Fairness Analysis of Machine Learning-Based Code Reviewer Recommendation

Mohammad Mahdi Mohajer[1], Alvine Boaye Belle[1], Nima Shiri Harzevili[1], Junjie Wang[2], Hadi Hemmati[1], Song Wang[1], and Zhen Ming (Jack) Jiang[1]

[1] York University, Toronto, ON, Canada
{mmm98, nshiri, hemmati, wangsong}@yorku.ca,
alvine.belle@lassonde.yorku.ca, zmjiang@cse.yorku.ca
[2] Institute of Software, Chinese Academy of Sciences, Beijing, China
junjie@iscas.ac.cn

Abstract. Ensuring the fairness of machine learning (ML) applications is critical to the reliability of modern artificial intelligence systems. Despite extensive study on this topic, the fairness of ML models in the software engineering (SE) domain has not yet been explored well. As a result, many ML-powered software systems, particularly those utilized in the software engineering community, continue to be prone to fairness issues. Taking one of the typical SE tasks, i.e., code reviewer recommendation, as a subject, this paper investigates the fairness of ML applications in the SE domain, specifically focusing on the code reviewer recommendation task. Our empirical study demonstrates that existing ML-based code reviewer recommendation systems exhibit unfairness and discriminating behaviors. Specifically, male reviewers get, on average, 7.25% more recommendations than female code reviewers compared to their distribution in the reviewer set. This paper also investigates why the studied ML-based code reviewer recommendation systems are unfair and provides solutions to mitigate the unfairness. For instance, such systems may recommend male reviewers at a significantly higher rate than female reviewers in a discriminatory manner. Our study further indicates that the existing mitigation methods can enhance fairness significantly in projects with a similar distribution of protected and privileged groups. Still, their effectiveness in improving fairness on imbalanced or skewed data is limited.

Keywords: Fairness · Machine Learning · Code Reviewer Recommendation

1 Introduction

Machine Learning (ML) approaches and models are increasingly being used in the development of modern software [50] to assist developers in different tasks, e.g., defect prediction [47,46], software bug triage [29], and code reviewer recommendation [13,38], etc. Meanwhile, the wide adoption of ML has given rise

to new concerns and issues regarding the trustworthiness and ethicality of such systems, one of which is the issue of fairness [40,33].

Despite the fact that previous studies [26,8,22,10,11,7,45,52,9,42] have extensively examined the fairness of ML applications, while most of these studies mainly focus on general ML applications, little is known about the fairness of ML applications in software engineering domain, e.g., automated bug triage [29,53]. In this work, we take one of the typical SE tasks, i.e., code reviewer recommendation as a subject to explore the fairness of ML applications in SE domain. Specifically, code reviewer recommendation systems are widely used in modern software development to identify the most appropriate code reviewers for a code change. Recently, many ML-based code reviewer recommendation systems have been proposed. For instance, Patanamon et al. proposed RevFinder that used a similarity of previously reviewed file path to recommend an appropriate code-reviewer [44] and Pandya et al. [38] proposed CORMS, which leveraged similarity analysis and support vector machine (SVM) models to recommend reviewers. Although these examined code reviewer recommendation approaches can achieve good performance, none of the fairness characteristics (e.g., race, age, and gender) were considered when recommending reviewers. As a result, there may be bias or fairness issues in such systems that have not previously been investigated, which can potentially harm reviewers' activities. [9,40,42]. For example, in such systems, the final recommendation list may exhibit a need for more representation of female reviewers, as they might be recommended less frequently than their male counterparts in a biased manner.

To address the aforementioned concerns, in this paper, we look into the problem of assessing fairness issues in ML-based code reviewer recommendation systems. Specifically, we conduct an empirical study on two recent ML-based code reviewer recommendation systems, RevFinder [44] and CORMS [38], and we use the same dataset from CORMS [38] to build these systems and run our experiments. Note that, when exploring the fairness of ML-based code reviewer recommendation systems, we only consider the factor of gender. This is mainly because collecting data to identify sensitive factors such as age or race is difficult, as reviewers and code review platforms often do not disclose this information. Moreover, since obtaining non-binary gender information is difficult, and in line with previous fairness research [26,40,11,10,33] that used gender as the sensitive attribute, we are treating gender as a binary attribute in our analysis. Nevertheless, it's essential to emphasize that our study's scope is not confined to binary values and can be expanded to include non-binary genders if we acquire adequate data (details are provided in 3.2). Our experimental results show that both RevFinder and CORMS have unfair behavior in their recommendations. Specifically, they favor male reviewers over female reviewers. For example, we observe that in the Node.js project, male reviewers recommended by CORMS have approximately 85% more chance of being recommended for new code review requests, which is 16% more than the fair condition (details are in Section 4). We further explore the underlying factors that contribute to the unfairness of ML-based code reviewer recommendation systems, e.g., popularity bias, and whether

the existing unfairness mitigation approaches [51,21,17,41] can help improve the fairness of these systems. Our experiment results show that the existing mitigation approaches can improve fairness, but not consistently across all projects. This study contributes to creating fairer ML-based code reviewer recommendation systems, reducing gender-based discrimination, and promoting societal equality and fairness. Those creating systems that engage with humans can apply the findings and methodology from our research to ensure their software products are equitable and contribute to equality among users. The methods used in our study also apply to other types of recommendation systems, such as team recommendation systems for collaborative software development [2]. As a summary, this paper makes the following contributions:

- We conduct an empirical study to investigate the fairness of two recent ML-based code reviewer recommendation systems.
- We analyze the underlying factors that influence the outcomes of code reviewer recommendation systems and demonstrate that the existing unfairness mitigation methods can be utilized to alleviate the unfairness in ML-based code reviewer recommendation systems.
- We show that the current mitigation approaches have limitations in terms of fairness improvement for projects with imbalanced or skewed data.
- We release the dataset and source code of our experiments to help other researchers replicate and extend our study[3].

2 Background and Related Work

2.1 Code Reviewer Recommendation System

A code reviewer recommendation system is a software application that supports software development teams in identifying the most appropriate code reviewers for a particular code change request by suggesting a list of the most qualified candidates to conduct the review request [16].

While the first code reviewer recommendation system already utilized machine learning techniques when it was introduced [23], earlier systems often relied on heuristic approaches [37,3], such as graph and search-based approaches. As an example, Ounti et al. [37] proposed RevRec, a recommendation system that uses a genetic algorithm to find an appropriate peer reviewer for a code change. As the field progressed, newer systems increasingly employed more machine learning methods [49,43,14,38] e.g., SVM (Support Vector Machines), collaborative filtering, and Naive Bayes, to improve their recommendations. These reviewer recommendation systems use different factors and features for determining the most qualified reviewer for a review request, such as file similarity, developers' expertise, social relations, developers' activeness, etc [16].

In this work, we select two state-of-the-art ML-based code reviewer recommendation systems, i.e., RevFinder [44] and CORMS [38], as our research subjects to explore their fairness (details are in Section 3.3). These code reviewer

[3] `https://doi.org/10.5281/zenodo.11054911`

recommendation systems have been shown to outperform their prior baselines and have been found effective in recommending code reviewers.

2.2 Fairness Analysis in Machine Learning Application

In machine learning applications, there are two common types of fairness [33,40]: i.e., 1) Group Fairness, which ensures that different groups of people, including protected groups and privileged groups, are treated similarly and fairly. For example, in cases where gender is not a deciding factor, the female group should be treated similarly to the other groups (e.g., the male group) [33,40] and 2) Individual Fairness which ensures that individuals that are similar based on a criterion should be treated fairly and similarly [33,40,1]. For example, regardless of demographic background, each applicant in the employment process should be treated equitably. In this study, we conduct our analysis based on the group fairness definition. Rather than focusing on individual cases, group fairness analysis evaluates the influence of machine learning models on distinct groups of people.

2.3 Unfairness in Recommendation Systems

In this work, we focus on the fairness of recommendation systems. The concepts and definitions of fairness analysis in recommendation systems and general ML applications slightly differ [48]. For example, in previous studies [40,26,33], since all approaches target classification problems, the concept of fairness thoroughly depends on the predictions of the target attribute and its relation to the protected group. On the contrary, the concept of fairness in recommendation systems can be discussed from several points of view, e.g., the fairness of each of the recommended items (item-based fairness) and the fairness of the exposure and quality that each user experiences from the recommended items (user-based fairness) [48]. Also, in recommendation systems, not all of the biases are considered as unfairness, e.g., popularity bias, position bias, and conformity bias [48,12]. The fairness of recommendation systems can be categorized into two groups [48]:

- Process fairness: This means ensuring that the process used to produce recommendations is fair and unbiased to all users, regardless of their sensitive attributes.
- Outcome fairness: This means ensuring that the system's outcomes are distributed fairly and proportionately among various groups of people. This means that recommendations should neither favor nor discriminate against any particular group based on their sensitive attributes.

In this study, we focus on investigating **outcome fairness** in ML-powered code reviewer recommendation systems, specifically we examine the **item-based fairness** through **group fairness** analysis. In particular, the items being recommended in our case are code reviewers. Therefore, our research focuses on assessing whether the final list of recommendations upholds fairness with regards to these recommended code reviewers.

2.4 Unfairness Mitigation Techniques

To mitigate the unfairness in ML applications, many unfairness mitigation mechanisms have been proposed [33,40], which can be categorized into three major types, i.e., pre-processing strategies [41,17] (by modifying training data before it is utilized for training), in-processing approaches [5,28] (by altering the machine-learning algorithm to increase fairness), post-processing techniques [24,35] (by updating output scores and predictions of the machine learning models). In this work, we focus on the "post-processing" category of fairness mitigation methods and exclude both "in-processing" and "pre-processing" mitigation approaches. "in-processing" approaches require non-trivial changes to the code reviewer recommendation systems, which are not generalizable. Additionally, the majority of "pre-processing" approaches are unsuitable for the recommendation tasks employed in our study subjects, given the inherent characteristics of our data, thus not suitable for fairness study in recommendation systems [41,17]. In many "pre-processing" methods, unfairness is often identified by analyzing proxy attributes, which are features correlated to sensitive attributes [26,40,33]. These methods heavily rely on numerical features. However, in the code reviewer recommendation systems we studied, the features are not numerical. Additionally, vector embedding as a way to calculate the score and rank the code reviewers is not used among both recommendation systems, as only CORMS uses this approach partially in its hybrid workflow [38]. Instead, using string-matching methods, RevFinder and CORMS determine the reviewers' scores based on the similarity between the file paths used in previous reviews [38,44]. This limited use of vector embedding and the non-numerical nature of the features in our studied systems pose significant challenges unique to our subjects. As a result, many "pre-processing" techniques are not applicable in this specific scenario, highlighting the need to focus more on "post-processing" approaches.

3 Empirical Study Setup

3.1 Research Questions

In this study, we are going to answer the following three research questions (RQs):

RQ1 (Existence of Fairness): Is there any unfairness in the ML-based code reviewer recommendation systems?

RQ2 (Root Cause for Unfairness): What is the root cause for unfairness in the code reviewer recommendation systems?

RQ3 (Effectiveness of Existing Unfairness Mitigation Techniques): Do existing unfairness mitigation approaches work for ML-based code reviewer recommendation systems?

3.2 Data Collection

For our analysis, we need datasets that include human-related information of reviewers, such as their gender, which is our desired sensitive attribute. However,

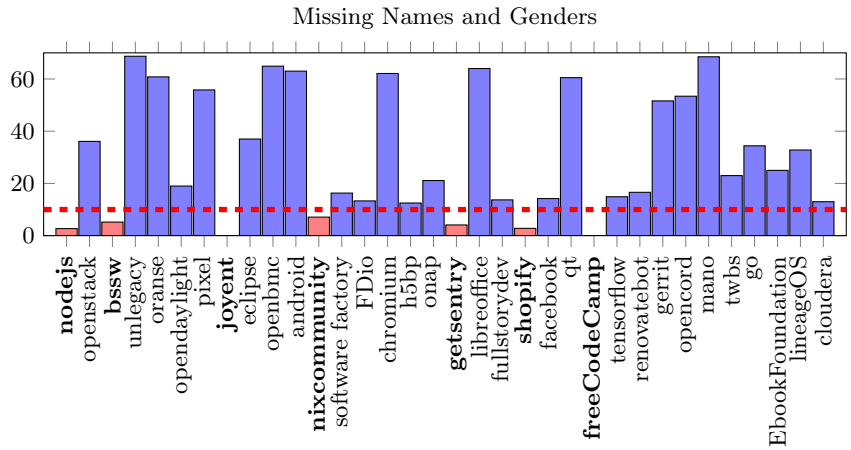

Fig. 1. The percentages of missing names and genders ("unknown") for each project examined in our study. We select projects with at most 10% missing names and genders, which is any project whose bar is beneath the red dashed line.

existing code reviewer recommendation datasets from previous studies [38,44,48] do not include that critical information. To collect the gender information of reviewers, in this work, we propose heuristic methods to infer a reviewer's gender information from their names and other publicly available information online (e.g., homepage, GitHub account information, and LinkedIn profile). Our experiment is conducted on the same dataset as the previous study [38], which includes code review requests from 34 open-source projects. These code review requests are collected from code review platforms like Gerrit and GitHub that are available to the public. As mentioned before, these code review systems do not keep reviewers' human-related information, such as gender, and we cannot directly obtain the genders of the reviewers from the datasets.

For each project in our dataset, we first get reviewers' names through the user ID provided by the code review platform. Then we use the following steps to infer a reviewer's demographic gender.

We first remove projects that have a high percentage of reviewers with missing names. If the field for a reviewer's name is null or blank, we consider this as not having a name mainly because the reviewer did not specify a name on his/her profile. Also, reviewers may have nicknames instead of their real names on their profiles. We consider all these records "unknown" since we cannot infer the gender from them. To distinguish nicknames from real names, we use the following heuristic, i.e., if a name contains any number, symbol, or sign, we consider that a nickname; e.g., in the nixcommunity project, there was a reviewer with the name "jD91mZM2", which has been removed from the data. We discard projects with reviewers who have more than 10% "unknown" names to ensure the quality of our experiment data. The missing rates are demonstrated in Figure 1. As a

Table 1. Details of the four selected projects. The columns "Missing Genders", "Total Ex.", "F. Ratio", "M#", and "F#", are the percentage of missing genders, the total number of examples, the ratio of female reviewers, the number of female reviewers, and the number of male reviewers, respectively.

Project	**Missing Genders**	**Total Ex.**	**F.Ratio**	**F#**	**M#**
nodejs	2.7%	110	0.13	5	28
bssw	5.2%	213	0.67	9	9
getsentry	4.1%	775	0.08	6	64
shopify	2.8%	565	0.06	17	148

result, we selected seven out of 34 projects. In the seven projects examined, there were at most four reviewers who had nicknames on their profiles.

Second, for the selected seven projects, we manually check each reviewer's information through online resources, e.g., personal and institutional homepages, GitHub accounts, and LinkedIn profiles. Some of the reviewers have links to their social media accounts on their GitHub profiles, so we can access their information directly. For the reviewers who do not provide social connections (20% of our reviewer set), we search their full name on the internet to find their social media accounts and get their gender information. During the process, we discard names that point to indistinguishable users (e.g., people who have the same name and similar profiles on social media). We look for gender information in their specified pronouns in their profiles, and their online content, including the pronouns and genders the reviewers themselves or others used to describe them. In this manual analysis, we checked 355 reviewers' information on the internet. These reviewers may possess several reviews and contributions in our dataset, employed for training the models, and the corresponding statistics are displayed in Table 1. According to our analysis, we obtained at least 90% of the reviewers' gender information through this manual analysis. Three authors are involved during this process to ensure the correctness of the manually analyzed results.

Finally, we remove projects for which there is only one reviewer from the protected group, i.e., the female reviewer group. In each remaining project, we exclude records relating to reviewers of unknown genders. After applying our gender identification approach, four out of the 34 candidate projects from [38] are selected for our experiments, details are in Table 1.

3.3 Subjects of Study

As mentioned in Section 2.1, We use two recent existing ML-based code reviewer recommendation systems as the research subjects, i.e., CORMS [38] and RevFinder [44]. RevFinder recommends code reviewers based on a machine learning-based ranking algorithm that learns the scores of different candidates for a given review request based on the similarities in the file locations involved in past code reviews from the dataset. By evaluating the records in the dataset and

propagating the scores of each reviewer involved in review requests with similar file path locations, RevFinder provides a list of unique reviewers and their scores for the given dataset. On the other hand, CORMS is a hybrid code reviewer recommendation system that uses the same similarity model as RevFinder (with additional features) and employs an SVM model that learns from each review subject in the training set. Finally, appropriate reviewers are chosen based on their computed scores and associated ranks corresponding to their scores in the final recommendation list.

Although these two code reviewer recommendation approaches that we have examined can achieve good performance in terms of accuracy of the reviewer recommendation task, none of the fairness characteristics, e.g., race, age, and gender, were considered when recommending reviewers by using these approaches. In this study, the female reviewer group is considered the protected group. Furthermore, we ensure the validity of our findings by replicating the configurations of the selected two code reviewer recommendation systems as specified in their respective publications [44,38].

It is worth noting that we are selecting the mentioned subjects because they are suitable for datasets that either contain human-related information or support indirectly adding human-related information to them to facilitate our analysis. Although there were also other different code reviewer recommendation systems, they were either trained on different datasets that are incompatible for fairness analysis or their architecture simply does not comply with our dataset.

3.4 Evaluation Measures

Existing studies [30,19] revealed that improving fairness usually comes at a cost in terms of performance measures such as model accuracy. As a result, in order to demonstrate that an unfairness mitigation strategy is useful to deploy, most studies in fairness analysis will include a performance measure before and after applying the unfairness mitigation technique [40,26,15]. This trade-off can be balanced through different approaches, but it also relies on the context and domain of the application and the extent to which we can sacrifice accuracy for fairness [15,30,25,4,20,34]. It is important to clarify that the objective of our study is to evaluate the fairness of code reviewer recommendation systems, rather than focusing on their performance. Performance measures are solely utilized to assess the influence of unfairness mitigation strategies on the recommendation outcomes, both before and after implementing these strategies. In this work, we follow existing studies [26,40] and use the average of all measure values for each record in our dataset to represent the overall performance.

Fairness Measures To evaluate the fairness of recommendation systems, we use two measures that rely on the top-K results and one measure that is independent of the top-K results [21,48]. Similar to existing works [38,44] that conducted their experiments in a specific setting, we limit the values of K to 4, 6, and 10.

$Skew_{\mathrm{S_i}}@K$: The skew of the ranked list of top-K candidates for a certain value S_i of the sensitive attribute is:

$$Skew_{\mathrm{S_i}}@K(C) = \ln(\frac{P_{\mathrm{C^K,S_i}}}{P_{\mathrm{D,S_i}}}) \tag{1}$$

where C is the ranked list of candidates, C^K is the top-K candidates from C, $P_{\mathrm{C^K,S_i}}$ is the proportion of candidates having the sensitive attribute value S_i in the top-K results, and $P_{\mathrm{D,S_i}}$ is the desired proportion of candidates with the sensitive attribute value S_i in the given dataset.

Statistical Parity Difference For top-K results ($SPD@K$): The statistical parity difference (SPD) is a well-known measure of fairness that is used in many articles about how fair machine learning is when it comes to classification tasks [40,26]. This measure is demonstrated in Eq. 2 as an example for the binary classification:

$$\Big|P[\hat{Y} = 1|S = 1] - P[\hat{Y} = 1|S \neq 1]\Big| \leq \epsilon \tag{2}$$

Nevertheless, to use this measure for top-K results in recommendation systems such as code reviewer recommendation systems, we must change the calculation in Eq. 2. As a result, we introduce the $SPD@K$ measure, which is described in Eq. 3:

$$\Big|P[\hat{Y} \in C^K|S = 1] - P[\hat{Y} \in C^K|S \neq 1]\Big| \leq \epsilon \tag{3}$$

Where C^K is the ranked list of top-K candidates from the ranked list of candidates C as the result of the recommendation. In this measure, we also refer to ϵ as the expected SPD threshold. This threshold will be computed by calculating the absolute difference between the male and female ratios in the dataset, which is described in Eq. 4:

$$\epsilon = \left|\frac{\#\text{ Females}}{\#\text{ Reviewers}} - \frac{\#\text{ Males}}{\#\text{ Reviewers}}\right| \tag{4}$$

The reason that we calculate an expected value for ϵ as depicted in Eq. 4 is that we expect that the difference in recommendation rates for males and females should be similar to the disparity in ratios of these groups in the dataset. Although ϵ can be variable and should usually be identified by domain experts of the application in which the fairness analysis is being conducted [40], Eq. 4 is still a reasonable statistical estimation of the difference between the rates of recommendations for both privileged and unprivileged groups to be in a fair state.

Normalized Discounted Cumulative KL-divergence (NDKL): Eq. 5 describes the normalized discounted cumulative Kullback-Leiber (KL) divergence given a ranked list of the candidates C:

$$NDKL(C) = \frac{1}{Z}\sum_{i \in Ks}\frac{1}{\log_2(i+1)} d_{\mathrm{KL}}(D_{\mathrm{C^i}}||D_d) \tag{5}$$

In Eq. 5 $d_{\mathrm{KL}}(D_{\mathrm{C^i}}||D_d) = \sum_j D_{\mathrm{C^i}}(j) \ln \frac{D_{\mathrm{C^i}}(j)}{D_d}$, $Z = \sum_{\mathrm{i=1}}^{\mathrm{C}} \frac{1}{\log_2(i+1)}$, and $Ks = \{4, 6, 10\}$ where $D_{\mathrm{C^i}}$ and D_d represent the proportion of the top i candidates in the ranked list of candidates C having the sensitive attribute j, respectively, and the desired proportion based on the dataset with the sensitive attribute value j.

Performance Measures In this study, we adopt the identical performance metrics utilized in previous research papers discussing CORMS and RevFinder [38,44], i.e., Top-K Accuracy and Mean Reciprocal Rank (MRR). This enables us to cross-compare and guarantee accurate replications of both works and to ensure a fair assessment before and after applying bias mitigation strategies. In our scenario, we use $MRR@K$ to focus on the top-K recommendations. This means that we only consider the top-K candidates in the original calculations. Every candidate not on the top-K list has a reciprocal rank of zero. The rest of the calculations remain the same.

3.5 Selected Unfairness Mitigation Approaches

Researchers have proposed several unfairness mitigation strategies for recommendation systems [48,51,27,21,6,36,39], but not all of them are applicable to our case due to different views on fairness [48] (more details in Sec. 2.3) and limitations around different techniques such as "in-processing" and "pre-processing" approaches (more details in Sec. 2.4). Also, due to the inherent differences in the fairness of classification and recommendation tasks (e.g., techniques used for classification do not support top-K results), we cannot employ those mitigation techniques for classification tasks in our study [48]. Moreover, each recommendation system can have a distinct and different architecture, and only some mitigation techniques apply to that type of architecture (e.g., reinforcement learning-based approaches may not be suitable for many recommendation systems) [48]. Also, although there were various "post-processing" mitigation approaches in the literature, not all could be easily adapted and applied to our study subjects. For example, the re-ranking approaches proposed by Liu et al. [32] and Naghiaei et al. [36] look at the fairness of recommendation from a multi-sided perspective. In contrast, in our study, we only focus on item-based fairness, which is the fairness of the reviewers recommended for each review request (more details in Sec. 2.3).

Thus, in this work, we select two applicable approaches from the "post-processing" category of fairness mitigation methods (see 2.4) for reviewer recommendation systems [48]. The details of these two mitigation approaches are as follows.

DetGreedy Algorithm Geyik et al. [21] developed this algorithm for fairness-aware recommendation in LinkedIn Talent Search recommendation systems. This algorithm works as follows. Given a top-K list, there are two requirements to satisfy the fairness condition:

a. Min: $\forall K < |C| \wedge \forall s_i \in A, count_K(S_i) \geq \lfloor P_{D,S_i} \cdot K \rfloor$

b. Max: $\forall K < |C| \wedge \forall s_i \in A, count_K(S_i) \leq \lceil P_{D,S_i} \cdot K \rceil$

Where A is the set of attribute values, C is the list of candidates, P_{D,S_i} is the desired proportion of candidates with the attribute value S_i, and $count_k(S_i)$ is the number of candidates with the attribute value S_i in the top-K results. If some candidates are close to not meeting the minimum requirement, select the one with the highest score from that group. If all candidates meet the minimum requirement, choose the one with the highest score among those who have not yet reached their maximum requirements.

DetRelaxed To improve DetGreedy, Geyik et al. [21] further proposed DetRelaxed. While DetGreedy aims to include as many high-scoring candidates as possible in the ranked list, it may not be effective in various scenarios, according to the authors [21]. The DetRelaxed algorithm was proposed to consider all candidates who satisfy the minimum requirement and minimize the term $\left\lceil \frac{\lceil P_{D,S_i} \cdot K \rceil}{P_{D,S_i}} \right\rceil$ to select the candidate with the highest score for the next position.

The formulas and details mentioned for both approaches are based on Geyik et al.'s study [21], which you can refer to for more information. According to this research study [21], these approaches resulted in a huge improvement in fairness at the production stage. Hence, we select these approaches to assess their effectiveness with our subjects.

4 Results and Analysis

4.1 RQ1: Existence of Fairness

Approach: To answer this RQ, we first build RevFinder and CORMS on the training data selected from our experimental dataset (details are in Section 3.2). When training each model, following existing studies [44,38], we divide the dataset into two sections, with 80% for training and the remaining 20% for testing, chronologically. Then, we use the measures mentioned in Section 3.4 to evaluate the fairness of the recommendations generated by these code reviewer recommendation systems based on the testing data.

Result: Table 2 presents the findings of our experiments, where we analyze all the evaluation measures for each project across three different scenarios, i.e., original (i.e., results obtained without any mitigation approaches), DG (i.e., outcomes obtained after using DetGreedy), and DR (i.e., results obtained after using DetRelaxed). Also, it presents the SPD threshold for each project in the dataset. As we can see from column "Original" in the table, $Skew@K$ has negative values under both CORMS and RevFinder on the four projects with different K values, e.g., BSSW, GetSentry, Node.js, and Shopify, which have negative $Skew@K$ values, i.e, -0.19, -0.01, -0.73, and -0.90 under CORMS when recommending top 10 reviewers. For $SPD@K$, as we can see on most projects, the values of $SPD@K$ are above the specified threshold values which indicates that the disparity between the percentages of males and females being recommended

Table 2. Summary of experiment results. The results that are considered unfair recommendations based on the $SPD@K$ and $Skew@K$ measures are highlighted in ● and ●, respectively.

Top-K	Subjects	Measures	Projects											
			BSSW			GetSentry			Node.js			Shopify		
			Original	DG	DR	Original	DG	DR	Original	DG	DR	Original	DG	DR
Top-4	CORMS	TopK-ACC	79%	79%	79%	43%	43%	37%	41%	41%	41%	25%	25%	30%
		MRR@K	0.54	0.54	0.54	0.23	0.23	0.21	0.26	0.26	0.27	0.19	0.19	0.21
		SPD@K	0	0	0	0.93	0.93	1	0.85	0.85	1	0.80	0.87	1
		Skew@K	0.01	0.01	0.01	-1.82	-1.82	-2.25	-1.79	-1.79	-2.7	-1.41	-1.50	-2.35
	RevFinder	TopK-ACC	71%	73%	73%	43%	43%	37%	50%	50%	45%	25%	25%	24%
		MRR@K	0.5	0.5	0.5	0.25	0.25	0.22	0.32	0.32	0.31	0.15	0.15	0.14
		SPD@K	0.09	**0**	**0**	0.90	**0.88**	1	0.52	0.56	1	0.97	**0.96**	1
		Skew@K	-0.001	**0.01**	**0.01**	-1.48	-1.48	-2.12	0.15	**0.09**	-2.71	-2.03	**-1.94**	-2.17
Top-6	CORMS	TopK-ACC	84%	87%	87%	44%	44%	38%	52%	52%	47%	34%	36%	37%
		MRR@K	0.55	0.56	0.56	0.24	0.24	0.21	0.29	0.29	0.28	0.21	0.21	0.22
		SPD@K	0.31	**0**	**0**	0.87	0.87	1	0.86	0.86	**0.66**	0.80	0.87	1
		Skew@K	-0.35	**0.01**	**0.01**	-1.20	-1.20	-2.25	-1.56	-1.56	**0.12**	-1.14	-1.28	-2.35
	RevFinder	TopK-ACC	88%	83%	83%	49%	49%	46%	54%	54%	54%	36%	36%	37%
		MRR@K	0.53	0.52	0.52	0.26	0.26	0.24	0.33	0.33	0.33	0.17	0.17	0.17
		SPD@K	0.13	**0**	**0**	0.81	**0.80**	1	0.60	0.69	0.66	0.97	**0.96**	1
		Skew@K	-0.14	**0.01**	**0.01**	-0.53	-0.58	-2.12	0.07	-0.1	0.15	-1.97	**-1.87**	-2.17
Top-10	CORMS	TopK-ACC	94%	100%	100%	45%	46%	45%	58%	58%	58%	42%	43%	46%
		MRR@K	0.57	0.57	0.57	0.24	0.24	0.24	0.29	0.29	0.29	0.22	0.22	0.22
		SPD@K	0.2	**0**	**0**	0.79	0.82	0.8	0.82	**0.78**	**0.8**	0.77	**0.73**	0.8
		Skew@K	-0.19	**0.01**	**0.01**	-0.01	-0.1	**0.14**	-0.73	**-0.31**	**-0.35**	-0.90	**0.26**	**0.04**
	RevFinder	TopK-ACC	88%	92%	92%	62%	62%	55%	68%	68%	68%	44%	44%	43%
		MRR@K	0.53	0.53	0.53	0.28	0.28	0.25	0.35	0.35	0.35	0.18	0.18	0.17
		SPD@K	0.17	**0**	**0**	0.81	0.82	1	0.68	0.68	0.8	0.96	0.96	1
		Skew@K	-0.19	**0.01**	**0.01**	-0.03	-0.16	-2.12	0.06	0.06	-0.32	-1.83	**-1.77**	-2.17
	CORMS	NDKL	0.04	**0.01**	**0.01**	0.08	0.08	0.08	0.11	**0.10**	**0.08**	0.14	**0.08**	**0.09**
	RevFinder	NDKL	0.04	**0.01**	**0.01**	0.07	0.07	0.09	0.06	0.06	0.08	0.09	0.09	0.10
	SPD Threshold		0			0.82			0.69			0.79		

in BSSW, GetSentry, Node.js, and Shopify projects is 15%, 3%, 3%, and 8%, respectively. Overall, considering the different $SPD@K$ metric variations and their corresponding expected SPD threshold, male reviewers get an average of 7.25% more recommendations than female code reviewers compared to their distribution in the reviewer set. This calculation involves averaging across all the mentioned variations. Results from $Skew@K$ and $SPD@K$ indicate there are unfairness issues for each of the four experimental projects under both CORMS and RevFinder.

Furthermore, we analyze the normalized discounted KL divergence measure (NDKL), which is shown in Table 2. Our experiments indicate that the NDKL measure yielded positive values for all projects, suggesting that the outcomes may be biased towards a particular value of the sensitive attribute.

Answer to RQ1: In the examined projects, male reviewers get an average of 7.25% more recommendations than female code reviewers, considering their distribution in the recommendation list under both CORMS and RevFinder. Our experiment results suggest that the two ML-based code reviewer recommendation systems studied exhibit bias towards gender.

4.2 RQ2: Root Cause for Unfairness

Approach: To answer this question, we first conduct an exclusive analysis of current fairness studies in the literature to collect all the possible factors that have been examined to be effective in determining the fairness of ML applications [33,12,40,54,48], especially recommendation systems. Three factors were collected: (1) imbalanced or skewed data, (2) popularity bias, and (3) algorithmic objectives.

Note that we exclude the factor of algorithmic objectives since it requires us to examine the code reviewer recommendation architectures, which is out of the scope of this paper. As a result, we employ the factors of imbalanced or skewed data and popularity bias to explore the possible root causes of code reviewer recommendation systems. Imbalanced or skewed data can lead to unfairness because models trained on such data have a strong probability of learning behavior towards over-represented groups, eventually becoming overfitted to them [48,12]. Popularity bias and unfairness also have a strong connection with each other [12,54]. The "long-tail effect" occurs when recommendation systems favor popular items over less popular ones [18,31], which leads to discrimination against the less popular items. If the less popular items are generally from the protected group, popularity bias will turn into unfairness.

Result: The following are the primary factors responsible for the unfairness and disparities between actual and expected distributions of the protected group in the outputs of the code reviewer recommendation system:

Imbalanced or Skewed Data: Our analysis indicates that the imbalanced representation of male and female reviewers in some projects is one of the reasons behind the unfair outcomes of code reviewer recommendation systems. Table 1 shows that, apart from the BSSW project, which has an equal number of female and male reviewers, the number of male reviewers is higher than that of female reviewers in all other projects (males are approximately 8 times more than females), which leads to skewed data. The BSSW project has a record of females accounting for 67% of the total, while in the Node.js, GetSentry, and Shopify projects, this percentage is 13%, 8%, and 6%, respectively. Consequently, Table 2 reveals that the results of the $SPD@K$ measure for the BSSW project are substantially lower than those of other projects, suggesting a fairer outcome. The same pattern is observed for the $Skew@K$ measure in the BSSW project, as other projects exhibit more negative values for this measure, indicating larger discrepancies between the current and fair conditions.

Popularity Bias: In the projects examined, bias was found to be more prevalent in projects where male reviewers were more popular than female reviewers. For instance, in Shopify and Node.js, the first 14% and 15% of popular reviewers were all male, respectively. This led to unfairness in the CORMS and RevFinder recommendation systems, which learned and incorporated this preference for male reviewers into their decision-making processes. In contrast, in the BSSW project, the first two popular reviewers were female, resulting in fairness measures like $Skew@K$ and $SPD@K$ being closer to fair values, such as $Skew@K$ being closer to zero.

Answer to RQ2: We confirm that popularity bias and imbalanced or skewed data are two factors that can affect code reviewer recommendation systems, both of which can lead to unfairness. Projects that do not have these issues (e.g., BSSW) result in values in fairness measures that are closer to a fair state, in contrast to projects that do suffer from these problems.

4.3 RQ3: Effectiveness of Existing Unfairness Mitigation Techniques

Approach: In this RQ, we examine the selected unfairness mitigation approaches (i.e., DetGreedy and DetRelaxed) in Section 3.5 to see if they could improve the fairness of code reviewer recommendation systems. Furthermore, because applying an unfairness mitigation technique has been shown to have a trade-off with performance measures [15,30], we should also guarantee that applying these mechanisms does not adversely affect performance measures. As a result, we compare performance measures before and after using unfairness mitigation techniques.
Result: The fairness measures after applying these two mitigation approaches, i.e., DetGreedy and DetRelaxed, for each project, are shown in Table 2.

As we can see from the table, both DetGreedy and DetRelaxed mitigation techniques can improve fairness, but not across all projects. The bolded values on this table indicate fairness improvements compared to original settings. With the use of these approaches, the BSSW project saw a significant fairness improvement of 100%. Also, in the BSSW project, RevFinder's top-K accuracy decreased by 5% solely in the top-6 scenario. While performance measures were not adversely impacted (only a 1.75% decrease on average on all projects), the fairness enhancement was not as noticeable in projects with an imbalanced distribution of male and female reviewers (e.g., Node.js, GetSentry, and Shopify). Thus, to mitigate unfairness in code reviewer recommendation systems, we should select a mitigation technique that aligns with the dataset's characteristics or performs independently of those characteristics.

Answer to RQ3: DetGreedy and DetRelaxed mitigation approaches can improve fairness while maintaining performance, but not consistently across all projects.

5 Conclusion

This paper represents a novel investigation into the fairness issue of machine learning-based code reviewer recommendation systems. Specifically, two recent existing systems (CORMS and RevFinder) and code review data from four open-source projects were used to conduct the fairness analysis. Our empirical study demonstrates that current ML-based code reviewer recommendation techniques exhibit unfairness and discriminating behaviors. This paper also discusses the reasons why the studied ML-based code reviewer recommendation systems are unfair and provides solutions to mitigate the unfairness.

References

1. How do fairness definitions fare? examining public attitudes towards algorithmic definitions of fairness. AIES 2019 pp. 99–106 (1 2019)
2. Automatic team recommendation for collaborative software development. Empirical Software Engineering **26**, 1–53 (7 2021)
3. Rstrace+: Reviewer suggestion using software artifact traceability graphs. Information and Software Technology **130**, 106455 (2021)
4. Bechavod, Y., Ligett, K.: Penalizing unfairness in binary classification. arXiv preprint arXiv:1707.00044 (2017)
5. Beutel, A., Chen, J., Doshi, T., Qian, H., Wei, L., Wu, Y., Heldt, L., Zhao, Z., Hong, L., Chi, E.H., Goodrow, C.: Fairness in recommendation ranking through pairwise comparisons. In: SIGKDD. p. 2212–2220. KDD '19 (2019)
6. Biega, A.J., Gummadi, K.P., Weikum, G.: Equity of attention: Amortizing individual fairness in rankings. 41st International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2018 **18**, 405–414 (6 2018). https://doi.org/10.1145/3209978.3210063, https://dl.acm.org/doi/10.1145/3209978.3210063
7. Biswas, S., Rajan, H.: Do the machine learning models on a crowd sourced platform exhibit bias? an empirical study on model fairness. ESEC/FSE 2020 pp. 642–653 (2020)
8. Biswas, S., Rajan, H.: Fair preprocessing: Towards understanding compositional fairness of data transformers in machine learning pipeline. ESEC/FSE 2021 **21**, 981–993 (2021)
9. Brun, Y., Meliou, A.: Software fairness. p. 754–759. ESEC/FSE 2018 (2018)
10. Chakraborty, J., Majumder, S., Menzies, T.: Bias in machine learning software: Why? how? what to do? ESEC/FSE 2021 pp. 429–440 (8 2021)
11. Chakraborty, J., Majumder, S., Yu, Z., Menzies, T.: Fairway: a way to build fair ml software. pp. 654–665 (11 2020)
12. Chen, J., Dong, H., Wang, X., Feng, F., Wang, M., Dong, H., Wang, X., Feng, F., He, X.: Bias and debias in recommender system: A survey and future directions. ACM Transactions on Information Systems **41**, 67 (2 2023)
13. Chouchen, M., Ouni, A., Mkaouer, M.W., Kula, R.G., Inoue, K.: Whoreview: A multi-objective search-based approach for code reviewers recommendation in modern code review. Applied Soft Computing **100**, 106908 (2021)
14. Chueshev, A., Lawall, J., Bendraou, R., Ziadi, T.: Expanding the number of reviewers in open-source projects by recommending appropriate developers. In: ICSME 2020. pp. 499–510 (2020)
15. Corbett-Davies, S., Pierson, E., Feller, A., Goel, S., Huq, A.: Algorithmic decision making and the cost of fairness. In: KDD '17. p. 797–806. KDD '17, Association for Computing Machinery (2017)
16. Davila, N., Nunes, I.: A systematic literature review and taxonomy of modern code review. Journal of Systems and Software **177**, 110951 (7 2021)
17. Ekstrand, M.D., Tian, M., Azpiazu, I.M., Ekstrand, J.D., Anuyah, O., McNeill, D., Pera, M.S.: All the cool kids, how do they fit in?: Popularity and demographic biases in recommender evaluation and effectiveness. In: Friedler, S.A., Wilson, C. (eds.) PMLR. Proceedings of Machine Learning Research, vol. 81, pp. 172–186. PMLR (23–24 Feb 2018)
18. Ferraro, A.: Music cold-start and long-tail recommendation: Bias in deep representations. p. 586–590. RecSys '19 (2019)

19. Friedler, S.A., Scheidegger, C., Venkatasubramanian, S., Choudhary, S., Hamilton, E.P., Roth, D.: A comparative study of fairness-enhancing interventions in machine learning. In: Proceedings of the Conference on Fairness, Accountability, and Transparency. p. 329–338. FAT* '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3287560.3287589, https://doi.org/10.1145/3287560.3287589
20. Friedler, S.A., Scheidegger, C., Venkatasubramanian, S., Choudhary, S., Hamilton, E.P., Roth, D.: A comparative study of fairness-enhancing interventions in machine learning. In: Proceedings of the Conference on Fairness, Accountability, and Transparency. p. 329–338. FAT* '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3287560.3287589, https://doi.org/10.1145/3287560.3287589
21. Geyik, S.C., Ambler, S., Kenthapadi, K.: Fairness-aware ranking in search and recommendation systems with application to linkedin talent search. KDD'19 pp. 2221–2231 (7 2019)
22. Hort, M., Zhang, J.M., Sarro, F., Harman, M.: Fairea: A model behaviour mutation approach to benchmarking bias mitigation methods. ESEC/FSE 2021 pp. 994–1006 (8 2021)
23. Jeong, G., Kim, S., Zimmermann, T., Yi, K.: Improving code review by predicting reviewers and acceptance of patches. Research on software analysis for error-free computing center Tech-Memo (ROSAEC MEMO 2009-006) pp. 1–18 (2009)
24. Kaya, M., Bridge, D., Tintarev, N.: Ensuring fairness in group recommendations by rank-sensitive balancing of relevance. In: RecSys '20. p. 101–110. RecSys '20 (2020)
25. Kleinberg, J.: Inherent trade-offs in algorithmic fairness. SIGMETRICS Perform. Eval. Rev. **46**(1), 40 (jun 2018). https://doi.org/10.1145/3292040.3219634, https://doi.org/10.1145/3292040.3219634
26. Li, Y., Meng, L., Chen, L., Yu, L., Wu, D., Zhou, Y., Xu, B.: Training data debugging for the fairness of machine learning software. In: ICSE '2. p. 2215–2227. ICSE '22, Association for Computing Machinery (2022)
27. Li, Y., Chen, H., Fu, Z., Ge, Y., Zhang, Y.: User-oriented fairness in recommendation. The Web Conference 2021 - Proceedings of the World Wide Web Conference, WWW 2021 pp. 624–632 (4 2021). https://doi.org/10.1145/3442381.3449866, https://dl.acm.org/doi/10.1145/3442381.3449866
28. Li, Y., Chen, H., Xu, S., Ge, Y., Zhang, Y.: Towards personalized fairness based on causal notion. In: SIGIR '21. p. 1054–1063. SIGIR '21 (2021)
29. Li, Z., Zhong, H.: Revisiting textual feature of bug-triage approach. In: ASE'21. pp. 1183–1185 (2021)
30. Lipton, Z.C., Chouldechova, A., McAuley, J.: Does mitigating ml's impact disparity require treatment disparity? In: Proceedings of the 32nd International Conference on Neural Information Processing Systems. p. 8136–8146. NIPS'18, Curran Associates Inc., Red Hook, NY, USA (2018)
31. Liu, S., Zheng, Y.: Long-tail session-based recommendation. In: RecSys '20. p. 509–514. RecSys '20 (2020)
32. Liu, W., Guo, J., Sonboli, N., Burke, R., Zhang, S.: Personalized fairness-aware re-ranking for microlending. In: Proceedings of the 13th ACM Conference on Recommender Systems. p. 467–471. RecSys '19, Association for Computing Machinery, New York, NY, USA (2019). https://doi.org/10.1145/3298689.3347016, https://doi.org/10.1145/3298689.3347016

33. Mehrabi, N., Morstatter, F., Saxena, N., Lerman, K., Galstyan, A.: A survey on bias and fairness in machine learning. ACM Computing Surveys (CSUR) **54** (7 2021)
34. Menon, A.K., Williamson, R.C.: The cost of fairness in binary classification. In: Friedler, S.A., Wilson, C. (eds.) Proceedings of the 1st Conference on Fairness, Accountability and Transparency. Proceedings of Machine Learning Research, vol. 81, pp. 107–118. PMLR (23–24 Feb 2018), https://proceedings.mlr.press/v81/menon18a.html
35. Morik, M., Singh, A., Hong, J., Joachims, T.: Controlling fairness and bias in dynamic learning-to-rank. In: SIGIR '20. p. 429–438. SIGIR '20 (2020)
36. Naghiaei, M., Rahmani, H.A., Deldjoo, Y.: Cpfair: Personalized consumer and producer fairness re-ranking for recommender systems. In: Proceedings of the 45th International ACM SIGIR Conference on Research and Development in Information Retrieval. p. 770–779. SIGIR '22, Association for Computing Machinery, New York, NY, USA (2022). https://doi.org/10.1145/3477495.3531959, https://doi.org/10.1145/3477495.3531959
37. Ouni, A., Kula, R.G., Inoue, K.: Search-based peer reviewers recommendation in modern code review. In: ICSME'16. pp. 367–377 (2016)
38. Pandya, P., Tiwari, S.: Corms: a github and gerrit based hybrid code reviewer recommendation approach for modern code review. pp. 546–557. Association for Computing Machinery (ACM) (11 2022)
39. Patro, G.K., Biswas, A., Ganguly, N., Gummadi, K.P., Chakraborty, A.: Fairrec: Two-sided fairness for personalized recommendations in two-sided platforms. In: Proceedings of The Web Conference 2020. p. 1194–1204. WWW '20, Association for Computing Machinery, New York, NY, USA (2020). https://doi.org/10.1145/3366423.3380196, https://doi.org/10.1145/3366423.3380196
40. Pessach, D., Shmueli, E.: A review on fairness in machine learning. ACM Computing Surveys **55**, 1–44 (4 2023). https://doi.org/10.1145/3494672
41. Rastegarpanah, B., Gummadi, K.P., Crovella, M.: Fighting fire with fire: Using antidote data to improve polarization and fairness of recommender systems. In: WSDM '19. p. 231–239. WSDM '19 (2019)
42. Soremekun, E., Papadakis, M., Cordy, M., Traon, Y.L.: Software fairness: An analysis and survey (5 2022)
43. Strand, A., Gunnarson, M., Britto, R., Usman, M.: Using a context-aware approach to recommend code reviewers: Findings from an industrial case study. In: ICSE-SEIP '20. p. 1–10. ICSE-SEIP '20 (2020)
44. Thongtanunam, P., Tantithamthavorn, C., Kula, R.G., Yoshida, N., Iida, H., Matsumoto, K.I.: Who should review my code? a file location-based code-reviewer recommendation approach for modern code review. SANER 2015 pp. 141–150 (4 2015)
45. Tizpaz-Niari, S., Kumar, A., Tan, G., Trivedi, A.: Fairness-aware configuration of machine learning libraries. In: ICSE '22. p. 909–920. ICSE '22 (2022)
46. Wang, S., Liu, T., Nam, J., Tan, L.: Deep semantic feature learning for software defect prediction. IEEE Transactions on Software Engineering **46**(12), 1267–1293 (2020)
47. Wang, S., Liu, T., Tan, L.: Automatically learning semantic features for defect prediction. In: ICSE '16. p. 297–308. ICSE '16 (2016)
48. Wang, Y., Ma, W., Zhang, M., Liu, Y., Ma, S.: A survey on the fairness of recommender systems. ACM Transactions on Information Systems **41**, 1–43 (7 2023). https://doi.org/10.1145/3547333, https://doi.org/10.1145/3547333

49. Xia, Z., Sun, H., Jiang, J., Wang, X., Liu, X.: A hybrid approach to code reviewer recommendation with collaborative filtering. In: MSR'17. pp. 24–31 (2017)
50. Yang, Y., Xia, X., Lo, D., Grundy, J.: A survey on deep learning for software engineering. ACM Comput. Surv. **54**(10s) (sep 2022). https://doi.org/10.1145/3505243, https://doi.org/10.1145/3505243
51. Zehlike, M., Bonchi, F., Castillo, C., Hajian, S., Megahed, M., Baeza-Yates, R.: Fa*ir: A fair top-k ranking algorithm. International Conference on Information and Knowledge Management, Proceedings pp. 1569–1578 (11 2017)
52. Zhang, J.M., Harman, M.: 'ignorance and prejudice' in software fairness. Proceedings - International Conference on Software Engineering pp. 1436–1447 (5 2021). https://doi.org/10.1109/ICSE43902.2021.00129
53. Zhang, W.: Efficient bug triage for industrial environments. In: 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME). pp. 727–735 (2020). https://doi.org/10.1109/ICSME46990.2020.00082
54. Zhu, Z., He, Y., Zhao, X., Zhang, Y., Wang, J., Caverlee, J.: Popularity-opportunity bias in collaborative filtering. In: WSDM '21. p. 85–93. WSDM '21, Association for Computing Machinery, New York, NY, USA (2021)