 0.019**	0.621 ± 0.13
	EG	**0.044 ± 0.027**	**0.161 ± 0.085**	**0.114 ± 0.048**	**0.932 ± 0.039**	0.644 ± 0.025	**0.833 ± 0.029**
	Grid	0.294 ± 0.235	0.396 ± 0.148	0.276 ± 0.064	0.593 ± 0.198	0.584 ± 0.016	0.592 ± 0.174
	DEMV	0.116 ± 0.048	0.185 ± 0.119	0.119 ± 0.042	0.831 ± 0.067	0.662 ± 0.016	0.802 ± 0.046
German	No one	0.206 ± 0.139	0.647 ± 0.41	0.317 ± 0.123	0.743 ± 0.173	0.741 ± 0.046	0.597 ± 0.187
	EG	0.116 ± 0.093	0.833 ± 0.764	0.264 ± 0.121	**0.86 ± 0.117**	**0.749 ± 0.039**	**0.687 ± 0.198**
	Grid	**0.691 ± 0.06**	0.811 ± 0.049	0.44 ± 0.108	0.0 ± 0.0	0.67 ± 0.024	0.0 ± 0.0
	DEMV	0.148 ± 0.131	**0.628 ± 0.373**	**0.26 ± 0.126**	0.81 ± 0.157	**0.749 ± 0.036**	0.662 ± 0.105
Mean	No one	0.206 ± 0.036	0.456 ± 0.252	0.2 ± 0.102	0.532 ± 0.311	**0.749 ± 0.083**	0.558 ± 0.09
	EG	**0.06 ± 0.05**	0.463 ± 0.341	**0.165 ± 0.086**	**0.888 ± 0.039**	0.738 ± 0.089	**0.775 ± 0.077**
	Grid	0.45 ± 0.212	0.576 ± 0.213	0.318 ± 0.108	0.198 ± 0.342	0.675 ± 0.094	0.197 ± 0.342
	DEMV	0.121 ± 0.024	**0.366 ± 0.233**	0.173 ± 0.076	0.705 ± 0.2	0.748 ± 0.085	0.723 ± 0.072

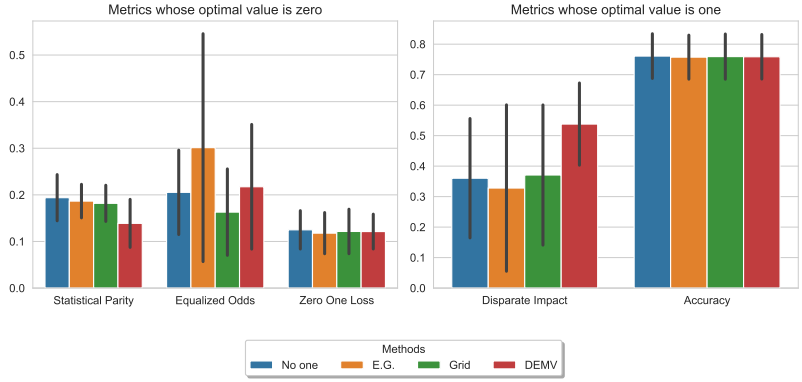
TABLE A.5: Evaluation results for all binary datasets and methods with three sensitive variables

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
Adult	No one	0.179 ± 0.016	**0.248 ± 0.189**	0.286 ± 0.077	0.124 ± 0.073	0.835 ± 0.007	0.352 ± 0.138
	EG	0.179 ± 0.012	0.259 ± 0.21	0.271 ± 0.063	0.128 ± 0.06	0.829 ± 0.007	0.367 ± 0.115
	Grid	0.35 ± 0.112	0.434 ± 0.099	0.312 ± 0.031	0.119 ± 0.251	0.757 ± 0.022	0.207 ± 0.221
	DEMV	**0.096 ± 0.023**	0.325 ± 0.15	0.31 ± 0.06	**0.465 ± 0.136**	0.821 ± 0.005	**0.658 ± 0.105**
Compas	No one	0.244 ± 0.045	0.612 ± 0.204	0.351 ± 0.092	0.683 ± 0.055	0.653 ± 0.02	0.56 ± 0.142
	EG	0.262 ± 0.038	0.666 ± 0.254	0.361 ± 0.082	0.661 ± 0.046	0.652 ± 0.02	0.468 ± 0.243
	Grid	0.215 ± 0.055	0.434 ± 0.141	0.339 ± 0.102	0.707 ± 0.073	0.641 ± 0.015	0.657 ± 0.069
	DEMV	**0.1 ± 0.045**	**0.215 ± 0.127**	0.323 ± 0.1	**0.861 ± 0.06**	0.648 ± 0.023	**0.758 ± 0.061**
German	No one	0.191 ± 0.073	0.851 ± 0.504	0.552 ± 0.135	**0.786 ± 0.08**	0.744 ± 0.042	0.542 ± 0.138
	EG	0.236 ± 0.118	0.554 ± 0.281	0.631 ± 0.131	0.717 ± 0.152	0.729 ± 0.033	0.527 ± 0.13
	Grid	**0.181 ± 0.154**	**0.404 ± 0.171**	0.565 ± 0.179	0.775 ± 0.191	0.694 ± 0.034	**0.593 ± 0.124**
	DEMV	0.188 ± 0.07	0.831 ± 0.476	0.512 ± 0.157	**0.786 ± 0.076**	0.747 ± 0.034	0.536 ± 0.18
Mean	No one	0.205 ± 0.035	0.57 ± 0.304	0.396 ± 0.139	0.531 ± 0.356	0.744 ± 0.091	0.485 ± 0.115
	EG	0.226 ± 0.042	0.493 ± 0.21	0.421 ± 0.187	0.502 ± 0.325	0.737 ± 0.089	0.454 ± 0.081
	Grid	0.249 ± 0.089	**0.424 ± 0.017**	0.405 ± 0.139	0.534 ± 0.361	0.697 ± 0.058	0.486 ± 0.243
	DEMV	**0.128 ± 0.052**	0.457 ± 0.329	0.382 ± 0.113	**0.704 ± 0.21**	0.739 ± 0.087	**0.651 ± 0.111**

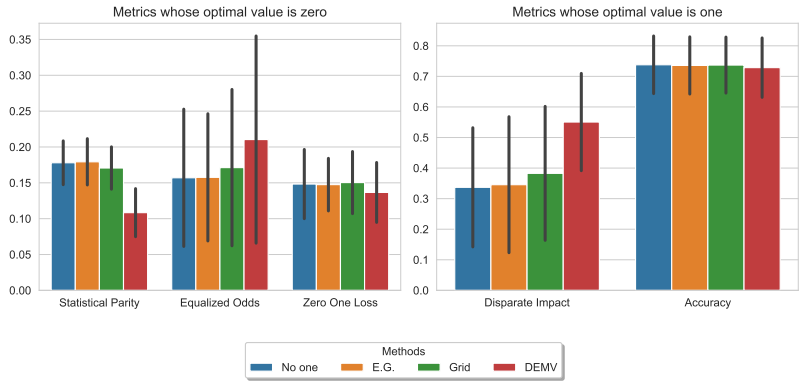
TABLE A.6: Evaluation results for binary datasets using Gradient Boostring classifier

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
Adult	No one	0.154 ± 0.017	0.225 ± 0.123	0.158 ± 0.009	0.175 ± 0.073	0.833 ± 0.006	0.454 ± 0.099
	EG	0.166 ± 0.013	0.438 ± 0.298	0.155 ± 0.012	0.062 ± 0.059	0.829 ± 0.006	0.205 ± 0.175
	Grid	0.156 ± 0.013	**0.16 ± 0.13**	0.16 ± 0.01	0.151 ± 0.067	0.832 ± 0.007	0.421 ± 0.098
	DEMV	**0.099 ± 0.022**	0.299 ± 0.144	0.145 ± 0.014	**0.443 ± 0.118**	0.831 ± 0.005	**0.684 ± 0.084**
Compas	No one	0.234 ± 0.039	0.185 ± 0.047	0.092 ± 0.035	0.546 ± 0.059	0.689 ± 0.019	0.722 ± 0.035
	EG	0.207 ± 0.042	0.164 ± 0.047	0.08 ± 0.033	0.594 ± 0.072	0.686 ± 0.018	0.746 ± 0.04
	Grid	0.208 ± 0.041	0.166 ± 0.046	0.083 ± 0.042	0.591 ± 0.072	0.686 ± 0.019	0.744 ± 0.038
	DEMV	**0.179 ± 0.04**	**0.136 ± 0.043**	0.098 ± 0.039	**0.632 ± 0.069**	0.687 ± 0.017	**0.765 ± 0.035**
Mean	No one	0.194 ± 0.057	0.205 ± 0.028	0.125 ± 0.047	0.361 ± 0.262	0.761 ± 0.102	0.588 ± 0.19
	EG	0.186 ± 0.029	0.301 ± 0.194	0.118 ± 0.053	0.328 ± 0.376	0.758 ± 0.101	0.476 ± 0.383
	Grid	0.182 ± 0.037	**0.163 ± 0.004**	0.122 ± 0.054	0.371 ± 0.311	0.759 ± 0.103	0.582 ± 0.228
	DEMV	**0.139 ± 0.057**	0.218 ± 0.115	0.122 ± 0.033	**0.538 ± 0.134**	0.759 ± 0.102	**0.724 ± 0.057**

differences are statistically significant.



(A) Application with Gradient Boosting



(B) Application with Support Vector Machines

(C) Application with Neural Networks

FIGURE A.2: Comparison of DEMV with the baselines in binary classification using other classifiers

A.3 Detailed results for multi-class classification

In the following we report the tables describing the detailed results of experiments involving multi-class datasets. For each dataset and for each method, we report the mean and standard deviation of all metrics. In addition, we report the mean and standard deviation of the H-Mean computed from the obtained values. Finally, we also report the overall means and standard deviations of all the values obtained by each method in each experiment. We split the results among experiments involving one, two, and three sensitive variables and experiments with more complex classifiers. For each dataset, we highlight in boldface the best value of each metric whose differences are statistically significant.

TABLE A.7: Evaluation results for binary datasets using Support Vector Machines classifier

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
Adult	No one	0.165 ± 0.013	0.185 ± 0.133	0.167 ± 0.009	0.151 ± 0.042	0.831 ± 0.005	0.428 ± 0.074
	EG	0.162 ± 0.014	0.175 ± 0.126	0.163 ± 0.013	0.132 ± 0.057	0.828 ± 0.006	0.386 ± 0.124
	Grid	0.162 ± 0.021	0.208 ± 0.146	0.162 ± 0.017	0.178 ± 0.092	0.828 ± 0.006	0.445 ± 0.155
	DEMV	**0.096 ± 0.021**	0.298 ± 0.155	0.151 ± 0.017	**0.431 ± 0.116**	0.825 ± 0.006	**0.674 ± 0.083**
Compas	No one	0.191 ± 0.039	0.129 ± 0.029	0.129 ± 0.066	0.523 ± 0.074	0.645 ± 0.021	0.712 ± 0.037
	EG	0.196 ± 0.039	0.14 ± 0.032	0.132 ± 0.048	0.559 ± 0.073	0.643 ± 0.02	0.722 ± 0.033
	Grid	0.179 ± 0.037	0.134 ± 0.044	0.139 ± 0.06	0.588 ± 0.072	0.646 ± 0.019	0.735 ± 0.036
	DEMV	**0.121 ± 0.039**	0.123 ± 0.049	0.122 ± 0.053	**0.67 ± 0.094**	0.632 ± 0.019	**0.768 ± 0.038**
Mean	No one	0.178 ± 0.018	0.157 ± 0.04	0.148 ± 0.027	0.337 ± 0.263	0.738 ± 0.132	0.57 ± 0.201
	EG	0.179 ± 0.024	0.158 ± 0.025	0.148 ± 0.022	0.346 ± 0.302	0.736 ± 0.131	0.554 ± 0.238
	Grid	0.17 ± 0.012	0.171 ± 0.052	0.151 ± 0.016	0.383 ± 0.29	0.737 ± 0.129	0.59 ± 0.205
	DEMV	**0.108 ± 0.018**	0.21 ± 0.124	0.136 ± 0.021	**0.55 ± 0.169**	0.728 ± 0.136	**0.721 ± 0.066**

TABLE A.8: Evaluation results for binary datasets using Neural Network classifier

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
Adult	No one	0.185 ± 0.031	0.23 ± 0.174	0.17 ± 0.013	0.17 ± 0.074	0.819 ± 0.008	0.441 ± 0.133
	EG			Not applicable			
	Grid			Not applicable			
	DEMV	**0.144 ± 0.028**	0.24 ± 0.136	0.16 ± 0.017	**0.315 ± 0.101**	0.815 ± 0.006	**0.6 ± 0.096**
Compas	No one	0.214 ± 0.053	0.193 ± 0.075	0.089 ± 0.061	0.583 ± 0.068	0.652 ± 0.017	0.727 ± 0.045
	EG			Not applicable			
	Grid			Not applicable			
	DEMV	**0.136 ± 0.046**	0.15 ± 0.053	0.12 ± 0.059	**0.719 ± 0.077**	0.651 ± 0.017	**0.779 ± 0.035**
Mean	No one	0.2 ± 0.021	0.212 ± 0.026	0.13 ± 0.057	0.376 ± 0.292	0.736 ± 0.118	0.584 ± 0.202
	EG			Not applicable			
	Grid			Not applicable			
	DEMV	**0.14 ± 0.006**	0.195 ± 0.064	0.14 ± 0.028	**0.517 ± 0.286**	0.733 ± 0.116	**0.69 ± 0.127**

In particular, table A.9 reports the results of experiments involving one sensitive variable, table A.10 reports the results of experiments with two sensitive variables, and table A.11 shows the results of experiments with three sensitive variables.

Finally, tables A.12, A.13, and A.14 reports the detailed results for each dataset of the experiments involving respectively Gradient Boosting, SVM, and Neural Networks.

A.4 ANOVA tables

In the following, we report the ANOVA tables of our experiments. In particular, table A.15 shows the results for binary, and table A.16 reports the results for multi-class experiments involving sensitive groups identified by a different number of sensitive variables. Tables A.17 and A.18 reports instead the results of the ANOVA tests involving more complex classifiers for respectively binary and multi-class classification. We recall that, in order to be statistically significant the probability value (*p-value*) most be lower than 0.05. In this case, the test rejects the null hypothesis of equal mean for all groups.

TABLE A.9: Evaluation results for all multi-class datasets and methods using one sensitive variables

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
CMC	No one	0.188 ± 0.15	0.305 ± 0.231	0.122 ± 0.081	0.51 ± 0.282	0.521 ± 0.039	0.6 ± 0.166
	Blackbox	0.125 ± 0.1	0.275 ± 0.169	0.098 ± 0.08	0.539 ± 0.308	0.515 ± 0.042	0.606 ± 0.185
	EG	0.162 ± 0.128	0.292 ± 0.193	0.111 ± 0.074	0.536 ± 0.279	0.505 ± 0.038	0.62 ± 0.138
	Grid	0.089 ± 0.086	0.278 ± 0.231	0.105 ± 0.057	**0.748 ± 0.154**	0.501 ± 0.04	**0.699 ± 0.131**
	DEMV	**0.088 ± 0.072**	0.254 ± 0.146	**0.092 ± 0.072**	0.641 ± 0.224	0.516 ± 0.038	0.687 ± 0.107
Crime	No one	0.389 ± 0.082	0.329 ± 0.113	0.069 ± 0.049	0.182 ± 0.076	0.497 ± 0.028	0.409 ± 0.109
	Blackbox	0.425 ± 0.074	0.884 ± 0.314	0.069 ± 0.049	0.097 ± 0.082	0.497 ± 0.028	0.186 ± 0.118
	EG	0.39 ± 0.084	0.332 ± 0.112	0.066 ± 0.05	0.179 ± 0.077	0.496 ± 0.03	0.403 ± 0.118
	Grid	0.3 ± 0.111	0.399 ± 0.135	0.117 ± 0.06	0.336 ± 0.182	0.433 ± 0.039	0.487 ± 0.124
	DEMV	**0.253 ± 0.064**	**0.317 ± 0.109**	**0.062 ± 0.034**	0.377 ± 0.106	0.47 ± 0.029	**0.568 ± 0.063**
Drug	No one	0.264 ± 0.121	0.308 ± 0.236	0.142 ± 0.076	0.343 ± 0.216	0.68 ± 0.025	0.542 ± 0.183
	Blackbox	0.441 ± 0.144	0.806 ± 0.58	0.145 ± 0.073	0.095 ± 0.047	0.683 ± 0.025	0.268 ± 0.087
	EG	0.246 ± 0.107	0.267 ± 0.135	0.137 ± 0.093	0.371 ± 0.193	0.68 ± 0.026	0.583 ± 0.153
	Grid	0.26 ± 0.117	0.298 ± 0.245	0.134 ± 0.091	0.336 ± 0.201	0.683 ± 0.025	0.541 ± 0.189
	DEMV	**0.128 ± 0.083**	0.218 ± 0.112	**0.126 ± 0.058**	**0.585 ± 0.199**	0.678 ± 0.026	**0.72 ± 0.091**
Law	No one	0.26 ± 0.04	0.31 ± 0.038	0.072 ± 0.04	0.441 ± 0.128	0.521 ± 0.01	0.61 ± 0.062
	Blackbox	0.179 ± 0.035	0.231 ± 0.093	0.072 ± 0.04	0.408 ± 0.206	0.521 ± 0.01	0.584 ± 0.119
	EG	0.231 ± 0.048	0.264 ± 0.044	0.072 ± 0.037	0.487 ± 0.134	0.521 ± 0.009	0.64 ± 0.064
	Grid	0.176 ± 0.133	0.228 ± 0.149	0.104 ± 0.093	0.626 ± 0.289	0.503 ± 0.013	0.67 ± 0.135
	DEMV	**0.103 ± 0.024**	0.126 ± 0.039	**0.06 ± 0.03**	**0.757 ± 0.056**	0.518 ± 0.011	**0.76 ± 0.02**
Park	No one	0.221 ± 0.042	0.207 ± 0.055	**0.084 ± 0.064**	0.473 ± 0.075	0.503 ± 0.029	0.643 ± 0.046
	Blackbox	0.21 ± 0.092	0.381 ± 0.2	0.087 ± 0.057	0.334 ± 0.164	0.504 ± 0.024	0.496 ± 0.148
	EG	0.216 ± 0.051	0.23 ± 0.072	0.171 ± 0.079	0.461 ± 0.091	0.49 ± 0.022	0.622 ± 0.056
	Grid	0.221 ± 0.056	0.228 ± 0.05	0.172 ± 0.077	0.454 ± 0.11	0.493 ± 0.024	0.619 ± 0.064
	DEMV	**0.111 ± 0.054**	0.157 ± 0.043	0.085 ± 0.05	**0.697 ± 0.121**	0.502 ± 0.022	**0.728 ± 0.037**
Wine	No one	0.342 ± 0.165	1.067 ± 0.734	0.043 ± 0.033	0.544 ± 0.172	0.56 ± 0.02	0.607 ± 0.133
	Blackbox	**0.056 ± 0.043**	0.192 ± 0.132	0.048 ± 0.029	0.675 ± 0.223	0.561 ± 0.02	**0.735 ± 0.123**
	EG	0.338 ± 0.17	1.075 ± 0.756	0.043 ± 0.036	0.552 ± 0.179	0.56 ± 0.019	0.624 ± 0.104
	Grid	0.363 ± 0.206	0.761 ± 0.233	0.204 ± 0.184	0.453 ± 0.346	0.498 ± 0.043	0.38 ± 0.196
	DEMV	0.195 ± 0.078	0.84 ± 0.642	**0.033 ± 0.025**	**0.737 ± 0.077**	0.545 ± 0.022	0.628 ± 0.186
Mean	No one	0.277 ± 0.075	0.421 ± 0.319	0.089 ± 0.037	0.416 ± 0.134	0.547 ± 0.069	0.568 ± 0.085
	Blackbox	0.239 ± 0.159	0.462 ± 0.305	0.087 ± 0.033	0.358 ± 0.234	0.547 ± 0.07	0.479 ± 0.211
	EG	0.264 ± 0.084	0.41 ± 0.328	0.1 ± 0.048	0.431 ± 0.139	0.542 ± 0.072	0.582 ± 0.09
	Grid	0.235 ± 0.096	0.365 ± 0.204	0.139 ± 0.041	0.492 ± 0.164	0.518 ± 0.085	0.566 ± 0.121
	DEMV	**0.146 ± 0.064**	0.319 ± 0.264	**0.076 ± 0.032**	**0.632 ± 0.14**	0.538 ± 0.073	**0.682 ± 0.072**

TABLE A.10: Evaluation results for all multi-class datasets and methods using two sensitive variables

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
CMC	No one	0.126 ± 0.034	0.219 ± 0.118	0.33 ± 0.155	0.494 ± 0.128	**0.521 ± 0.04**	0.62 ± 0.058
	EG	0.107 ± 0.045	0.218 ± 0.15	0.35 ± 0.171	0.543 ± 0.173	0.509 ± 0.035	0.617 ± 0.081
	Grid	0.079 ± 0.049	0.241 ± 0.109	0.26 ± 0.176	0.815 ± 0.115	0.445 ± 0.049	0.679 ± 0.062
	DEMV	**0.056 ± 0.029**	**0.206 ± 0.191**	**0.233 ± 0.102**	**0.663 ± 0.157**	0.512 ± 0.038	**0.694 ± 0.074**
Crime	No one	0.339 ± 0.051	0.442 ± 0.139	0.209 ± 0.07	0.09 ± 0.066	**0.497 ± 0.029**	0.261 ± 0.108
	EG	0.332 ± 0.052	0.458 ± 0.166	0.212 ± 0.084	0.091 ± 0.074	0.493 ± 0.029	0.252 ± 0.139
	Grid	0.217 ± 0.077	0.335 ± 0.091	0.318 ± 0.077	**0.445 ± 0.136**	0.34 ± 0.042	0.515 ± 0.039
	DEMV	**0.202 ± 0.049**	**0.32 ± 0.138**	**0.164 ± 0.044**	0.365 ± 0.143	0.441 ± 0.028	**0.542 ± 0.076**
Drug	No one	0.299 ± 0.055	0.319 ± 0.15	0.335 ± 0.103	0.142 ± 0.086	0.68 ± 0.026	0.357 ± 0.144
	EG	0.272 ± 0.047	0.23 ± 0.118	0.375 ± 0.087	0.198 ± 0.068	**0.681 ± 0.032**	0.448 ± 0.073
	Grid	0.198 ± 0.057	0.193 ± 0.073	**0.331 ± 0.101**	0.356 ± 0.182	0.653 ± 0.025	0.574 ± 0.104
	DEMV	**0.148 ± 0.047**	**0.17 ± 0.072**	0.337 ± 0.109	**0.486 ± 0.13**	0.675 ± 0.025	**0.662 ± 0.062**
Law	No one	0.2 ± 0.027	0.2 ± 0.029	0.164 ± 0.03	0.502 ± 0.072	**0.521 ± 0.01**	0.655 ± 0.033
	EG	0.248 ± 0.031	0.308 ± 0.043	0.184 ± 0.027	0.456 ± 0.076	0.509 ± 0.013	0.61 ± 0.033
	Grid	0.3 ± 0.057	0.359 ± 0.09	0.193 ± 0.026	0.351 ± 0.086	0.508 ± 0.013	0.546 ± 0.067
	DEMV	**0.041 ± 0.028**	**0.145 ± 0.063**	**0.159 ± 0.022**	**0.887 ± 0.077**	0.512 ± 0.011	**0.77 ± 0.019**
Park	No one	0.208 ± 0.041	0.218 ± 0.072	0.246 ± 0.067	0.424 ± 0.089	**0.503 ± 0.026**	0.603 ± 0.05
	EG	0.272 ± 0.03	0.216 ± 0.041	0.324 ± 0.107	0.185 ± 0.044	0.481 ± 0.012	0.424 ± 0.053
	Grid	0.125 ± 0.031	0.127 ± 0.038	0.446 ± 0.063	0.617 ± 0.081	0.463 ± 0.019	0.631 ± 0.025
	DEMV	**0.062 ± 0.048**	**0.073 ± 0.046**	**0.211 ± 0.047**	**0.809 ± 0.136**	0.493 ± 0.024	**0.746 ± 0.042**
Wine	No one	0.322 ± 0.039	0.768 ± 0.185	0.146 ± 0.053	0.534 ± 0.048	**0.56 ± 0.021**	0.461 ± 0.117
	EG	0.189 ± 0.066	**0.35 ± 0.163**	0.178 ± 0.08	0.692 ± 0.096	0.541 ± 0.021	0.649 ± 0.066
	Grid	0.153 ± 0.042	0.616 ± 0.234	0.425 ± 0.055	0.77 ± 0.049	0.435 ± 0.021	0.535 ± 0.085
	DEMV	**0.106 ± 0.038**	0.478 ± 0.264	**0.078 ± 0.03**	**0.858 ± 0.047**	0.519 ± 0.018	**0.676 ± 0.177**
Mean	No one	0.249 ± 0.084	0.361 ± 0.219	0.238 ± 0.081	0.364 ± 0.196	**0.547 ± 0.069**	0.493 ± 0.16
	EG	0.237 ± 0.078	0.297 ± 0.096	0.27 ± 0.089	0.361 ± 0.238	0.536 ± 0.074	0.505 ± 0.16
	Grid	0.179 ± 0.078	0.312 ± 0.172	0.329 ± 0.096	0.559 ± 0.205	0.474 ± 0.104	0.58 ± 0.063
	DEMV	**0.102 ± 0.063**	**0.232 ± 0.145**	**0.197 ± 0.087**	**0.678 ± 0.214**	0.525 ± 0.079	**0.677 ± 0.081**

TABLE A.11: Evaluation results for all multi-class datasets and methods using three sensitive variables

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
CMC	No one	0.148 ± 0.039	0.283 ± 0.141	0.305 ± 0.143	0.353 ± 0.12	**0.497 ± 0.042**	0.54 ± 0.095
	EG	0.134 ± 0.047	0.27 ± 0.108	**0.346 ± 0.114**	0.427 ± 0.141	0.489 ± 0.038	0.574 ± 0.073
	Grid	0.065 ± 0.057	0.237 ± 0.116	0.277 ± 0.196	0.854 ± 0.128	0.432 ± 0.043	**0.673 ± 0.066**
	DEMV	**0.031 ± 0.019**	0.326 ± 0.235	0.274 ± 0.122	0.695 ± 0.189	0.489 ± 0.036	0.656 ± 0.112
Crime	No one	0.267 ± 0.066	0.435 ± 0.152	0.371 ± 0.097	0.176 ± 0.161	**0.504 ± 0.035**	0.336 ± 0.203
	EG	0.258 ± 0.072	0.459 ± 0.215	0.413 ± 0.128	0.171 ± 0.17	0.493 ± 0.04	0.307 ± 0.242
	Grid	**0.141 ± 0.058**	0.539 ± 0.229	0.381 ± 0.074	0.159 ± 0.178	0.309 ± 0.028	0.218 ± 0.233
	DEMV	0.149 ± 0.074	0.291 ± 0.133	**0.349 ± 0.048**	0.498 ± 0.226	0.437 ± 0.03	**0.571 ± 0.096**
Drug	No one	0.299 ± 0.045	0.293 ± 0.152	0.331 ± 0.099	0.144 ± 0.086	0.67 ± 0.029	0.36 ± 0.142
	EG	0.286 ± 0.056	0.236 ± 0.124	0.366 ± 0.067	0.172 ± 0.054	**0.671 ± 0.042**	0.419 ± 0.065
	Grid	0.207 ± 0.038	0.278 ± 0.14	**0.295 ± 0.092**	0.338 ± 0.155	0.64 ± 0.025	0.546 ± 0.147
	DEMV	**0.142 ± 0.055**	0.178 ± 0.068	0.362 ± 0.093	0.504 ± 0.169	0.66 ± 0.033	**0.659 ± 0.072**
Law	No one	0.2 ± 0.027	0.201 ± 0.028	0.165 ± 0.03	0.502 ± 0.071	**0.52 ± 0.01**	0.655 ± 0.032
	EG	0.225 ± 0.03	0.238 ± 0.032	0.172 ± 0.031	0.457 ± 0.072	0.517 ± 0.012	0.628 ± 0.034
	Grid	0.278 ± 0.091	0.359 ± 0.116	0.189 ± 0.028	0.408 ± 0.189	0.505 ± 0.016	0.566 ± 0.089
	DEMV	**0.042 ± 0.029**	0.144 ± 0.064	**0.159 ± 0.022**	0.885 ± 0.078	0.512 ± 0.011	**0.769 ± 0.019**
Wine	No one	0.434 ± 0.049	1.513 ± 0.308	0.163 ± 0.07	0.448 ± 0.049	**0.546 ± 0.019**	0.538 ± 0.063
	EG	0.419 ± 0.049	1.453 ± 0.294	0.169 ± 0.07	0.463 ± 0.051	0.541 ± 0.018	0.524 ± 0.071
	Grid	**0.057 ± 0.043**	0.101 ± 0.045	0.429 ± 0.063	0.76 ± 0.125	0.398 ± 0.022	**0.642 ± 0.041**
	DEMV	0.097 ± 0.04	0.593 ± 0.287	**0.109 ± 0.048**	0.877 ± 0.051	0.508 ± 0.02	0.577 ± 0.192
Mean	No one	0.27 ± 0.109	0.545 ± 0.548	0.267 ± 0.097	0.325 ± 0.16	**0.547 ± 0.071**	0.486 ± 0.135
	EG	0.264 ± 0.104	0.531 ± 0.524	0.293 ± 0.115	0.338 ± 0.153	0.542 ± 0.075	0.49 ± 0.128
	Grid	0.15 ± 0.094	0.303 ± 0.162	0.314 ± 0.094	0.504 ± 0.293	0.457 ± 0.124	0.529 ± 0.182
	DEMV	**0.092 ± 0.055**	0.306 ± 0.177	**0.251 ± 0.113**	0.692 ± 0.19	0.521 ± 0.083	**0.646 ± 0.08**

TABLE A.12: Evaluation results for multi-class datasets using Gradient Boosting classifier

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
CMC	No one	0.09 ± 0.053	0.178 ± 0.107	0.279 ± 0.127	0.656 ± 0.177	0.557 ± 0.04	0.696 ± 0.062
	EG	0.095 ± 0.068	**0.183 ± 0.119**	0.309 ± 0.153	0.658 ± 0.221	0.546 ± 0.039	0.685 ± 0.086
	Grid	0.065 ± 0.043	0.194 ± 0.091	0.195 ± 0.077	**0.742 ± 0.138**	0.443 ± 0.042	0.693 ± 0.047
	DEMV	**0.056 ± 0.04**	0.192 ± 0.139	0.272 ± 0.146	0.74 ± 0.17	**0.559 ± 0.042**	**0.716 ± 0.061**
Law	No one	0.232 ± 0.03	0.221 ± 0.035	0.175 ± 0.025	0.405 ± 0.082	**0.536 ± 0.01**	0.61 ± 0.045
	EG	**0.071 ± 0.053**	0.167 ± 0.072	0.154 ± 0.026	**0.809 ± 0.142**	0.527 ± 0.008	**0.754 ± 0.039**
	Grid	0.322 ± 0.071	0.433 ± 0.049	0.161 ± 0.025	0.344 ± 0.157	0.512 ± 0.01	0.522 ± 0.063
	DEMV	0.091 ± 0.029	**0.15 ± 0.065**	0.156 ± 0.02	0.739 ± 0.088	0.526 ± 0.008	0.742 ± 0.025
Mean	No one	0.161 ± 0.1	0.2 ± 0.03	0.227 ± 0.074	0.53 ± 0.177	**0.546 ± 0.015**	0.653 ± 0.061
	EG	0.083 ± 0.017	0.175 ± 0.011	0.231 ± 0.11	0.734 ± 0.107	0.536 ± 0.013	0.72 ± 0.049
	Grid	0.194 ± 0.182	0.314 ± 0.169	0.178 ± 0.024	0.543 ± 0.281	0.478 ± 0.049	0.607 ± 0.121
	DEMV	**0.074 ± 0.025**	**0.171 ± 0.03**	0.214 ± 0.082	**0.74 ± 0.001**	0.542 ± 0.023	**0.729 ± 0.018**

TABLE A.13: Evaluation results for multi-class datasets using Support Vector Machines classifier

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
CMC	No one	0.105 ± 0.046	0.174 ± 0.119	0.321 ± 0.18	0.574 ± 0.17	0.543 ± 0.046	0.652 ± 0.077
	EG	0.109 ± 0.044	**0.16 ± 0.071**	0.337 ± 0.158	0.549 ± 0.142	**0.546 ± 0.045**	0.652 ± 0.067
	Grid	0.197 ± 0.068	0.273 ± 0.083	0.295 ± 0.191	0.197 ± 0.22	0.435 ± 0.045	0.302 ± 0.261
	DEMV	**0.047 ± 0.03**	0.218 ± 0.164	0.279 ± 0.128	**0.73 ± 0.153**	**0.546 ± 0.042**	**0.707 ± 0.062**
Law	No one	0.267 ± 0.022	0.241 ± 0.019	0.173 ± 0.031	0.311 ± 0.048	**0.533 ± 0.011**	0.554 ± 0.035
	EG	0.234 ± 0.022	0.207 ± 0.04	0.192 ± 0.03	0.343 ± 0.063	0.525 ± 0.01	0.575 ± 0.044
	Grid	0.375 ± 0.024	0.492 ± 0.039	0.159 ± 0.019	0.277 ± 0.04	0.511 ± 0.013	0.482 ± 0.033
	DEMV	**0.116 ± 0.034**	**0.134 ± 0.042**	0.161 ± 0.02	**0.67 ± 0.099**	0.523 ± 0.01	**0.724 ± 0.031**
Mean	No one	0.186 ± 0.115	0.208 ± 0.047	0.247 ± 0.105	0.442 ± 0.186	**0.538 ± 0.007**	0.603 ± 0.069
	EG	0.172 ± 0.088	0.184 ± 0.033	0.264 ± 0.103	0.446 ± 0.146	0.536 ± 0.015	0.613 ± 0.054
	Grid	0.286 ± 0.126	0.382 ± 0.155	0.227 ± 0.096	0.237 ± 0.057	0.473 ± 0.054	0.392 ± 0.127
	DEMV	**0.082 ± 0.049**	**0.176 ± 0.059**	0.22 ± 0.083	**0.7 ± 0.042**	0.534 ± 0.016	**0.716 ± 0.012**

TABLE A.14: Evaluation results for multi-class datasets using Neural Network classifier

Data	Method	SP	AO	ZO Loss	DI	Acc	H-Mean
CMC	No one	0.081 ± 0.087	0.149 ± 0.104	0.338 ± 0.195	0.702 ± 0.261	0.542 ± 0.053	0.683 ± 0.098
	EG	Not applicable					
	Grid	Not applicable					
	DEMV	**0.06 ± 0.059**	0.17 ± 0.111	0.293 ± 0.135	**0.756 ± 0.171**	**0.544 ± 0.048**	**0.717 ± 0.071**
Law	No one	0.218 ± 0.04	0.197 ± 0.039	0.168 ± 0.03	0.436 ± 0.085	**0.531 ± 0.01**	0.629 ± 0.047
	EG	Not applicable					
	Grid	Not applicable					
	DEMV	**0.096 ± 0.044**	0.138 ± 0.06	0.125 ± 0.032	**0.721 ± 0.129**	0.519 ± 0.01	**0.739 ± 0.04**
Mean	No one	0.15 ± 0.097	0.173 ± 0.034	0.253 ± 0.12	0.569 ± 0.188	**0.536 ± 0.008**	0.656 ± 0.038
	EG	Not applicable					
	Grid	Not applicable					
	DEMV	**0.078 ± 0.025**	0.154 ± 0.023	0.209 ± 0.119	**0.738 ± 0.025**	0.532 ± 0.018	**0.728 ± 0.016**

TABLE A.15: ANOVA tables for binary datasets

(A) One sensitive variable

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	4.0	153.413	38.353	40.894	0.0
Residual	2405.0	2255.587	0.938		
Equalized Odds					
C(method)	4.0	57.828	14.457	12.001	0.0
Residual	670.0	807.154	1.205		
Zero-one Loss					
C(method)	4.0	2.999	0.75	0.749	0.558
Residual	2405.0	2406.001	1.00		
Disparate Impact					
C(method)	4.0	108.786	27.197	28.436	0.0
Residual	2405.0	2300.214	0.956		
Accuracy					
C(method)	4.0	0.453	0.113	0.113	0.978
Residual	2405.0	2408.547	1.001		
H-Mean					
C(method)	3.0	1.378	0.459	24.349	0.0
Residual	366.0	6.906	0.019		

(B) Two sensitive variables

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	3.0	42.582	14.194	25.563	0.0
Residual	89.0	49.418	0.555		
Equalized Odds					
C(method)	3.0	8.884	2.961	3.365	0.026
Residual	49.0	43.116	0.880		
Zero One Loss					
C(method)	3.0	24.807	8.269	10.953	0.0
Residual	89.0	67.193	0.755		
Disparate Impact					
C(method)	3.0	44.572	14.857	27.881	0.0
Residual	89.0	47.428	0.533		
Accuracy					
C(method)	3.0	14.831	4.944	5.702	0.001
Residual	89.0	77.169	0.867		
H-Mean					
C(method)	3.0	6.423	2.141	77.032	0.0
Residual	276.0	7.671	0.028		

(C) Three sensitive variables

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	3.0	9.643	3.214	3.474	0.019
Residual	89.0	82.357	0.925		
Equalized Odds					
C(method)	3.0	19.087	6.362	4.388	0.01
Residual	38.0	55.102	1.450		
Zero One Loss					
C(method)	3.0	0.432	0.144	0.14	0.936
Residual	89.0	91.568	1.029		
Disparate Impact					
C(method)	3.0	1.196	0.399	0.391	0.76
Residual	89.0	90.804	1.020		
Accuracy					
C(method)	3.0	7.084	2.361	2.475	0.067
Residual	89.0	84.916	0.954		
H-Mean					
C(method)	3.0	1.038	0.346	12.147	0.0
Residual	276.0	7.858	0.028		

TABLE A.16: ANOVA tables for multi-class datasets

(A) One sensitive variable

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	4.0	7.402	1.850	1.86	0.016
Residual	651.0	647.598	0.995		
Equalized Odds					
C(method)	4.0	13.725	3.431	1.326	0.262
Residual	184.0	476.038	2.587		
Zero-one Loss					
C(method)	4.0	50.71	12.678	13.657	0.0
Residual	651.0	604.29	0.928		
Disparate Impact					
C(method)	4.0	19.447	4.862	4.98	0.001
Residual	651.0	635.553	0.976		
Accuracy					
C(method)	4.0	4.338	1.084	1.085	0.363
Residual	651.0	650.662	0.999		
H-Mean					
C(method)	3.0	0.628	0.209	7.547	0.0
Residual	926.0	25.670	0.028		

(B) Two sensitive variables

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	4.0	104.788	26.197	39.255	0.0
Residual	303.0	202.212	0.667		
Equalized Odds					
C(method)	4.0	19.262	4.816	3.127	0.018
Residual	112.0	172.494	1.540		
Zero One Loss					
C(method)	4.0	29.399	7.350	8.022	0.0
Residual	303.0	277.601	0.916		
Disparate Impact					
C(method)	4.0	18.98	4.745	4.992	0.001
Residual	303.0	288.02	0.951		
Accuracy					
C(method)	4.0	17.356	4.339	4.539	0.001
Residual	303.0	289.644	0.956		
H-Mean					
C(method)	3.0	1.243	0.414	16.59	0.0
Residual	686.0	17.130	0.025		

(C) Three sensitive variables

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	3.0	7.6	2.533	2.582	0.054
Residual	243.0	238.4	0.981		
Equalized Odds					
C(method)	3.0	9.151	3.050	1.333	0.27
Residual	73.0	167.103	2.289		
Zero One Loss					
C(method)	3.0	24.773	8.258	9.07	0.0
Residual	243.0	221.227	0.910		
Disparate Impact					
C(method)	3.0	7.054	2.351	2.391	0.069
Residual	243.0	238.946	0.983		
Accuracy					
C(method)	3.0	21.399	7.133	7.717	0.0
Residual	243.0	224.601	0.924		
H-Mean					
C(method)	3.0	0.572	0.191	6.921	0.0
Residual	586.0	16.132	0.028		

TABLE A.17: ANOVA tables of binary experiments with other classifiers

(A) Gradient Boosting

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	3.0	46.935	15.645	16.768	0.0
Residual	656.0	612.065	0.933		
Equalized Odds					
C(method)	3.0	10.709	3.570	3.612	0.013
Residual	656.0	648.291	0.988		
Zero-one Loss					
C(method)	3.0	3074.488	1024.829	2.16	0.092
Residual	385.0	182633.494	474.373		
Disparate Impact					
C(method)	3.0	77.001	25.667	28.931	0.0
Residual	656.0	581.999	0.887		
Accuracy					
C(method)	3.0	0.026	0.009	0.009	0.999
Residual	656.0	658.974	1.005		
H-Mean					
C(method)	3.0	0.385	0.128	46.927	0.0
Residual	656.0	1.793	0.003		

(B) Support Vector Machines

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	3.0	167.808	55.936	74.704	0.0
Residual	656.0	491.192	0.749		
Equalized Odds					
C(method)	3.0	6.486	2.162	2.173	0.09
Residual	656.0	652.514	0.995		
Zero One Loss					
C(method)	3.0	0.008	0.003	1.523	0.207
Residual	656.0	1.159	0.002		
Disparate Impact					
C(method)	3.0	69.386	23.129	25.733	0.0
Residual	656.0	589.614	0.899		
Accuracy					
C(method)	3.0	0.404	0.135	0.134	0.94
Residual	656.0	658.596	1.004		
H-Mean					
C(method)	1.0	0.101	0.101	28.775	0.0
Residual	618.0	2.164	0.004		

(C) Neural Networks

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	1.0	42.363	42.363	45.402	0.0
Residual	618.0	576.637	0.933		
Equalized Odds					
C(method)	1.0	0.387	0.387	0.386	0.534
Residual	618.0	618.613	1.001		
Zero One Loss					
C(method)	1.0	349.329	349.329	0.826	0.364
Residual	373.0	157719.633	422.841		
Disparate Impact					
C(method)	1.0	7.721	7.721	7.806	0.005
Residual	618.0	611.279	0.989		
Accuracy					
C(method)	1.0	0.02	0.020	0.02	0.886
Residual	618.0	618.98	1.002		
H-Mean					
C(method)	1.0	0.101	0.101	28.775	0.0
Residual	618.0	2.164	0.004		

TABLE A.18: ANOVA tables of multi-class experiments with other classifiers

(A) Gradient Boosting

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	3.0	141.392	47.131	59.732	0.0
Residual	656.0	517.608	0.789		
Equalized Odds					
C(method)	3.0	31.746	10.582	11.067	0.0
Residual	656.0	627.254	0.956		
Zero-one Loss					
C(method)	3.0	164.678	54.893	0.429	0.732
Residual	203.0	25949.535	127.830		
Disparate Impact					
C(method)	3.0	67.483	22.494	24.947	0.0
Residual	656.0	591.517	0.902		
Accuracy					
C(method)	3.0	62.187	20.729	22.785	0.0
Residual	656.0	596.813	0.910		
H-Mean					
C(method)	3.0	0.385	0.128	46.927	0.0
Residual	656.0	1.793	0.003		

(B) Support Vector Machines

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	3.0	1.119	0.373	137.255	0.0
Residual	656.0	1.783	0.003		
Equalized Odds					
C(method)	3.0	0.839	0.280	17.93	0.0
Residual	656.0	10.227	0.016		
Zero One Loss					
C(method)	3.0	0.051	0.017	1.346	0.258
Residual	656.0	8.290	0.013		
Disparate Impact					
C(method)	3.0	6.316	2.105	115.033	0.0
Residual	656.0	12.005	0.018		
Accuracy					
C(method)	3.0	0.074	0.025	22.195	0.0
Residual	656.0	0.731	0.001		
H-Mean					
C(method)	3.0	2.383	0.794	211.341	0.0
Residual	656.0	2.465	0.004		

(C) Neural Networks

	DF	SS	MS	F	p-value
Statistical Parity					
C(method)	1.0	0.098	0.098	30.711	0.0
Residual	618.0	1.975	0.003		
Equalized Odds					
C(method)	1.0	0.007	0.007	0.892	0.345
Residual	618.0	5.064	0.008		
Zero One Loss					
C(method)	1.0	0.038	0.038	2.216	0.137
Residual	618.0	10.539	0.017		
Disparate Impact					
C(method)	1.0	0.556	0.556	23.01	0.0
Residual	618.0	14.938	0.024		
Accuracy					
C(method)	1.0	0.001	0.001	0.403	0.526
Residual	618.0	0.844	0.001		
H-Mean					
C(method)	1.0	0.101	0.101	28.775	0.0
Residual	618.0	2.164	0.004		

Bibliography

[1] J. Bosch, H. H. Olsson, and I. Crnkovic, *Engineering AI Systems: A Research Agenda*, en, ch., ISBN: 9781799851011 Pages: 1-19 Publisher: IGI Global, 2021. DOI: 10.4018/978-1-7998-5101-1.ch001.

[2] H. Muccini and K. Vaidhyanathan, "Software Architecture for ML-based Systems: What Exists and What Lies Ahead," in *2021 IEEE/ACM 1st Workshop on AI Engineering - Software Engineering for AI (WAIN)*, 2021, pp. 121–128. DOI: 10.1109/wain52551.2021.00026.

[3] S. Martínez-Fernández, J. Bogner, X. Franch, *et al.*, "Software Engineering for AI-Based Systems: A Survey," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 2, 37e:1–37e:59, 2022, ISSN: 1049-331X. DOI: 10.1145/3487043.

[4] S. Amershi, A. Begel, C. Bird, *et al.*, "Software Engineering for Machine Learning: A Case Study," en, in *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, Montreal, QC, Canada: IEEE, 2019, pp. 291–300, ISBN: 978-1-72811-760-7. DOI: 10.1109/ICSE-SEIP.2019.00042.

[5] N. Nahar, H. Zhang, G. Lewis, *et al.*, *A Meta-Summary of Challenges in Building Products with ML Components – Collecting Experiences from 4758+ Practitioners*, 2023. DOI: 10.48550/arXiv.2304.00078. (visited on 07/31/2023).

[6] F. Kumeno, "Sofware engneering challenges for machine learning applications: A literature review," en, *Intelligent Decision Technologies*, vol. 13, no. 4, pp. 463–476, 2020, ISSN: 18724981, 18758843. DOI: 10.3233/idt-190160.

[7] J. M. Zhang, M. Harman, L. Ma, *et al.*, "Machine learning testing: Survey, landscapes and horizons," en, *IEEE Transactions on Software Engineering*, pp. 1–1, 2020, ISSN: 0098-5589, 1939-3520, 2326-3881. DOI: 10.1109/tse.2019.2962027.

[8] S. Studer, T. B. Bui, C. Drescher, *et al.*, "Towards crisp-ml (q): A machine learning process model with quality assurance methodology," *Machine Learning and Knowledge Extraction*, vol. 3, no. 2, pp. 392–413, 2021.

[9] K. Hamada, F. Ishikawa, S. Masuda, *et al.*, "Guidelines for quality assurance of machine learning-based artificial intelligence.," in *SEKE*, 2020, pp. 335–341.

[10] S. Azimi and C. Pahl, "A layered quality framework for machine learning-driven data and information models.," in *ICEIS (1)*, 2020, pp. 579–587.

[11] F. Ishikawa, "Concepts in quality assessment for machine learning-from test data to arguments," in *International Conference on Conceptual Modeling*, Springer, 2018, pp. 536–544.

[12] J. Siebert, L. Joeckel, J. Heidrich, *et al.*, "Construction of a quality model for machine learning systems," *Software Quality Journal*, pp. 1–29, 2021.

[13] Y. Chang, X. Wang, J. Wang, *et al.*, "A survey on evaluation of large language models," *ACM Transactions on Intelligent Systems and Technology*, vol. 15, no. 3, pp. 1–45, 2024.

[14] Y. Brun and A. Meliou, "Software fairness," en, in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, Lake Buena Vista FL USA: Acm, 2018, pp. 754–759, ISBN: 978-1-4503-5573-5. DOI: 10.1145/3236024.3264838.

[15] Z. Chen, J. M. Zhang, M. Hort, *et al.*, *Fairness Testing: A Comprehensive Survey and Analysis of Trends*, 2022. arXiv: 2207.10223 [cs.SE]. [Online]. Available: http://arxiv.org/abs/2207.10223 (visited on 09/28/2022).

[16] S. Georgiou, M. Kechagia, T. Sharma, *et al.*, "Green ai: Do deep learning frameworks have different costs?" In *Procs. of the 44th International Conference on Software Engineering*, ser. ICSE '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 1082–1094, ISBN: 9781450392211. DOI: 10.1145/3510003.3510221.

[17] R. Verdecchia, P. Lago, C. Ebert, *et al.*, "Green it and green software," *IEEE Software*, vol. 38, no. 6, pp. 7–15, 2021.

[18] ONU, *ONU Sustainable Development Goals*, en-US. [Online]. Available: https://www.un.org/sustainabledevelopment/.

[19] N. Mehrabi, F. Morstatter, N. Saxena, *et al.*, "A survey on bias and fairness in machine learning," en, *ACM Comput. Surv.*, vol. 54, no. 6, pp. 1–35, 2021, ISSN: 0360-0300. DOI: 10.1145/3457607.

[20] S. Hajian, F. Bonchi, and C. Castillo, "Algorithmic bias: From discrimination discovery to fairness-aware data mining," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, ACM, 2016, pp. 2125–2126.

[21] A. D'Angelo, G. d'Aloisio, F. Marzi, *et al.*, "Uncovering gender gap in academia: A comprehensive analysis within the software engineering community," *Journal of Systems and Software*, p. 112 162, 2024.

[22] G. d'Aloisio, A. D'Angelo, F. Marzi, *et al.*, "Data-driven analysis of gender fairness in the software engineering academic landscape," in *European Conference on Software Architecture*, Springer Nature Switzerland Cham, 2023, pp. 89–103.

[23] R. Fischer, M. Jakobs, S. Mücke, *et al.*, "A unified framework for assessing energy efficiency of machine learning," in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2022, pp. 39–54.

[24] S. Wenninger, C. Kaymakci, C. Wiethe, *et al.*, "How sustainable is machine learning in energy applications?–the sustainable machine learning balance sheet," 2022.

[25] E. García-Martín, C. F. Rodrigues, G. Riley, *et al.*, "Estimation of energy consumption in machine learning," *Journal of Parallel and Distributed Computing*, vol. 134, pp. 75–88, 2019.

[26] I. Sommerville, *Software Engineering, 9/E*. Pearson Education India, 2011.

[27] R. K. E. Bellamy, K. Dey, M. Hind, *et al.*, "AI Fairness 360: An extensible toolkit for detecting and mitigating algorithmic bias," *IBM Journal of Research and Development*, vol. 63, no. 4/5, 4:1–4:15, 2019, Conference Name: IBM Journal of Research and Development, ISSN: 0018-8646. DOI: 10.1147/jrd.2019.2942287.

[28] S. Bird, M. Dudík, R. Edgar, *et al.*, "Fairlearn: A toolkit for assessing and improving fairness in AI," Microsoft, Tech. Rep. Msr-tr-2020-32, 2020. [Online]. Available: https://www.microsoft.com/en-us/research/publication/fairlearn-a-toolkit-for-assessing-and-improving-fairness-in-ai/.

[29] J. Zhang, P. Cao, D. P. Gross, *et al.*, "On the application of multi-class classification in physical therapy recommendation," en, *Health Information Science and Systems*, vol. 1, no. 1, p. 15, 2013, ISSN: 2047-2501. DOI: 10.1186/2047-2501-1-15.

[30] A. Baskota and Y.-K. Ng, "A graduate school recommendation system using the multi-class support vector machine and knn approaches," in *2018 IEEE International Conference on Information Reuse and Integration (IRI)*, IEEE, 2018, pp. 277–284.

[31] S. Caton and C. Haas, "Fairness in Machine Learning: A Survey," *ACM Computing Surveys*, 2023, ISSN: 0360-0300. DOI: 10.1145/3616865. eprint: 2010.04053.

[32] T. Nguyen, M. T. Baldassarre, L. F. de Lima, *et al.*, "From literature to practice: Exploring fairness testing tools for the software industry adoption," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, 2024, pp. 549–555.

[33] L. Sundberg and J. Holmström, "Democratizing artificial intelligence: How no-code AI can leverage machine learning operations," en, *Business Horizons*, 2023, ISSN: 0007-6813. DOI: 10.1016/j.bushor.2023.04.003.

[34] Z. Chen, J. M. Zhang, F. Sarro, *et al.*, "MAAT: A novel ensemble approach to addressing fairness and performance bugs for machine learning software," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. Esec/fse 2022, New York, NY, USA: Association for Computing Machinery, 2022, pp. 1122–1134, ISBN: 978-1-4503-9413-0. DOI: 10.1145/3540250.3549093.

[35] M. Hort, J. M. Zhang, F. Sarro, *et al.*, "Fairea: A model behaviour mutation approach to benchmarking bias mitigation methods," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. Esec/fse 2021, Athens, Greece: Association for Computing Machinery, 2021, 994–1006, ISBN: 9781450385626. DOI: 10.1145/3468264.3468565.

[36] M. H. Asyrofi, Z. Yang, I. N. B. Yusuf, *et al.*, "Biasfinder: Metamorphic test generation to uncover bias for sentiment analysis systems," *IEEE Transactions on Software Engineering*, vol. 48, no. 12, pp. 5087–5101, 2022. DOI: 10.1109/tse.2021.3136169.

[37] A. Shome, L. Cruz, and A. Van Deursen, "Data vs. model machine learning fairness testing: An empirical study," in *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, 2024, pp. 366–367.

[38] A. Fan, B. Gokkaya, M. Harman, *et al.*, *Large language models for software engineering: Survey and open problems*, 2023. arXiv: 2310.03533.

[39] R. Schwartz, J. Dodge, N. A. Smith, *et al.*, *Green ai*, 2019. arXiv: 1907.10597 [cs.CY]. [Online]. Available: https://arxiv.org/abs/1907.10597.

[40] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," in *NIPS Deep Learning and Representation Learning Workshop*, 2015. [Online]. Available: http://arxiv.org/abs/1503.02531.

[41] O. Zafrir, G. Boudoukh, P. Izsak, *et al.*, "Q8bert: Quantized 8bit bert," in *2019 Fifth Workshop on Energy Efficient Machine Learning and Cognitive Computing - NeurIPS Edition (EMC2-NIPS)*, Los Alamitos, CA, USA: IEEE Computer Society, 2019, pp. 36–39. DOI: 10.1109/EMC2-NIPS53020.2019.00016.

[42] V. Sanh, T. Wolf, and A. M. Rush, "Movement pruning: Adaptive sparsity by fine-tuning," in *Proceedings of the 34th International Conference on Neural Information Processing Systems*, ser. NIPS '20, Vancouver, BC, Canada: Curran Associates Inc., 2020, ISBN: 9781713829546.

[43] G. d'Aloisio, A. D'Angelo, A. Di Marco, *et al.*, "Debiaser for Multiple Variables to enhance fairness in classification tasks," en, *Information Processing & Management*, vol. 60, no. 2, p. 103 226, 2023, ISSN: 0306-4573. DOI: 10.1016/j.ipm.2022.103226.

[44] G. d'Aloisio, G. Stilo, A. Di Marco, *et al.*, "Enhancing Fairness in Classification Tasks with Multiple Variables: A Data-and Model-Agnostic Approach," in *International Workshop on Algorithmic Bias in Search and Recommendation*, Springer, 2022, pp. 117–129.

[45] V. Grossi, B. Rapisarda, F. Giannotti, *et al.*, "Data science at SoBigData: The European research infrastructure for social mining and big data analytics," en, *International Journal of Data Science and Analytics*, vol. 6, no. 3, pp. 205–216, 2018, ISSN: 2364-4168. DOI: 10.1007/s41060-018-0126-x.

[46] G. d'Aloisio, A. Di Marco, and G. Stilo, "Democratizing quality-based machine learning development through extended feature models," in *Fundamental Approaches to Software Engineering*, L. Lambers and S. Uchitel, Eds., Springer Nature Switzerland Cham, Cham: Springer Nature Switzerland, 2023, pp. 88–110, ISBN: 978-3-031-30826-0.

[47] G. d'Aloisio, C. Di Sipio, A. Di Marco, *et al.*, "How fair are we? from conceptualization to automated assessment of fairness definitions," 2024. DOI: arXiv:2404.09919.

[48] S. Apel, D. Batory, C. Kästner, *et al.*, *Feature-oriented software product lines*. Springer, 2016.

[49] G. d'Aloisio, C. Di Sipio, A. Di Marco, *et al.*, *MODNESS*, version 1.0.0, 2024. [Online]. Available: https://github.com/giordanoDaloisio/MODNESS.

[50] G. d'Aloisio, C. D. Sipio, A. D. Marco, *et al.*, *Towards Early Detection of Algorithmic Bias from Dataset's Bias Symptoms: An Empirical Study*, version 1.0.1, 2024. DOI: 10.5281/zenodo.14040687.

[51] F. Marzi, G. d'Aloisio, A. Di Marco, *et al.*, "Towards a prediction of machine learning training time to support continuous learning systems development," in *European Conference on Software Architecture*, Springer Nature Switzerland Cham, 2023, pp. 169–184.

[52] X. Zheng, J. Jia, S. Guo, *et al.*, "Full parameter time complexity (fptc): A method to evaluate the running time of machine learning classifiers for land use/land cover classification," *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, vol. 14, pp. 2222–2235, 2021.

[53] S. Menard, *Applied logistic regression analysis*. Sage, 2002, vol. 106.

[54] S. J. Rigatti, "Random forest," *Journal of Insurance Medicine*, vol. 47, no. 1, pp. 31–39, 2017.

[55] F. Marzi, G. d'Aloisio, A. Di Marco, *et al.*, *Towards a prediction of machine learning training time to support continuous learning systems development. replication package.* [Online]. Available: `https://github.com/giordanoDaloisio/QUALIFIER2023-FPTC\%5Fevaluation`.

[56] R. Rombach, A. Blattmann, D. Lorenz, *et al.*, "High-resolution image synthesis with latent diffusion models," in *CVPR*, 2022, pp. 10 674–10 685.

[57] D. Podell, Z. English, K. Lacey, *et al.*, "Sdxl: Improving latent diffusion models for high-resolution image synthesis," *arXiv preprint arXiv:2307.01952*, 2023.

[58] P. Esser, S. Kulal, A. Blattmann, *et al.*, "Scaling rectified flow transformers for high-resolution image synthesis," in *Forty-first International Conference on Machine Learning*, 2024.

[59] A. Valyaeva, *AI Image Statistics for 2024: How Much Content Was Created by AI*, en-US, 2023. [Online]. Available: `https://journal.everypixel.com/ai-image-statistics` (visited on 11/14/2024).

[60] M. Sami, A. Sami, and P. Barclay, "A case study of fairness in generated images of Large Language Models for Software Engineering tasks," in *2023 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, ISSN: 2576-3148, 2023, pp. 391–396. DOI: `10.1109/ICSME58846.2023.00051`.

[61] C. Treude and H. Hata, "She Elicits Requirements and He Tests: Software Engineering Gender Bias in Large Language Models," in *2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR)*, ISSN: 2574-3864, 2023, pp. 624–629. DOI: `10.1109/MSR59073.2023.00088`.

[62] T. Fadahunsi, G. d'Aloisio, A. Di Marco, and F. Sarro, *SD Bias Analysis Replication package*, 2024. [Online]. Available: `https://anonymous.4open.science/r/sd-bias/README.md`.

[63] A. Ait, J. L. C. Izquierdo, and J. Cabot, "HFCommunity: A Tool to Analyze the Hugging Face Hub Community," in *2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, ISSN: 2640-7574, 2023, pp. 728–732. DOI: `10.1109/SANER56733.2023.00080`.

[64] B. Johnson and Y. Brun, "Fairkit-learn: A fairness evaluation and comparison toolkit," in *2022 IEEE/ACM 44th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, ser. ICSE '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, pp. 70–74, ISBN: 978-1-4503-9223-5. DOI: `10.1145/3510454.3516830`.

[65] C. Di Sipio, G. d'Aloisio, A. Di Marco, and D. Di Ruscio, *HF Fairness Study*, version 1.0.1, 2024. [Online]. Available: `https://anonymous.4open.science/r/HF-Fairness-Study-F16E/README.md`.

[66] Z. Feng, D. Guo, D. Tang, *et al.*, "CodeBERT: A pre-trained model for programming and natural languages," in *Findings of the Association for Computational Linguistics: EMNLP 2020*, T. Cohn, Y. He, and Y. Liu, Eds., Online: Association for Computational Linguistics, 2020, pp. 1536–1547. DOI: 10.18653/v1/2020.findings-emnlp.139.

[67] A. Gholami, S. Kim, Z. Dong, *et al.*, "A survey of quantization methods for efficient neural network inference," in *Low-Power Computer Vision*, Chapman and Hall/CRC, 2022, pp. 291–326.

[68] M. A. Gordon, K. Duh, and N. Andrews, *Compressing BERT: Studying the Effects of Weight Pruning on Transfer Learning*, en, 2020. [Online]. Available: http://arxiv.org/abs/2002.08307 (visited on 09/22/2024).

[69] G. d'Aloisio, L. Traini, F. Sarro, and A. Di Marco, *On the compression of language models for code: An empirical study on codebert*, version 0.1, 2024. DOI: 10.5281/zenodo.14357478.

[70] J. Gong, S. Li, G. d'Aloisio, *et al.*, "Greenstableyolo: Optimizing inference time and image quality of text-to-image generation," in *International Symposium on Search Based Software Engineering*, Springer Nature Switzerland Cham, 2024, pp. 70–76.

[71] K. Deb, S. Agrawal, A. Pratap, *et al.*, "A fast and elitist multiobjective genetic algorithm: NSGA-II," *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, 2002.

[72] J. Gong, S. Li, G. d'Aloisio, *et al.*, *Greenstableyolo replication package*, 2024. [Online]. Available: https://github.com/gjz78910/GreenStableYolo.

[73] M. Assante, L. Candela, D. Castelli, *et al.*, "Enacting open science by d4science," *Future Generation Computer Systems*, vol. 101, pp. 555–563, 2019.

[74] K. A. Austin, C. M. Christopher, and D. Dickerson, "Will I Pass the Bar Exam: Predicting Student Success Using LSAT Scores and Law School Performance," *HofstrA l. rev.*, vol. 45, p. 753, 2016, Publisher: HeinOnline.

[75] P. T. Nguyen, R. Rubei, J. D. Rocco, *et al.*, *Dealing with popularity bias in recommender systems for third-party libraries: How far are we?* 2023. arXiv: 2304.10409 [cs.SE].

[76] S. Udeshi, P. Arora, and S. Chattopadhyay, "Automated directed fairness testing," in *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, ser. Ase '18, Montpellier, France: Association for Computing Machinery, 2018, 98–108, ISBN: 9781450359375. DOI: 10.1145/3238147.3238165.

[77] M. Hort, Z. Chen, J. M. Zhang, *et al.*, "Bias Mitigation for Machine Learning Classifiers: A Comprehensive Survey," en, *ACM Journal on Responsible Computing*, p. 3 631 326, 2023, ISSN: 2832-0565. DOI: 10.1145/3631326.

[78] A. Castelnovo, R. Crupi, G. Greco, *et al.*, "A clarification of the nuances in the fairness metrics landscape," *Scientific Reports*, vol. 12, no. 1, p. 4209, 2022.

[79] S. Majumder, J. Chakraborty, G. R. Bai, *et al.*, "Fair enough: Searching for sufficient measures of fairness," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 6, pp. 1–22, 2023.

[80] Z. Chen, J. M. Zhang, F. Sarro, *et al.*, "Fairness improvement with multiple protected attributes: How far are we?" eng, in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639083.

[81] C. Dwork, M. Hardt, T. Pitassi, *et al.*, "Fairness through awareness," in *Proceedings of the 3rd Innovations in Theoretical Computer Science Conference*, ser. Itcs '12, New York, NY, USA: Association for Computing Machinery, 2012, pp. 214–226, ISBN: 978-1-4503-1115-1. DOI: 10.1145/2090236.2090255. (visited on 12/16/2021).

[82] M. Feldman, S. A. Friedler, J. Moeller, *et al.*, "Certifying and Removing Disparate Impact," en, in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Sydney NSW Australia: Acm, 2015, pp. 259–268, ISBN: 978-1-4503-3664-2. DOI: 10.1145/2783258.2783311.

[83] M. Hardt, E. Price, E. Price, *et al.*, "Equality of Opportunity in Supervised Learning," in *Advances in Neural Information Processing Systems*, vol. 29, Curran Associates, Inc., 2016. [Online]. Available: https://proceedings.neurips.cc/paper/2016/hash/9d2682367c3935defcb1f9e247a97c0d-Abstract.html (visited on 01/27/2022).

[84] P. T. Nguyen, J. Di Rocco, D. Di Ruscio, *et al.*, "CrossRec: Supporting Software Developers by Recommending Third-party Libraries," *Journal of Systems and Software*, p. 110 460, 2019, ISSN: 0164-1212. DOI: https://doi.org/10.1016/j.jss.2019.110460.

[85] B. d'Alessandro, C. O'Neil, and T. LaGatta, "Conscientious classification: A data scientist's guide to discrimination-aware classification," *Big data*, vol. 5, no. 2, pp. 120–134, 2017.

[86] F. Kamiran and T. Calders, "Data preprocessing techniques for classification without discrimination," en, *Knowledge and Information Systems*, vol. 33, no. 1, pp. 1–33, 2012, ISSN: 0219-1377, 0219-3116. DOI: 10.1007/s10115-011-0463-8.

[87] C. Denis, R. Elie, M. Hebiri, *et al.*, "Fairness guarantee in multi-class classification," *arXiv:2109.13642 [math, stat]*, 2021. (visited on 11/17/2021).

[88] A. Agarwal, A. Beygelzimer, M. Dudik, *et al.*, "A reductions approach to fair classification," in *Proceedings of the 35th International Conference on Machine Learning*, J. Dy and A. Krause, Eds., ser. Proceedings of Machine Learning Research, vol. 80, PMLR, 2018, pp. 60–69. [Online]. Available: https://proceedings.mlr.press/v80/agarwal18a.html.

[89] P. Putzel and S. Lee, "Blackbox Post-Processing for Multiclass Fairness," 2022. [Online]. Available: http://arxiv.org/abs/2201.04461 (visited on 01/27/2022).

[90] D. H. Wolpert, *What does dinner cost?* en, 1999. [Online]. Available: http://www.no-free-lunch.org/coev.pdf.

[91] J. Kivinen and M. K. Warmuth, "Exponentiated gradient versus gradient descent for linear predictors," *information and computation*, vol. 132, no. 1, pp. 1–63, 1997.

[92] D. Pedreschi, S. Ruggieri, and F. Turini, "Measuring discrimination in socially-sensitive decision records," in *Proceedings of the 2009 SIAM international conference on data mining*, SIAM, 2009, pp. 581–592.

[93] F. Kamiran, A. Karim, and X. Zhang, "Decision theory for discrimination-aware classification," in *2012 IEEE 12th international conference on data mining*, IEEE, 2012, pp. 924–929.

[94] H. Zhang, M. A. Babar, and P. Tell, "Identifying relevant studies in software engineering," *Inf. Softw. Technol.*, vol. 53, no. 6, 625–637, 2011, ISSN: 0950-5849. DOI: 10.1016/j.infsof.2010.12.010.

[95] R. Angell, B. Johnson, Y. Brun, *et al.*, "Themis: Automatically testing software for discrimination," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. Esec/fse 2018, New York, NY, USA: Association for Computing Machinery, 2018, pp. 871–875, ISBN: 978-1-4503-5573-5. DOI: 10.1145/3236024.3264590. (visited on 04/04/2023).

[96] A. Sharma and H. Wehrheim, "Testing Machine Learning Algorithms for Balanced Data Usage," in *2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST)*, Issn: 2159-4848, 2019, pp. 125–135. DOI: 10.1109/icst.2019.00022.

[97] P. Zhang, J. Wang, J. Sun, *et al.*, "Automatic Fairness Testing of Neural Classifiers Through Adversarial Sampling," *IEEE Transactions on Software Engineering*, vol. 48, no. 9, pp. 3593–3612, 2022, Conference Name: IEEE Transactions on Software Engineering, ISSN: 1939-3520. DOI: 10.1109/tse.2021.3101478.

[98] J. Chakraborty, S. Majumder, Z. Yu, *et al.*, "Fairway: A way to build fair ML software," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. Esec/fse 2020, New York, NY, USA: Association for Computing Machinery, 2020, pp. 654–665, ISBN: 978-1-4503-7043-1. DOI: 10.1145/3368089.3409697.

[99] Y. Tian, Z. Zhong, V. Ordonez, *et al.*, "Testing dnn image classifiers for confusion & bias errors," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. Icse '20, Seoul, South Korea: Association for Computing Machinery, 2020, 1122–1134, ISBN: 9781450371216. DOI: 10.1145/3377811.3380400.

[100] A. Aggarwal, S. Shaikh, S. Hans, *et al.*, "Testing Framework for Black-box AI Models," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, Issn: 2574-1926, 2021, pp. 81–84. DOI: 10.1109/ICSE-Companion52605.2021.00041.

[101] L. Zhang, Y. Zhang, and M. Zhang, "Efficient white-box fairness testing through gradient search," in *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2021, pp. 103–114.

[102] J. Chakraborty, S. Majumder, and T. Menzies, "Bias in machine learning software: Why? how? what to do?" In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. Esec/fse 2021, Athens, Greece: Association for Computing Machinery, 2021, 429–440, ISBN: 9781450385626. DOI: 10.1145/3468264.3468537.

[103] S. Biswas and H. Rajan, "Fair preprocessing: Towards understanding compositional fairness of data transformers in machine learning pipeline," in *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. Esec/fse 2021, New York, NY, USA: Association for Computing Machinery, 2021, pp. 981–993, ISBN: 978-1-4503-8562-6. DOI: 10.1145/3468264.3468536.

[104] S. Tizpaz-Niari, A. Kumar, G. Tan, *et al.*, "Fairness-aware configuration of machine learning libraries," in *Proceedings of the 44th International Conference on Software Engineering*, ser. Icse '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 909–920, ISBN: 9781450392211. DOI: 10.1145/3510003.3510202.

[105] K. Peng, J. Chakraborty, and T. Menzies, "FairMask: Better Fairness via Model-Based Rebalancing of Protected Attributes," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 2426–2439, 2023, Conference Name: IEEE Transactions on Software Engineering, ISSN: 1939-3520. DOI: 10.1109/tse.2022.3220713.

[106] M. Fan, W. Wei, W. Jin, *et al.*, "Explanation-guided fairness testing through genetic algorithm," in *Proceedings of the 44th International Conference on Software Engineering*, ser. Icse '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 871–882, ISBN: 9781450392211. DOI: 10.1145/3510003.3510137.

[107] E. Soremekun, S. Udeshi, and S. Chattopadhyay, "Astraea: Grammar-Based Fairness Testing," *IEEE Transactions on Software Engineering*, vol. 48, no. 12, pp. 5188–5211, 2022, Conference Name: IEEE Transactions on Software Engineering, ISSN: 1939-3520. DOI: 10.1109/tse.2022.3141758.

[108] A. Perera, A. Aleti, C. Tantithamthavorn, *et al.*, "Search-based fairness testing for regression-based machine learning systems," en, *Empirical Software Engineering*, vol. 27, no. 3, p. 79, 2022, ISSN: 1573-7616. DOI: 10.1007/s10664-022-10116-7. (visited on 04/04/2023).

[109] H. Zheng, Z. Chen, T. Du, *et al.*, "Neuronfair: Interpretable white-box fairness testing through biased neuron identification," in *2022 IEEE/ACM 44th International Conference on Software Engineering (ICSE)*, 2022, pp. 1519–1531. DOI: 10.1145/3510003.3510123.

[110] Y. Li, L. Meng, L. Chen, *et al.*, "Training data debugging for the fairness of machine learning software," in *Proceedings of the 44th International Conference on Software Engineering*, ser. Icse '22, Pittsburgh, Pennsylvania: Association for Computing Machinery, 2022, 2215–2227, ISBN: 9781450392211. DOI: 10.1145/3510003.3510091.

[111] J. Wang, Y. Yang, S. Wang, *et al.*, "Context- and Fairness-Aware In-Process Crowdworker Recommendation," *ACM Transactions on Software Engineering and Methodology*, vol. 31, no. 3, 35:1–35:31, 2022, ISSN: 1049-331x. DOI: 10.1145/3487571. (visited on 04/04/2023).

[112] A. Yohannis and D. Kolovos, "Towards model-based bias mitigation in machine learning," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems*, ser. Models '22, Montreal, Quebec, Canada: Association for Computing Machinery, 2022, 143–153, ISBN: 9781450394666. DOI: 10.1145/3550355.3552401.

[113] S. S. Rajan, S. Udeshi, and S. Chattopadhyay, "AequeVox: Automated Fairness Testing of Speech Recognition Systems," en, in *Fundamental Approaches to Software Engineering*, E. B. Johnsen and M. Wimmer, Eds., Cham: Springer International Publishing, 2022, pp. 245–267, ISBN: 978-3-030-99429-7. DOI: 10.1007/978-3-030-99429-7_14.

[114] S. Biswas and H. Rajan, "Fairify: Fairness Verification of Neural Networks," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23, Melbourne, Victoria, Australia: IEEE Press, 2023, pp. 1546–1558, ISBN: 978-1-66545-701-9. DOI: 10.1109/ICSE48619.2023.00134.

[115] V. Monjezi, A. Trivedi, G. Tan, *et al.*, "Information-Theoretic Testing and Debugging of Fairness Defects in Deep Neural Networks," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23, Melbourne, Victoria, Australia: IEEE Press, 2023, pp. 1571–1582, ISBN: 978-1-66545-701-9. DOI: 10.1109/ICSE48619.2023.00136. (visited on 05/06/2024).

[116] R. Berk, H. Heidari, S. Jabbari, *et al.*, "Fairness in Criminal Justice Risk Assessments: The State of the Art," vol. 50, no. 1, pp. 3–44, 2018, Publisher: SAGE PublicationsSage CA: Los Angeles, CA, ISSN: 15528294. DOI: 10.1177/0049124118782533.

[117] L. Oneto, M. Doninini, A. Elders, *et al.*, "Taking Advantage of Multitask Learning for Fair Classification," in *Proceedings of the 2019 AAAI/ACM Conference on AI, Ethics, and Society*, ser. AIES '19, New York, NY, USA: Association for Computing Machinery, 2019, pp. 227–237, ISBN: 978-1-4503-6324-2. DOI: 10.1145/3306618.3314255.

[118] M. Openja, G. Laberge, and F. Khomh, "Detection and evaluation of bias-inducing features in machine learning," en, *Empirical Software Engineering*, vol. 29, no. 1, p. 22, 2023, ISSN: 1573-7616. DOI: 10.1007/s10664-023-10409-5.

[119] C. Ferrara, F. Casillo, and C. Gravino, "ReFAIR: Toward a Context-Aware Recommenderfor Fairness Requirements Engineering," en, 2024.

[120] B. Salimi, L. Rodriguez, B. Howe, *et al.*, "Interventional Fairness: Causal Database Repair for Algorithmic Fairness," en, in *Proceedings of the 2019 International Conference on Management of Data*, Amsterdam Netherlands: ACM, 2019, pp. 793–810, ISBN: 978-1-4503-5643-5. DOI: 10.1145/3299869.3319901.

[121] S. Galhotra, K. Shanmugam, P. Sattigeri, *et al.*, "Causal Feature Selection for Algorithmic Fairness," in *Proceedings of the 2022 International Conference on Management of Data*, ser. SIGMOD '22, New York, NY, USA: Association for Computing Machinery, 2022, pp. 276–285, ISBN: 978-1-4503-9249-5. DOI: 10.1145/3514221.3517909.

[122] M. Mecati, A. Vetrò, and M. Torchiano, "Detecting Discrimination Risk in Automated Decision-Making Systems with Balance Measures on Input Data," in *2021 IEEE International Conference on Big Data (Big Data)*, 2021, pp. 4287–4296. DOI: 10.1109/BigData52589.2021.9671443.

[123] W. Yik, L. Serafini, T. Lindsey, *et al.*, "Identifying Bias in Data Using Two-Distribution Hypothesis Tests," in *Proceedings of the 2022 AAAI/ACM Conference on AI, Ethics, and Society*, ser. AIES '22, New York, NY, USA: Association for Computing Machinery, 2022, pp. 831–844, ISBN: 978-1-4503-9247-1. DOI: 10.1145/3514094.3534169.

[124] R. Constantin, M. Dück, A. Alexandrov, *et al.*, "How Do Algorithmic Fairness Metrics Align with Human Judgement? A Mixed-Initiative System for Contextualized Fairness Assessment," in *2022 IEEE Workshop on TRust and EXpertise in Visual Analytics (TREX)*, 2022, pp. 1–7. DOI: 10.1109/TREX57753.2022.00005.

[125] J. M. Zhang and M. Harman, ""Ignorance and Prejudice" in Software Fairness," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, 2021, pp. 1436–1447. DOI: 10.1109/ICSE43902.2021.00129.

[126] C. Du and T. Chen, "Contexts matter: An empirical study on contextual influence in fairness testing for deep learning systems," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '24, Barcelona, Spain: Association for Computing Machinery, 2024, 107–118, ISBN: 9798400710476. DOI: 10.1145/3674805.3686673.

[127] F. Bianchi, P. Kalluri, E. Durmus, *et al.*, "Easily accessible text-to-image generation amplifies demographic stereotypes at large scale," in *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, 2023, pp. 1493–1504.

[128] R. Naik and B. Nushi, "Social biases through the text-to-image generation lens," in *Proceedings of the 2023 AAAI/ACM Conference on AI, Ethics, and Society*, 2023, pp. 786–808.

[129] L. Sun, M. Wei, Y. Sun, *et al.*, "Smiling women pitching down: Auditing representational and presentational gender biases in image-generative ai," *Journal of Computer-Mediated Communication*, vol. 29, no. 1, zmad045, 2024.

[130] S. Luccioni, C. Akiki, M. Mitchell, *et al.*, "Stable Bias: Evaluating Societal Representations in Diffusion Models," en, *Advances in Neural Information Processing Systems*, vol. 36, pp. 56 338–56 351, 2023. (visited on 06/20/2024).

[131] Y. Wan, A. Subramonian, A. Ovalle, *et al.*, "Survey of bias in text-to-image generation: Definition, evaluation, and mitigation," *arXiv preprint arXiv:2404.01030*, 2024.

[132] J. Castaño, S. Martínez-Fernández, X. Franch, *et al.*, *Analyzing the Evolution and Maintenance of ML Models on Hugging Face*, 2023. DOI: 10.48550/arXiv.2311.13380.

[133] L. Gong, J. Zhang, M. Wei, *et al.*, "What Is the Intended Usage Context of This Model? An Exploratory Study of Pre-Trained Models on Various Model Repositories," *ACM Transactions on Software Engineering and Methodology*, vol. 32, no. 3, 69:1–69:57, 2023, ISSN: 1049-331X. DOI: 10.1145/3569934.

[134] C. Di Sipio, R. Rubei, J. Di Rocco, *et al.*, "Automated categorization of pretrained models in software engineering: A case study with a hugging face dataset," in *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*, ser. EASE '24, Salerno, Italy: Association for Computing Machinery, 2024, 351–356, ISBN: 9798400717017. DOI: 10.1145/3661167.3661215.

[135] D. Montes, P. Peerapatanapokin, J. Schultz, *et al.*, "Discrepancies among pretrained deep neural networks: A new threat to model zoo reliability," in *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2022, New York, NY, USA: Association for Computing Machinery, 2022, pp. 1605–1609, ISBN: 978-1-4503-9413-0. DOI: 10.1145/3540250.3560881.

[136] F. Pepe, V. Nardone, A. Mastropaolo, *et al.*, "How do hugging face models document datasets, bias, and licenses? an empirical study," in *Proceedings of the 32nd IEEE/ACM International Conference on Program Comprehension*, ser. ICPC '24, Lisbon, Portugal: Association for Computing Machinery, 2024, 370–381, ISBN: 9798400705861. DOI: 10.1145/3643916.3644412.

[137] H. Gao, M. Zahedi, C. Treude, *et al.*, "Documenting ethical considerations in open source ai models," in *Proceedings of the 18th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, ser. ESEM '24, Barcelona, Spain: Association for Computing Machinery, 2024, 177–188, ISBN: 9798400710476. DOI: 10.1145/3674805.3686679.

[138] N. Yanes, A. M. Mostafa, M. Ezz, *et al.*, "A machine learning-based recommender system for improving students learning experiences," *IEEE Access*, vol. 8, pp. 201 218–201 235, 2020.

[139] C. Jiang, Y. Liu, Y. Ding, *et al.*, "Capturing helpful reviews from social media for product quality improvement: A multi-class classification approach," *International Journal of Production Research*, vol. 55, no. 12, pp. 3528–3541, 2017.

[140] N. V. Chawla, K. W. Bowyer, L. O. Hall, *et al.*, "Smote: Synthetic minority over-sampling technique," *Journal of artificial intelligence research*, vol. 16, pp. 321–357, 2002.

[141] H. He, Y. Bai, E. A. Garcia, *et al.*, "Adasyn: Adaptive synthetic sampling approach for imbalanced learning," in *2008 IEEE international joint conference on neural networks (IEEE world congress on computational intelligence)*, IEEE, 2008, pp. 1322–1328.

[142] J. H. Friedman, "Stochastic gradient boosting," *Computational statistics & data analysis*, vol. 38, no. 4, pp. 367–378, 2002, Publisher: Elsevier.

[143] W. S. Noble, "What is a support vector machine?" *Nature biotechnology*, vol. 24, no. 12, pp. 1565–1567, 2006, Publisher: Nature Publishing Group.

[144] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural network design*. PWS Publishing Co., 1997.

[145] F. Pedregosa, G. Varoquaux, A. Gramfort, *et al.*, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[146] S. Radovanović, A. Petrović, B. Delibašić, *et al.*, "A fair classifier chain for multi-label bank marketing strategy classification," en, *International Transactions in Operational Research*, 2021, ISSN: 1475-3995. DOI: 10.1111/itor.13059. (visited on 12/01/2021).

[147] P. Domingos and M. Pazzani, "On the Optimality of the Simple Bayesian Classifier under Zero-One Loss," en, *Machine Learning*, vol. 29, no. 2, pp. 103–130, 1997, ISSN: 1573-0565. DOI: 10.1023/A:1007413511361.

[148] G. Rosenfield and K. Fitzpatrick-Lins, "A coefficient of agreement as a measure of thematic classification accuracy.," *Photogrammetric Engineering and Remote Sensing*, vol. 52, no. 2, pp. 223–227, 1986. [Online]. Available: http://pubs.er.usgs.gov/publication/70014667.

[149] W. F. Ferger, "The nature and use of the harmonic mean," *Journal of the American Statistical Association*, vol. 26, no. 173, 36 – 40, 1931. DOI: 10.1080/01621459.1931.10503148.

[150] Fairlearn, *Fairlearn documentation*, 2022. [Online]. Available: `https://fairlearn.org/main/faq.html`.

[151] P. Refaeilzadeh, L. Tang, and H. Liu, "Cross-Validation," in *Encyclopedia of Database Systems*, L. LIU and M. T. ÖZSU, Eds., vol. 5, New York, NY: Springer New York, 2016, pp. 1–7, ISBN: 978-1-4899-7993-3. DOI: `10.1007/978-1-4899-7993-3\_565-2`.

[152] R. Kohavi *et al.*, "Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid.," en, in *Kdd*, vol. 96, 1996, pp. 202–207.

[153] J. Angwin, J. Larson, S. Mattu, *et al.*, "Machine bias," *ProPublica, May*, vol. 23, no. 2016, pp. 139–159, 2016.

[154] H. Hofmann, *Statlog (German Credit Data)*, UCI Machine Learning Repository, DOI: https://doi.org/10.24432/C5NC77, 1994.

[155] T.-S. Lim, W.-Y. Loh, and Y.-S. Shih, "A comparison of prediction accuracy, complexity, and training time of thirty-three old and new classification algorithms," *Machine learning*, vol. 40, no. 3, pp. 203–228, 2000, Publisher: Springer.

[156] M. Redmond and A. Baveja, "A data-driven software tool for enabling cooperative information sharing among police departments," *European Journal of Operational Research*, vol. 141, no. 3, pp. 660–678, 2002, Publisher: Elsevier.

[157] T. Calders, A. Karim, F. Kamiran, *et al.*, "Controlling Attribute Effect in Linear Regression," in *2013 IEEE 13th International Conference on Data Mining*, ISSN: 2374-8486, IEEE, 2013, pp. 71–80. DOI: `10.1109/ICDM.2013.114`.

[158] E. Fehrman, A. K. Muhammad, E. M. Mirkes, *et al.*, "The Five Factor Model of Personality and Evaluation of Drug Consumption Risk," en, in *Data Science*, F. Palumbo, A. Montanari, and M. Vichi, Eds., ser. Studies in Classification, Data Analysis, and Knowledge Organization, Cham: Springer International Publishing, 2017, pp. 231–242, ISBN: 978-3-319-55723-6. DOI: `10.1007/978-3-319-55723-6_18`.

[159] A. Tsanas, M. Little, P. McSharry, *et al.*, "Accurate telemonitoring of Parkinson's disease progression by non-invasive speech tests," en, *Nature Precedings*, pp. 1–1, 2009, Publisher: Nature Publishing Group, ISSN: 1756-0357. DOI: `10.1038/npre.2009.3920.1`.

[160] P. Cortez, A. Cerdeira, F. Almeida, *et al.*, "Modeling wine preferences by data mining from physicochemical properties," *Decision support systems*, vol. 47, no. 4, pp. 547–553, 2009, Publisher: Elsevier.

[161] J. H. McDonald, *One-way ANOVA*. sparky house publishing Baltimore, MD, 2009, vol. 2.

[162] D. Krstinić, M. Braović, L. Šerić, *et al.*, "Multi-label classifier performance evaluation with confusion matrix," *Comput Sci Inf Technol*, vol. 10, pp. 1–14, 2020.

[163] J. Pineau, P. Vincent-Lamarre, K. Sinha, *et al.*, "Improving reproducibility in machine learning research: A report from the neurips 2019 reproducibility program," *Journal of Machine Learning Research*, vol. 22, 2021.

[164] M. Brambilla, J. Cabot, and M. Wimmer, *Model-driven software engineering in practice*. Morgan & Claypool Publishers, 2017.

[165] D. Steinberg, F. Budinsky, M. Paternostro, *et al.*, *EMF: Eclipse Modeling Framework, 2nd Edition*, 2nd. Addison-Wesley Professional., 2008, ISBN: 978-0-321-33188-5. [Online]. Available: https://www.informit.com/store/emf-eclipse-modeling-framework-9780321331885 (visited on 05/10/2022).

[166] L. Bettini, *Implementing domain-specific languages with Xtext and Xtend*. Packt Publishing Ltd, 2016.

[167] J. Musset, É. Juliot, S. Lacrampe, *et al.*, "Acceleo user guide," *See also http://acceleo.org/doc/obeo/en/acceleo-2.6-user-guide. pdf*, vol. 2, p. 157, 2006.

[168] J. A. Galindo, D. Benavides, P. Trinidad, *et al.*, "Automated analysis of feature models: Quo vadis?" en, *Computing*, vol. 101, no. 5, pp. 387–433, 2019, ISSN: 0010-485X, 1436-5057. DOI: 10.1007/s00607-018-0646-1.

[169] K. C. Kang, S. G. Cohen, J. A. Hess, *et al.*, "Feature-oriented domain analysis (foda) feasibility study," Carnegie-Mellon Univ Pittsburgh Pa Software Engineering Inst, Tech. Rep., 1990.

[170] D. Benavides, S. Segura, and A. Ruiz-Cortés, "Automated analysis of feature models 20 years later: A literature review," en, *Information Systems*, vol. 35, no. 6, pp. 615–636, 2010, ISSN: 0306-4379. DOI: 10.1016/j.is.2010.01.001.

[171] C. Di Sipio, J. Di Rocco, D. Di Ruscio, *et al.*, "Lev4rec: A feature-based approach to engineering rsses," *Journal of Computer Languages*, vol. 78, p. 101 256, 2024, ISSN: 2590-1184. DOI: https://doi.org/10.1016/j.cola.2023.101256.

[172] T. Thüm, C. Kästner, F. Benduhn, *et al.*, "Featureide: An extensible framework for feature-oriented software development," *Science of Computer Programming*, vol. 79, pp. 70–85, 2014.

[173] S Patro and K. K. Sahu, "Normalization: A preprocessing stage," *arXiv preprint arXiv:1503.06462*, 2015.

[174] M. Kearns, S. Neel, A. Roth, *et al.*, "An Empirical Study of Rich Subgroup Fairness for Machine Learning," en, in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, Atlanta GA USA: ACM, 2019, pp. 100–109, ISBN: 978-1-4503-6125-5. DOI: 10.1145/3287560.3287592. (visited on 12/01/2021).

[175] L. E. Celis, L. Huang, V. Keswani, *et al.*, "Classification with fairness constraints: A meta-algorithm with provable guarantees," in *Proceedings of the conference on fairness, accountability, and transparency*, 2019, pp. 319–328.

[176] I. Giagkiozis and P. J. Fleming, "Pareto front estimation for decision making," *Evolutionary Computation*, vol. 22, no. 4, pp. 651–678, 2014, ISSN: 1063-6560. DOI: 10.1162/EVCO_a_00128. eprint: https://direct.mit.edu/evco/article-pdf/22/4/651/1530439/evco_a_00128.pdf.

[177] *React library*, 2025. [Online]. Available: https://react.dev/.

[178] *Flask library*, 2025. [Online]. Available: https://flask.palletsprojects.com/.

[179] *Celery library*, 2025. [Online]. Available: https://docs.celeryq.dev/en/stable.

[180] *Rabbitmq website*, 2025. [Online]. Available: https://www.rabbitmq.com/.

[181] *Redis website*, 2025. [Online]. Available: https://redis.io/.

[182] P.-E. Danielsson, "Euclidean distance mapping," *Computer Graphics and image processing*, vol. 14, no. 3, pp. 227–248, 1980.

[183] A. Singh, A. Yadav, and A. Rana, "K-means with three different distance metrics," *International Journal of Computer Applications*, vol. 67, no. 10, 2013.

[184] G. J. McLachlan, "Mahalanobis distance," *Resonance*, vol. 4, no. 6, pp. 20–26, 1999.

[185] E. Lepeschkin and B. Surawicz, "Characteristics of true-positive and false-positive results of electrocardiographs master two-step exercise tests," *New England Journal of Medicine*, vol. 258, no. 11, pp. 511–520, 1958.

[186] PalletsProject, *Jinja website*, 2023. [Online]. Available: `https://jinja.palletsprojects.com/`.

[187] *Conda website*. [Online]. Available: `https://docs.conda.io/`.

[188] J. Liu, E. Pacitti, P. Valduriez, *et al.*, "A survey of data-intensive scientific workflow management," *Journal of Grid Computing*, vol. 13, no. 4, pp. 457–493, 2015.

[189] M. R. Berthold, N. Cebron, F. Dill, *et al.*, "Knime - the konstanz information miner: Version 2.0 and beyond," *SIGKDD Explor. Newsl.*, vol. 11, no. 1, 26–31, 2009, ISSN: 1931-0145. DOI: `10.1145/1656274.1656280`.

[190] *Pickle documentation*, 2023. [Online]. Available: `https://docs.python.org/3/library/pickle.html`.

[191] P. Saleiro, B. Kuester, L. Hinkson, *et al.*, "Aequitas: A bias and fairness audit toolkit," en, *arXiv preprint arXiv:1811.05577*, 2018. [Online]. Available: `http://arxiv.org/abs/1811.05577` (visited on 11/11/2021).

[192] M. Robillard, R. Walker, and T. Zimmermann, "Recommendation systems for software engineering," *IEEE software*, vol. 27, no. 4, pp. 80–86, 2009.

[193] Wes McKinney, "Data Structures for Statistical Computing in Python," in *Proceedings of the 9th Python in Science Conference*, Stéfan van der Walt and Jarrod Millman, Eds., 2010, pp. 56–61. DOI: `10.25080/Majora-92bf1922-00a`.

[194] S. Verma and J. Rubin, "Fairness definitions explained," en, in *Proceedings of the International Workshop on Software Fairness*, IEEE, Gothenburg Sweden: Acm, 2018, pp. 1–7, ISBN: 978-1-4503-5746-3. DOI: `10.1145/3194770.3194776`. (visited on 03/03/2022).

[195] S. Moro, P. Cortez, and P. Rita, "A data-driven approach to predict the success of bank telemarketing," *Decision Support Systems*, vol. 62, pp. 22–31, 2014.

[196] J. Di Rocco and C. Di Sipio, "Resyduo: Combining data models and cf-based recommender systems to develop arduino projects," in *2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, Ieee, 2023, pp. 539–548. arXiv: `2308.13808 [cs.SE]`.

[197] Y. Wang, W. Ma, M. Zhang, *et al.*, "A Survey on the Fairness of Recommender Systems," *ACM Transactions on Information Systems*, vol. 41, no. 3, 52:1–52:43, 2023, ISSN: 1046-8188. DOI: `10.1145/3547333`.

[198] Y. Deldjoo, V. W. Anelli, H. Zamani, *et al.*, "Recommender systems fairness evaluation via generalized cross entropy," *arXiv preprint arXiv:1908.06708*, 2019.

[199] M. F. Bertoa and A. Vallecillo, "Quality attributes for software metamodels," 2010. [Online]. Available: `https://api.semanticscholar.org/CorpusID:15268921`.

[200] C. Croux and C. Dehon, "Influence functions of the spearman and kendall correlation measures," *Statistical methods & applications*, vol. 19, pp. 497–515, 2010.

[201] A. Altmann, L. Toloşi, O. Sander, *et al.*, "Permutation importance: A corrected feature importance measure," *Bioinformatics*, vol. 26, no. 10, pp. 1340–1347, 2010.

[202] C. Yang, Y. Akimoto, D. W. Kim, *et al.*, "OBOE: Collaborative Filtering for AutoML Model Selection," in *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, ser. KDD '19, New York, NY, USA: Association for Computing Machinery, 2019, pp. 1173–1183, ISBN: 978-1-4503-6201-6. DOI: 10.1145/3292500.3330909.

[203] M. Cerrada, L. Trujillo, D. E. Hernández, *et al.*, "Automl for feature selection and model tuning applied to fault severity diagnosis in spur gearboxes," *Mathematical and Computational Applications*, vol. 27, no. 1, p. 6, 2022.

[204] M. Feurer, A. Klein, K. Eggensperger, *et al.*, "Efficient and Robust Automated Machine Learning," in *Advances in Neural Information Processing Systems*, C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, Eds., vol. 28, Curran Associates, Inc., 2015. [Online]. Available: https://proceedings.neurips.cc/paper\%5Ffiles/paper/2015/file/11d0e6287202fced83f79975ec59a3a6-Paper.pdf.

[205] G. D. Ruxton, "The unequal variance t-test is an underused alternative to student's t-test and the mann–whitney u test," *Behavioral Ecology*, vol. 17, no. 4, pp. 688–690, 2006.

[206] T. E. Duncan, "On the calculation of mutual information," *SIAM Journal on Applied Mathematics*, vol. 19, no. 1, pp. 215–220, 1970.

[207] B. Becker and R. Kohavi, *Adult*, UCI Machine Learning Repository, 1996. DOI: 10.24432/C5XW20.

[208] H. Guvenir, B. Acar, H. Muderrisoglu, *et al.*, *Arrhythmia*, Published: UCI Machine Learning Repository, 1998.

[209] *Campus recruitment*. [Online]. Available: https://www.kaggle.com/datasets/benroshan/factors-affecting-campus-placement?resource=download.

[210] V. N. Dornadula and S. Geetha, "Credit card fraud detection using machine learning algorithms," *Procedia computer science*, vol. 165, pp. 631–641, 2019.

[211] J. Clore, K. Cios, J. DeShazo, *et al.*, *Diabetes 130-US Hospitals for Years 1999-2008*, UCI Machine Learning Repository, 2014. DOI: 10.24432/C5230J.

[212] *Heritage health dataset*, 2012. [Online]. Available: https://www.kaggle.com/c/hhp/data.

[213] A. Janosi, W. Steinbrunn, M. Pfisterer, *et al.*, *Heart Disease*, UCI Machine Learning Repository, 1988. DOI: 10.24432/C52P4X.

[214] *Ibm hr analytics employee attrition & performance*. [Online]. Available: https://www.kaggle.com/datasets/pavansubhasht/ibm-hr-analytics-attrition-dataset.

[215] F. M. Palechor and A. d. l. H. Manotas, "Dataset for estimation of obesity levels based on eating habits and physical condition in individuals from Colombia, Peru and Mexico," en, *Data in Brief*, vol. 25, p. 104 344, 2019, ISSN: 2352-3409. DOI: 10.1016/j.dib.2019.104344.

[216] *Ricci dataset*. [Online]. Available: https://jse.amstat.org/jse\%5Fdata\%5Farchive.htm.

[217] M. Chan, L. Klein, J. Fan, *et al.*, "Scg-rhc: Wearable seismocardiogram signal and right heart catheter database,"

[218] P. Cortez, *Student Performance*, UCI Machine Learning Repository, 2014. DOI: 10.24432/C5TG7T.

[219] M. Meek, T. Thiesson, and H. Heckerman, *US Census Data (1990)*, UCI Machine Learning Repository. DOI: 10.24432/C5VP42.

[220] D. Slunge, "The Willingness to Pay for Vaccination against Tick-Borne Encephalitis and Implications for Public Health Policy: Evidence from Sweden," en, *PLOS ONE*, vol. 10, no. 12, U. Pal, Ed., e0143875, 2015, ISSN: 1932-6203. DOI: 10.1371/journal.pone.0143875.

[221] A. Fabris, S. Messina, G. Silvello, *et al.*, "Algorithmic fairness datasets: The story so far," en, *Data Mining and Knowledge Discovery*, vol. 36, no. 6, pp. 2074–2152, 2022, ISSN: 1573-756X. DOI: 10.1007/s10618-022-00854-z.

[222] T. Le Quy, A. Roy, V. Iosifidis, *et al.*, "A survey on datasets for fairness-aware machine learning," en, *WIREs Data Mining and Knowledge Discovery*, vol. 12, no. 3, e1452, 2022, ISSN: 1942-4795. DOI: 10.1002/widm.1452. (visited on 02/26/2024).

[223] M. Hort, J. M. Zhang, F. Sarro, *et al.*, "Search-based Automatic Repair for Fairness and Accuracy in Decision-making Software," en, *Empirical Software Engineering*, vol. 29, no. 1, p. 36, 2024, ISSN: 1573-7616. DOI: 10.1007/s10664-023-10419-3.

[224] A. Mastropaolo, M. Ciniselli, M. Di Penta, *et al.*, *Evaluating Code Summarization Techniques: A New Metric and an Empirical Characterization*, en, 2023. DOI: arXiv:2312.15475.

[225] F. E. Harrell, C. Dupont, *et al.*, "Hmisc: Harrell miscellaneous," *R package version*, vol. 4, no. 0, 2020.

[226] S. Biswas and H. Rajan, "Do the machine learning models on a crowd sourced platform exhibit bias? an empirical study on model fairness," in *Proceedings of the 28th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2020, pp. 642–653.

[227] T. Chen, T. He, M. Benesty, *et al.*, "Xgboost: Extreme gradient boosting," *R package version 0.4-2*, vol. 1, no. 4, pp. 1–4, 2015.

[228] M. Sokolova, N. Japkowicz, and S. Szpakowicz, "Beyond accuracy, f-score and roc: A family of discriminant measures for performance evaluation," in *Australasian joint conference on artificial intelligence*, Springer, 2006, pp. 1015–1021.

[229] M. Buckland and F. Gey, "The relationship between recall and precision," *Journal of the American society for information science*, vol. 45, no. 1, pp. 12–19, 1994, Publisher: Wiley Online Library.

[230] C. Goutte and E. Gaussier, "A probabilistic interpretation of precision, recall and f-score, with implication for evaluation," in *European conference on information retrieval*, Springer, 2005, pp. 345–359.

[231] P. A. Flach, "Roc analysis," in *Encyclopedia of machine learning and data mining*, Springer, 2016, pp. 1–8.

[232] W. H. Kruskal and W. A. Wallis, "Use of ranks in one-criterion variance analysis," *Journal of the American statistical Association*, vol. 47, no. 260, pp. 583–621, 1952.

[233] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[234] R. B. Bendel, S. S. Higgins, J. E. Teberg, *et al.*, "Comparison of skewness coefficient, coefficient of variation, and Gini coefficient as inequality measures within populations," en, *Oecologia*, vol. 78, no. 3, pp. 394–400, 1989, ISSN: 1432-1939. DOI: 10.1007/BF00379115.

[235] X. Zeng and D. S. Yeung, "A Quantified Sensitivity Measure for Multilayer Perceptron to Input Perturbation," *Neural Computation*, vol. 15, no. 1, pp. 183–212, 2003, ISSN: 0899-7667. DOI: 10.1162/089976603321043757.

[236] J.-B. Yang, K.-Q. Shen, C.-J. Ong, *et al.*, "Feature selection for mlp neural network: The use of random permutation of probabilistic outputs," *IEEE Transactions on Neural Networks*, vol. 20, no. 12, pp. 1911–1922, 2009.

[237] N. Japkowicz and S. Stephen, "The class imbalance problem: A systematic study," *Intelligent data analysis*, vol. 6, no. 5, pp. 429–449, 2002.

[238] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of machine learning research*, vol. 3, no. Mar, pp. 1157–1182, 2003.

[239] R. Moussa and F. Sarro, "On the use of evaluation measures for defect prediction studies," in *Proceedings of the 31st ACM SIGSOFT International Symposium on Software Testing and Analysis*, New York, NY, USA: Association for Computing Machinery, 2022, ISBN: 9781450393799. DOI: 10.1145/3533767.3534405.

[240] D. Kumar, O. Lesota, G. Zerveas, *et al.*, *Parameter-efficient modularised bias mitigation via adapterfusion*, 2023. arXiv: 2302.06321 [cs.CL]. [Online]. Available: https://arxiv.org/abs/2302.06321.

[241] C. Schuhmann, R. Beaumont, R. Vencu, *et al.*, "Laion-5b: An open large-scale dataset for training next generation image-text models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 25 278–25 294, 2022.

[242] J. Li, D. Li, C. Xiong, *et al.*, *Blip: Bootstrapping language-image pre-training for unified vision-language understanding and generation*, 2022. DOI: 10.48550/ARXIV.2201.12086.

[243] J. Kotrlik and C. Higgins, "Organizational research: Determining appropriate sample size in survey research appropriate sample size in survey research," *Information technology, learning, and performance journal*, vol. 19, no. 1, p. 43, 2001.

[244] A. A. Taha and A. Hanbury, "Metrics for evaluating 3D medical image segmentation: Analysis, selection, and tool," *BMC Medical Imaging*, vol. 15, no. 1, p. 29, 2015, ISSN: 1471-2342. DOI: 10.1186/s12880-015-0068-x.

[245] O. Keyes, "The misgendering machines: Trans/hci implications of automatic gender recognition," *Procs. of the ACM on human-computer interaction*, vol. 2, no. CSCW, pp. 1–22, 2018.

[246] H. Weerts, *An Introduction to Responsible Machine Learning*. 2024. [Online]. Available: https://hildeweerts.github.io/responsiblemachinelearning/.

[247] M. Mitchell, S. Wu, A. Zaldivar, *et al.*, "Model cards for model reporting," in *Proceedings of the Conference on Fairness, Accountability, and Transparency*, ser. FAT*'19, Atlanta, GA, USA: Association for Computing Machinery, 2019, 220–229, ISBN: 9781450361255. DOI: 10.1145/3287560.3287596.

[248] H. Borges, A. Hora, and M. T. Valente, "Understanding the Factors That Impact the Popularity of GitHub Repositories," en, in *2016 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, Raleigh, NC, USA: IEEE, 2016, pp. 334–344, ISBN: 978-1-5090-3806-0. DOI: 10.1109/ICSME.2016.31.

[249] Elsevier, *Scopus*, 2023. [Online]. Available: https://www.scopus.com.

[250] Y. Gao, X. Gu, H. Zhang, H. Lin, and M. Yang, "Runtime performance prediction for deep learning models with graph neural network," in *2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*, IEEE, 2023, pp. 368–380.

[251] G. Yang, C. Shin, J. Lee, Y. Yoo, and C. Yoo, "Prediction of the resource consumption of distributed deep learning systems," *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, vol. 6, no. 2, pp. 1–25, 2022.

[252] H. Li, R. Wu, L. Qian, and H. An, "A systematic methodology for performance characterizing of heterogeneous systems with a dataflow runtime simulator," in *Proceedings of the 2022 4th International Conference on Robotics, Intelligent Control and Artificial Intelligence*, 2022, pp. 629–637.

[253] W. J. Robinson M., F. Esposito, and M. A. Zuluaga, "Dts: A simulator to estimate the training time of distributed deep neural networks," in *2022 30th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*, 2022, pp. 17–24. DOI: 10.1109/MASCOTS56607.2022.00011.

[254] Z. Lin, L. Feng, E. K. Ardestani, *et al.*, "Building a performance model for deep learning recommendation model training on gpus," in *2022 IEEE 29th International Conference on High Performance Computing, Data, and Analytics (HiPC)*, IEEE, 2022, pp. 48–58.

[255] M. Lattuada, E. Gianniti, D. Ardagna, and L. Zhang, "Performance prediction of deep learning applications training in gpu as a service systems," *Cluster Computing*, vol. 25, no. 2, pp. 1279–1302, 2022.

[256] A. Pourali, A. Boukani, and H. Khazaei, *Prenet: Leveraging computational features to predict deep neural network training time*, 2024. arXiv: 2412.15519 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2412.15519.

[257] K. Wang, Z. Zhou, and Z. Li, "Latte: Layer algorithm-aware training time estimation for heterogeneous federated learning," in *Proceedings of the 30th Annual International Conference on Mobile Computing and Networking*, ser. ACM MobiCom '24, Washington D.C., DC, USA: Association for Computing Machinery, 2024, 1470–1484, ISBN: 9798400704895. DOI: 10.1145/3636534.3690705. [Online]. Available: https://doi.org/10.1145/3636534.3690705.

[258] L. Zancato, A. Achille, A. Ravichandran, R. Bhotika, and S. Soatto, "Predicting training time without training," in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, Eds., vol. 33, Curran Associates, Inc., 2020, pp. 6136–6146. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2020/file/440e7c3eb9bbcd4c33c3535354a51605-Paper.pdf.

[259] I. Paun, Y. Moshfeghi, and N. Ntarmos, "Are we there yet? estimating training time for recommendation systems," in *Proceedings of the 1st Workshop on Machine Learning and Systems*, ser. EuroMLSys '21, Online, United Kingdom: Association for Computing Machinery, 2021, 39–47, ISBN: 9781450382984. DOI: 10.1145/3437984.3458832. [Online]. Available: https://doi.org/10.1145/3437984.3458832.

[260] M. Sivakumar, S. Parthasarathy, and T. Padmapriya, "A simplified approach for efficiency analysis of machine learning algorithms," *PeerJ Computer Science*, vol. 10, e2418, 2024.

[261] O. Kwon and J. M. Sim, "Effects of data set features on the performances of classification algorithms," *Expert Systems with Applications*, vol. 40, no. 5, pp. 1847–1857, 2013.

[262] S. Ali and K. A. Smith, "On learning algorithm selection for classification," *Applied Soft Computing*, vol. 6, no. 2, pp. 119–138, 2006.

[263] F. Mohr, M. Wever, A. Tornede, *et al.*, "Predicting machine learning pipeline runtimes in the context of automated machine learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 43, no. 9, pp. 3055–3066, 2021.

[264] X. Hou, Y. Zhao, Y. Liu, *et al.*, *Large Language Models for Software Engineering: A Systematic Literature Review*, 2023. DOI: 10.48550/arXiv.2308.10620. arXiv: 2308.10620 [cs.SE].

[265] J. Shi, Z. Yang, H. J. Kang, *et al.*, "Greening large language models of code," in *Proceedings of the 46th International Conference on Software Engineering: Software Engineering in Society*, ser. ICSE-SEIS'24, Lisbon, Portugal: Association for Computing Machinery, 2024, 142–153, ISBN: 9798400704994. DOI: 10.1145/3639475.3640097.

[266] J. Shi, Z. Yang, B. Xu, *et al.*, "Compressing pre-trained models of code into 3 mb," in *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '22, Rochester, MI, USA: Association for Computing Machinery, 2023, ISBN: 9781450394758. DOI: 10.1145/3551349.3556964.

[267] V. Sanh, L. Debut, J. Chaumond, *et al.*, *Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter*, 2020. arXiv: 1910.01108 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1910.01108.

[268] Y. LeCun, J. Denker, and S. Solla, "Optimal brain damage," in *Advances in Neural Information Processing Systems*, D. Touretzky, Ed., vol. 2, Morgan-Kaufmann, 1989. [Online]. Available: https://proceedings.neurips.cc/paper\%5Ffiles/paper/1989/file/6c9882bbac1c7093bd25041881277658-Paper.pdf.

[269] H. Li, A. Kadav, I. Durdanovic, *et al.*, "Pruning filters for efficient convnets," in *International Conference on Learning Representations*, 2017. [Online]. Available: https://openreview.net/forum?id=rJqFGTslg.

[270] Y. Guo, A. Yao, and Y. Chen, *Dynamic network surgery for efficient dnns*, 2016. arXiv: 1608.04493 [cs.NE]. [Online]. Available: https://arxiv.org/abs/1608.04493.

[271] P. Molchanov, S. Tyree, T. Karras, *et al.*, *Pruning convolutional neural networks for resource efficient inference*, 2017. arXiv: 1611.06440 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1611.06440.

[272] T. Gale, E. Elsen, and S. Hooker, *The state of sparsity in deep neural networks*, 2019. arXiv: 1902.09574 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1902.09574.

[273] D. Guo, S. Ren, S. Lu, *et al.*, *Graphcodebert: Pre-training code representations with data flow*, 2021. arXiv: 2009.08366 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2009.08366.

[274] X. Wei, S. K. Gonugondla, S. Wang, *et al.*, "Towards greener yet powerful code generation via quantization: An empirical study," in *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2023, San Francisco, CA, USA: Association for Computing Machinery, 2023, 224–236, ISBN: 9798400703270. DOI: 10.1145/3611643.3616302.

[275] Z. Sun, X. Du, F. Song, *et al.*, "When neural code completion models size up the situation: Attaining cheaper and faster completion through dynamic model inference," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639120.

[276] F. Magliani, L. Sani, S. Cagnoni, *et al.*, "Genetic algorithms for the optimization of diffusion parameters in content-based image retrieval," in *ICDSC 2019*.

[277] H. Berger, A. Dakhama, Z. Ding, *et al.*, "StableYolo: Optimizing image generation for large language models," in *SSBSE 2023*, ser. LNCS, vol. 14415, pp. 133–139.

[278] Y. Cao, S. Li, Y. Liu, *et al.*, *A comprehensive survey of AI-generated content (AIGC): a history of generative AI from GAN to ChatGPT*, arXiv 2303.04226, 2023.

[279] S. Kim, C. Hooper, T. Wattanawong, *et al.*, *Full stack optimization of transformer inference: A survey*, arXiv 2302.14017, 2023.

[280] S. Alla and S. K. Adari, "What is mlops?" *Beginning MLOps with MLFlow: Deploy Models in AWS SageMaker, Google Cloud, and Microsoft Azure*, pp. 79–124, 2021.

[281] T. Chai and R. R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)?–arguments against avoiding rmse in the literature," *Geoscientific model development*, vol. 7, no. 3, pp. 1247–1250, 2014.

[282] A. De Myttenaere, B. Golden, B. Le Grand, *et al.*, "Mean absolute percentage error for regression models," *Neurocomputing*, vol. 192, pp. 38–48, 2016.

[283] P. Rumao, *Detect Malacious Executable(AntiVirus)*, UCI Machine Learning Repository, 2016. DOI: 10.24432/C5531V.

[284] *APS Failure at Scania Trucks*, UCI Machine Learning Repository, 2017. DOI: 10.24432/C51S51.

[285] I. Guyon, S. Gunn, A. Ben-Hur, *et al.*, *Arcene*, UCI Machine Learning Repository, 2008. DOI: 10.24432/C58P55.

[286] I. Guyon, S. Gunn, A. Ben-Hur, *et al.*, *Dexter*, UCI Machine Learning Repository, 2008. DOI: 10.24432/C5P898.

[287] C. A. Ratanamahatana and D. Gunopulos, "Scaling up the naive Bayesian classifier: Using decision trees for feature selection," 2002, Publisher: Citeseer.

[288] A. Vaswani, N. Shazeer, N. Parmar, *et al.*, "Attention is all you need," in *Advances in Neural Information Processing Systems*, I. Guyon, U. V. Luxburg, S. Bengio, *et al.*, Eds., vol. 30, Curran Associates, Inc., 2017. [Online]. Available: https://proceedings.neurips.cc/paper\%5Ffiles/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf.

[289] A. Radford, J. Wu, R. Child, *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019.

[290] C. Raffel, N. Shazeer, A. Roberts, *et al.*, *Exploring the limits of transfer learning with a unified text-to-text transformer*, 2023. arXiv: 1910.10683 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1910.10683.

[291] OpenAI, *Openai codex*, 2019. arXiv: 1810.04805 [cs.CL]. [Online]. Available: https://arxiv.org/abs/1810.04805.

[292] Y. Wang, H. Le, A. Gotmare, *et al.*, "CodeT5+: Open code large language models for code understanding and generation," in *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, H. Bouamor, J. Pino, and K. Bali, Eds., Singapore: Association for Computational Linguistics, 2023, pp. 1069–1088. DOI: 10.18653/v1/2023.emnlp-main.68.

[293] M. Chen, J. Tworek, H. Jun, *et al.*, *Evaluating large language models trained on code*, 2021. arXiv: 2107.03374 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2107.03374.

[294] J. Gu, P. Salza, and H. C. Gall, "Assemble Foundation Models for Automatic Code Summarization," English, in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*, ISSN: 1534-5351, Los Alamitos, CA, USA: IEEE Computer Society, 2022, pp. 935–946, ISBN: 978-1-66543-786-8. DOI: 10.1109/SANER53432.2022.00112.

[295] X. Zhou, D. Han, and D. Lo, "Assessing generalizability of codebert," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, 2021, pp. 425–436. DOI: 10.1109/ICSME52107.2021.00044.

[296] Y. Ding, Y. Fu, O. Ibrahim, *et al.*, *Vulnerability detection with code language models: How far are we?* 2024. arXiv: 2403.18624 [cs.SE]. [Online]. Available: https://arxiv.org/abs/2403.18624.

[297] Q. Zhang and B. Wu, "Software defect prediction via transformer," in *2020 IEEE 4th Information Technology, Networking, Electronic and Automation Control Conference (ITNEC)*, IEEE, vol. 1, 2020, pp. 874–879.

[298] J. Chen, X. Hu, Z. Li, *et al.*, "Code search is all you need? improving code suggestions with code search," in *Proceedings of the IEEE/ACM 46th International Conference on Software Engineering*, ser. ICSE '24, Lisbon, Portugal: Association for Computing Machinery, 2024, ISBN: 9798400702174. DOI: 10.1145/3597503.3639085.

[299] W. Wang, Y. Zhang, Z. Zeng, *et al.*, "Trans^ 3: A transformer-based framework for unifying code summarization and code search," *arXiv preprint arXiv:2003.03238*, 2020.

[300] D. Chicco and G. Jurman, "The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation," *BMC genomics*, vol. 21, pp. 1–13, 2020.

[301] K. Papineni, S. Roukos, T. Ward, *et al.*, "Bleu: A Method for Automatic Evaluation of Machine Translation," in *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, P. Isabelle, E. Charniak, and D. Lin, Eds., Philadelphia, Pennsylvania, USA: Association for Computational Linguistics, 2002, pp. 311–318. DOI: 10.3115/1073083.1073135.

[302] T. Zhang, V. Kishore, F. Wu, *et al.*, "Bertscore: Evaluating text generation with bert," *arXiv preprint arXiv:1904.09675*, 2019.

[303] O. Chapelle, D. Metlzer, Y. Zhang, *et al.*, "Expected reciprocal rank for graded relevance," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, ser. CIKM '09, Hong Kong, China: Association for Computing Machinery, 2009, 621–630, ISBN: 9781605585123. DOI: 10.1145/1645953.1646033.

[304] Y. Zhou, S. Liu, J. Siow, *et al.*, "Devign: Effective vulnerability identification by learning comprehensive program semantics via graph neural networks," in *Advances in Neural Information Processing Systems*, 2019, pp. 10 197–10 207.

[305] H. Husain, H.-H. Wu, T. Gazit, *et al.*, "Codesearchnet challenge: Evaluating the state of semantic code search," *arXiv preprint arXiv:1909.09436*, 2019.

[306] S. Lu, D. Guo, S. Ren, *et al.*, "Codexglue: A machine learning benchmark dataset for code understanding and generation," *CoRR*, vol. abs/2102.04664, 2021.

[307] A. Kumar, A. M. Shaikh, Y. Li, *et al.*, "Pruning filters with l1-norm and capped l1-norm for cnn compression," *Applied Intelligence*, vol. 51, pp. 1152–1160, 2021.

[308] L. Traini, V. Cortellessa, D. Di Pompeo, *et al.*, "Towards effective assessment of steady state performance in java software: Are we there yet?" *Empirical Software Engineering*, vol. 28, no. 1, p. 13, 2022. DOI: 10.1007/s10664-022-10247-x.

[309] M. Jangali, Y. Tang, N. Alexandersson, *et al.*, "Automated generation and evaluation of jmh microbenchmark suites from unit tests," *IEEE Transactions on Software Engineering*, vol. 49, no. 4, pp. 1704–1725, 2023. DOI: 10.1109/TSE.2022.3188005.

[310] Z. Zhang, Z. Xing, X. Xia, *et al.*, "Faster or slower? performance mystery of python idioms unveiled with empirical evidence," in *Proceedings of the 45th International Conference on Software Engineering*, ser. ICSE '23, Melbourne, Victoria, Australia: IEEE Press, 2023, 1495–1507, ISBN: 9781665457019. DOI: 10.1109/ICSE48619.2023.00130.

[311] C. Laaber, S. Würsten, H. C. Gall, *et al.*, "Dynamically reconfiguring software microbenchmarks: Reducing execution time without sacrificing result quality," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2020, Virtual Event, USA: Association for Computing Machinery, 2020, 989–1001, ISBN: 9781450370431. DOI: 10.1145/3368089.3409683.

[312] T. Kalibera and R. Jones, "Rigorous benchmarking in reasonable time," in *Proceedings of the 2013 International Symposium on Memory Management*, ser. ISMM '13, Seattle, Washington, USA: Association for Computing Machinery, 2013, 63–74, ISBN: 9781450321006. DOI: 10.1145/2464157.2464160.

[313] T. Kalibera and R. Jones, "Quantifying performance changes with effect size confidence intervals," University of Kent, Technical Report 4–12, 2012, p. 55. [Online]. Available: http://www.cs.kent.ac.uk/pubs/2012/3233.

[314] R. F. Woolson, "Wilcoxon signed-rank test," *Encyclopedia of Biostatistics*, vol. 8, 2005.

[315] L. Traini, D. Di Pompeo, M. Tucci, *et al.*, "How software refactoring impacts execution time," *ACM Trans. Softw. Eng. Methodol.*, vol. 31, no. 2, 2021, ISSN: 1049-331X. DOI: 10.1145/3485136.

[316] T. Zimmermann, N. Nagappan, and L. Williams, "Searching for a needle in a haystack: Predicting security vulnerabilities for windows vista," in *Proceedings of the 3rd International Conference on Software Testing, Verification and Validation (ICST)*, Most Influential Paper Award at ICST 2020 MIP Practical, IEEE, 2010.

[317] M. Jimenez, R. Rwemalika, M. Papadakis, *et al.*, "The importance of accounting for real-world labelling when predicting software vulnerabilities," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2019, Tallinn, Estonia: Association for Computing Machinery, 2019, 695–705, ISBN: 9781450355728. DOI: 10.1145/3338906.3338941.

[318] W. Sun, Y. Miao, Y. Li, *et al.*, *Source Code Summarization in the Era of Large Language Models*, en, 2024. [Online]. Available: http://arxiv.org/abs/2407.07959 (visited on 10/05/2024).

[319] Y. Xie, J. Lin, H. Dong, *et al.*, "Survey of code search based on deep learning," *ACM Trans. Softw. Eng. Methodol.*, vol. 33, no. 2, 2023, ISSN: 1049-331X. DOI: 10.1145/3628161.

[320] M. Nagel, R. A. Amjad, M. van Baalen, *et al.*, *Up or down? adaptive rounding for post-training quantization*, 2020. arXiv: 2004.10568 [cs.LG]. [Online]. Available: https://arxiv.org/abs/2004.10568.

[321] P. Ganesh, Y. Chen, X. Lou, *et al.*, "Compressing large-scale transformer-based models: A case study on bert," *Transactions of the Association for Computational Linguistics*, vol. 9, pp. 1061–1080, 2021.

[322] Z. Liu, M. Sun, T. Zhou, *et al.*, *Rethinking the value of network pruning*, 2019. arXiv: 1810.05270 [cs.LG]. [Online]. Available: https://arxiv.org/abs/1810.05270.

[323] A. Maricq, D. Duplyakin, I. Jimenez, *et al.*, "Taming performance variability," in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, Carlsbad, CA: USENIX Association, 2018, pp. 409–425, ISBN: 978-1-939133-08-3. [Online]. Available: https://www.usenix.org/conference/osdi18/presentation/maricq.

[324] S. E. Reed, Z. Akata, X. Yan, *et al.*, "Generative adversarial text to image synthesis," in *ICML 2016*, pp. 1060–1069.

[325] F. Sarro, "Search-based software engineering in the era of modern software systems," in *IEEE International Requirements Engineering Conference*, 2023, pp. 3–5.

[326] J. Redmon, S. K. Divvala, R. B. Girshick, *et al.*, "You only look once: Unified, real-time object detection," in *CVPR 2016*, pp. 779–788.

[327] B. Ji, H. Huang, and S. S. Yu, "An enhanced NSGA-II for solving berth allocation and quay crane assignment problem with stochastic arrival times," *IEEE Trans.*,

[328] T. A. Rashid, J. Majidpour, R. Thinakaran, *et al.*, "NSGA-II-DL: metaheuristic optimal feature selection with deep learning framework for HER2 classification in breast cancer," *IEEE Access*,

[329] A. P. Guerreiro, C. M. Fonseca, and L. Paquete, "The hypervolume indicator: Computational problems and algorithms," *ACM Computing Surveys*, vol. 54, no. 6, pp. 1–42, 2021.

[330] Z. Ding, J. Chen, and W. Shang, "Towards the use of the readily available tests from the release pipeline as performance tests: Are we there yet?," ser. ICSE '20, 1435–1446.

[331] M. Robeer, G. Lucassen, J. M. E. Van Der Werf, F. Dalpiaz, and S. Brinkkemper, "Automated extraction of conceptual models from user stories via nlp," in *2016 IEEE 24th international requirements engineering conference (RE)*, IEEE, 2016, pp. 196–205.

[332] R. Moussa, G. Guizzo, and F. Sarro, "MEG: Multi-objective Ensemble Generation for Software Defect Prediction," en, in *Proceedings of the 16th ACM / IEEE International Symposium on Empirical Software Engineering and Measurement*, Helsinki Finland: ACM, Sep. 2022, pp. 159–170, ISBN: 978-1-4503-9427-7. DOI: 10.1145/3544902.3546255. [Online]. Available: https://dl.acm.org/doi/10.1145/3544902.3546255 (visited on 03/12/2024).