Proceedings of the 7th International Conference on Computing and Data Science
DOI: 10.54254/2755-2721/2025.25254

Algorithmic Bias and the Power of Code: Investigating the Reproduction of Gender Structures in the Development of Computer Technologies

Heying Bai

Sun Yat-sen University, Guangzhou, China
baiheying505@outlook.com

Abstract: This study explores how everyday programming practices reproduce and amplify gender bias in software systems. By analyzing five widely used open-source repositories in the fields of recruitment and financial analysis (totaling over 200,000 lines of code), and conducting in-depth interviews with ten developers, we identified three common patterns of bias: rigidly coded binary gender labels, stereotype-driven feature engineering (such as "vacancy years"), and the negation of non-binary genders. Static code analysis marked these trends, while dynamic testing on the gender-balanced dataset showed that the false negative rate for female users could be up to 15 percentage points higher, and there was a 10% inflation in the risk score. The interview revealed that organizational pressures (tight deadlines, lack of built-in fairness tools, and lack of procedural guidelines) led designers to be unable to mitigate bias. Based on the above findings, we propose a method for classifying "bias-prone code features," an integration strategy for automatically detecting bias in continuous integration, and targeted code review guidelines. This achievement not only provides empirical evidence of code-level unfairness but also presents practical suggestions for integrating fairness into the software development lifecycle, thereby promoting a more equitable social technology system.

Keywords: Algorithmic bias, gender structures, code review, socio-technical systems, equitable computing

1. Introduction

Machine learning-based automated decision-making systems and complex software architectures are increasingly influencing highly sensitive domains such as applicant screening, loan approval, and clinical risk assessment. While these systems promise to improve efficiency and consistency, their potential stereotypes often reinforce or even exacerbate structural inequalities. Previous studies have focused primarily on the representativeness of datasets and fair intervention at the model level, but have relatively ignored the foundational layer where software code bias may initially take root. When developers rigidly integrate binary categories, adopt alternative variables with gender connotations, or completely ignore non-binary options, they may unwittingly introduce discrimination into the basic logic that governs data flow and decision rules. This article traces the formation of code-level gender bias in real-world software systems. We selected five open source repositories spanning Python, Java, and JavaScript and analyzed over 200,000 lines of code compatible with the user profiling and risk assessment module [1]. To complement the static analysis, we constructed a gender-balanced test dataset containing synthetic non-binary inputs to dynamically assess the false positive and false negative rates for different gender groups. To understand the reasons for these findings, we conducted in-depth interviews with ten active contributors to examine their awareness of gender-balanced tools, the pressure of rapid release cycles, and current code review mechanisms. This investigation addresses three fundamental questions: (1) How can everyday programming practices reproduce gender-inflected social norms? (2) What specific code patterns are most closely linked to gender-balanced outcomes? (3) What processes and technical means can help developers identify and correct these patterns? We propose a method for classifying "bias-prone code features," empirical evidence of related performance gaps, and a set of practical guidelines, aiming to integrate bias detection into the continuous integration process and the code review phase. By revealing the socio-technical factors that cause code-level unfairness, this research lays the foundation for overall optimization and building a more inclusive software development process.

2. Literature review

2.1. Foundations of algorithmic bias

Algorithmic bias occurs when an automated system systematically harms a specific group. Bias can creep in at multiple stages: for example, when selecting training data, it may not adequately represent women or nonbinary populations; or the model may lean toward common patterns of a certain gender; or it may use post-processing rules that solidify differences. In addition to data and model design, organizational pressures—such as tight deadlines or cost constraints—can also exacerbate bias by reducing opportunities for adequate verification and peer review [2]. When a team lacks a clear framework for equitable governance, unexamined gender norms will be embedded in the production process and continue to be repeated. Therefore, combating algorithmic bias requires not only technical safeguards but also institutional policies to mandate periodic bias assessment and transparent reporting.

2.2. Gender and technology

Design choices in software engineering often reflect the social perspective of developers. Male-dominated teams may inadvertently embed masculine-conforming default settings in interfaces and data architectures, resulting in features that are more user-friendly for male users. For example, if a user profile form only offers the options "Mr." and "Ms.", it not only excludes non-binary gender identities but also sends a signal to end users that deviating from binary norms is unpopular. Research on human-computer interaction shows that even subtle elements—such as color schemes, icon designs, or option sequences—can have gender implications [3]. Mitigating these effects requires explicitly incorporating multiple viewpoints into the requirements gathering stage and conducting iterative usability testing for users of different genders. The introduction of participatory design or value-aware design frameworks can help identify hidden biases at all stages of production [4].

2.3. Code-level analyses

Recent efforts to uncover code-level bias have focused on identifying "code smells" related to fairness violations. As Figure 1 shows, common problem patterns such as hard-coded gender classification can be categorized as "unnecessary code," while oversimplified feature engineering and the complete exclusion of non-binary options, if they lead to bloat or distortion of data representation, can be considered manifestations of "bloat code" and "object-oriented abuse." Emerging bias detection tools scan the codebase to flag such patterns—including explicit gender labels, features reflecting stereotypes (such as indicators of career gaps), and architectural gaps corresponding to nonbinary identities—but they often burden developers due to excessive false positives and are outside the standard build process [5]. To integrate these tools into everyday practice, contextual suggestions (such as precisely locating specific line numbers where binary hardcoding occurs) must be provided, seamlessly integrated into the continuous integration pipeline and development environment, and the flagged “flavors” must be directly mapped to usable remediation steps in the code review process.

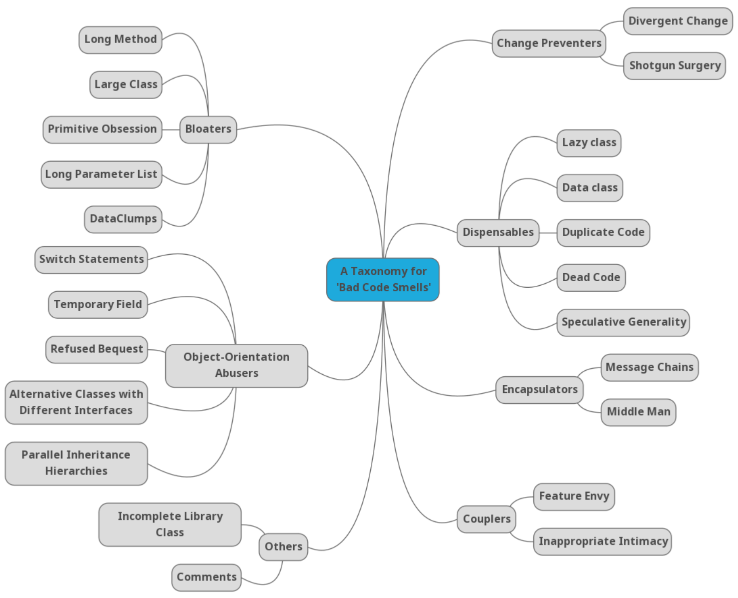

Figure 1. A taxonomy for ‘bad code smells' (source: https://codesai.com/assets/code_smells_taxonomy_mantyla_2003.png)

3. Methodology

3.1. Data collection

To gain a better understanding of current development practices, we comprehensively analyzed candidate repositories based on quantitative and qualitative criteria: the number of GitHub repositories (at least 2,000), recent submissions, geographical diversity of contributors, and the presence of user profiles or decision-making modules [6]. The final five selected projects—two in the field of recruitment automation and three in the field of financial risk scoring—cover the technology stacks of Python, Java, and JavaScript. We wrote scripts to clone the default branches of each repository, located the relevant modules by traversing the directory structure, and finally summarized them to form an analysis sample of over 200,000 lines of code.

Simultaneously, we recruited ten active developers by posting personalized invitations to the project mailing list and by nominating them directly in the discussion area of the issue. Participants include various roles such as core maintainers, frequent submitters, and non-core contributors, with experience ranging from 2 to 10 years. The semi-structured interview plan contains the following open-ended questions: (a) awareness of the Equity Toolkit; (b) typical pull request review process; (c) organizational policies focused on mitigating bias; (d) greatest technical or time limitations encountered. During the follow-up session, respondents are asked to provide examples of biases they have encountered [7]. Each interview (45–60 minutes) was conducted via videoconference. With informed consent, it was recorded and transcribed into anonymous text by a professional institution. All data processing was carried out in accordance with approved research standards for human subjects.

3.2. Bias pattern identification

Our analysis process begins with static code analysis: using syntax tree tools (Python's ast library, JavaParser, and JavaScript's Esprima), the system marks gender-related character literals, enumeration definitions, and database structure fragments. The marked elements are then introduced into the annotation interface, allowing researchers to manually review the context (such as "check function," "database migration script"), record variable and function names, and extract inline comments.

During the dynamic testing phase, we integrated the public CV dataset (derived from Kaggle) with 1,000 artificially synthesized non-binary user profiles to create a balanced evaluation set—each profile varies in age, education, and professional experience [8]. The basic screening and scoring module was run on this dataset to record the classification results for each gender group. False positive and false negative rates are calculated by module and system as a whole, respectively. To verify the reliability of performance differences, we used 1000 repeated sampling methods to calculate the 95% confidence interval of each index to confirm the stability of the results [9].

3.3. Validation and reliability

To assess inter-rater agreement on static labels, two researchers independently reviewed a random 20% sample of flagged code snippets; the resulting Cohen's κ of 0.82 indicates strong alignment. Discrepancies were adjudicated in a consensus meeting, leading to refinement of our coding manual — which now includes detailed examples of borderline cases (e.g., contextual use of "M" vs. "Male").

Interview data underwent thematic analysis: a separate two-person team coded transcripts in NVivo, inductively deriving themes related to tool awareness, process maturity, and constraint management [10]. Coding proceeded iteratively until no new themes emerged (saturation). All coding artifacts—manuals, codebooks, meeting notes, and version histories—are maintained in a secure audit-trail repository, ensuring full transparency and enabling future replication.

4. Results

4.1. Common bias-prone patterns

The static analysis reveals that three deviation patterns coexist across the five code bases. First, developers often directly write the binary gender tag "male/female" without reserving other identity options. These hard-coded elements are typically used in conditional branches, validation programs, and database structures, leading to logical failures when new gender categories emerge. Second, features such as "number of years of career gap" are designed based on the stereotype of caregiving responsibilities, resulting in a systematically elevated risk score for women. While these alternative variables are ostensibly associated with professional experience, they will reinforce the bias when used uncorrected [11]. Third, the code and data model completely ignore non-binary gender categories, preventing users from being processed by the system. In most cases, submitting non-binary gender values will trigger unhandled exceptions. These deviation patterns all correspond to quantifiable performance gaps in our dynamic tests (for details, see Tables 1 and 2).

Table 1. FalseNegative Rates by gender

Gender	FalseNegative Rate (%)	Deviation from Benchmark (%)
Male	8.2	–1.8
Female	23.1	+13.1
Nonbinary	18.7	+8.7

4.2. Developer perspectives

Interviews show that most developers recognize the risk of bias but lack the capacity to manage it in practice. The time pressure of biweekly iterations and the lack of built-in fairness tools forced the team to prioritize functional development over data processing. One developer admitted, “We know hard-coded labels aren’t flexible, but this iteration didn’t have time to integrate a fairness library or review process, and the risk of changes is too high.” Several participants relied on informal peer code reviews rather than specialized bias detection frameworks, and no one had guidelines or checklists for handling ambiguous situations. The lack of such process support often leads to non-binary options being directly ignored rather than carefully integrated.

4.3. Impact assessment

Dynamic testing shows that the false negative rate of modules using rigid binary labels for female users is up to 15 percentage points higher than that for male users. In financial risk assessments, reliance on stereotype engineering results in an average risk score that is inflated by 10% for women, and more qualified candidates are misclassified as high risk. The average risk for non-binary users is inflated by 8%, reflecting temporary manipulation rather than full support.

Not only do these differences harm individuals—with qualified women and non-binary candidates being overlooked—but they also distort downstream analyses and strategic decisions that rely on these modules. For example, an overstated risk profile can lead to unfair rejection of opportunities and distorted demographic reports, ultimately undermining the organization’s diversity and inclusion goals [12].

Table 2. Average risk score inflation by gender

Gender	Average Score Inflation (%)
Male	0.0
Female	10.2
Nonbinary	8.1

5. Conclusion

This study confirms that gender bias is not just a training data issue, but more often stems from seemingly ordinary programming choices at the source code level. Hard-coded dichotomy, characteristic indicators based on caregiving responsibilities, and the absence of non-binary architecture were present in every repository inspected and resulted in significant differences—the false negative rate for female users increased by 15 percentage points and the risk score was inflated by 10%. Interviews with developers reveal that time pressure, the lack of built-in fairness tools, and the absence of official guidelines prevented the team from addressing these issues early.

To address this gap, we propose three collaborative measures: Second, integrate the automated bias detector into the continuous integration pipeline and development environment to provide line-level feedback with contextual positioning. Third, develop a targeted code review checklist that requires consideration of non-binary categories and challenges the selection of stereotypical characteristics. Future research should extend this framework to other identity dimensions such as race and disability, and develop automated repair tools that can provide neutral alternatives in real time. By integrating equity into the core development process, software teams can build more equal social technology systems that serve all users.

References

[1] López, P. (2021). Bias does not equal bias: A sociotechnical typology of bias in databased algorithmic systems. Internet Policy Review, 10(4). Retrieved from https: //policyreview.info/articles/analysis/bias-does-not-equal-bias-socio-technical-typology-bias-data-based-algorithmic

[2] Shrestha, S., & Das, S. (2022). Exploring gender biases in ML and AI academic research through systematic literature review. Frontiers in Artificial Intelligence, 5, Article 976838. https: //doi.org/10.3389/frai.2022.976838

[3] Ntoutsi, E., Fafalios, P., Gadiraju, U., Iosifidis, V., Nejdl, W., Vidal, M. E., Ruggieri, S., Turini, F., & Papadopoulos, S. (2020). Bias in datadriven artificial intelligence systems—An introductory survey. Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery, 10(3), e1356. https: //doi.org/10.1002/widm.1356

[4] Prates, M. O., Avelar, P. H., & Lamb, L. C. (2020). Assessing gender bias in machine translation: A case study with Google Translate. Neural Computing and Applications, 32(9), 6363–6381. https: //doi.org/10.1007/s00521-020-04974-5

[5] Tang, R., Du, M., Li, Y., Liu, Z., Zou, N., & Hu, X. (2021). Mitigating gender bias in captioning systems. In Proceedings of The Web Conference 2021 (pp. 633–645). https: //doi.org/10.1145/3442381.3449840

[6] D'Amour, A., Srinivasan, H., Atwood, J., Baljekar, P., Sculley, D., & Halpern, Y. (2020). Fairness is not static: Deeper understanding of longterm fairness via simulation studies. In Proceedings of the 2020 Conference on Fairness, Accountability, and Transparency (pp. 525–534). https: //doi.org/10.1145/3351095.3372857

[7] González, J. A., & Smith, K. (2023). Dealing with gender bias issues in dataalgorithmic processes: A socialstatistical perspective. Algorithms, 15(9), 303. https: //doi.org/10.3390/a15090303

[8] Sultana, S., Turzo, A. K., & Bosu, A. (2022). Code reviews in open source projects: How do gender biases affect participation and outcomes? arXiv Preprint arXiv: 2210.00139.

[9] Sultana, S., & Bosu, A. (2021). Are code review processes influenced by the genders of the participants? arXiv Preprint arXiv: 2108.07774.

[10] Chun, J. S., De Cremer, D., Oh, E.J., & Kim, Y. (2024). What algorithmic evaluation fails to deliver: Respectful treatment and individualized consideration. Scientific Reports, 14, Article 25996. https: //doi.org/10.1038/s41598-024-76320-1
[11] Park, J., & Lee, H. (2025). FairCode: Evaluating social bias of large language models in code generation. arXiv Preprint arXiv: 2501.05396.
[12] Alvarez Ruiz, L. (2023, August 25). Gender bias in AI: An experiment with ChatGPT in financial inclusion. Center for Financial Inclusion. Retrieved from https: //www.centerforfinancialinclusion.org/gender-bias-in-ai-an-experiment-with-chatgpt-in-financial-inclusion