Fairness Risk Triage in Pull Requests Using an LLM-Based Reviewer: An Experiment on Prompting Strategies

Gabriel Caitano
Federal University of Mato Grosso do Sul (UFMS)
Campo Grande, Brazil
gabriel.caitano@ufms.br

Arthur Cacciatore
Federal University of Mato Grosso do Sul (UFMS)
Campo Grande, Brazil
arthur.cacciatore@ufms.br

Awdren Fontão
Federal University of Mato Grosso do Sul (UFMS)
Campo Grande, Brazil
awdren.fontao@ufms.br

Abstract

While LLM-based code reviewers can identify traditional defects, their ability to triage fairness risks in Pull Requests—serving as specialized skills within broader agentic systems—remains underexplored. This paper evaluates how prompting strategies affect an LLM reviewer's detection and characterization of algorithmic biases. Using Llama 3.3 70B, we conducted a controlled experiment across ten synthetic JavaScript scenarios, each pairing a biased implementation with a clean reference. We compared simple and advanced prompts across 200 executions (five repetitions per combination), evaluated by three LLM judges against ground truth. The simple prompt achieved higher mean recall (0.96 vs. 0.91) and precision (0.84 vs. 0.74). Bootstrap confidence intervals show this difference is not uniformly robust: only one scenario-level effect has non-overlapping intervals, and even the global difference overlaps given the limited number of repetitions. However, the advanced prompt better characterized the specific sensitive attribute and code location of flagged risks. While classifications remained mostly consistent across repetitions, judge-dependent divergences occurred in specific scenarios. Findings highlight a practical trade-off between detection sensitivity and characterization detail.

Keywords

Fairness, Large Language Models, Pull Request Review, Software Engineering

1 Introduction

Pull Request (PR) code review is a critical quality assurance and knowledge-sharing mechanism [3]. However, it remains a predominantly manual, time-consuming activity that often becomes a bottleneck in the delivery pipeline [3].

To mitigate this, Large Language Models (LLMs) are increasingly used to automate code analysis and review [3, 8, 12]. Within Agentic Software Engineering (ASE), these LLM capabilities function as specialized skills orchestrated alongside testing or static analysis [7]. This paper investigates an LLM-based reviewer not as a fully autonomous agent but as a specific, evaluated component, whose standalone capabilities provide evidence for how much autonomy it should be granted within broader agentic or human-in-the-loop pipelines.

Despite advances, automated code review focuses primarily on functional defects and structural issues. There is limited evidence regarding LLMs' ability to identify socio-technical risks, such as algorithmic fairness [2]. Unlike syntax errors, potential bias often depends on application semantics, where seemingly neutral variables cause disparate impacts on sensitive groups [2, 4, 6]. Identifying these issues during PRs is challenging, yet represents a strategic checkpoint against discriminatory logic before integration [2, 9]. It remains unclear how effectively LLMs can flag these risks, how prompting strategies affect their performance, and whether classifications are stable under repetition, factors that define the component's operational role: a reviewer with high sensitivity but unstable behavior may suit triage but not autonomous decisions.

Motivated by this gap, this paper presents a controlled experiment evaluating an LLM-based reviewer's ability to triage fairness risks across ten synthetic JavaScript PR scenarios. Our study offers three main contributions: (i) controlled evidence of an LLM's capability and stability in detecting algorithmic bias; (ii) an analysis of the trade-offs between detection performance, false positives, and characterization detail across simple and advanced prompts; and (iii) preliminary design implications for integrating this capability into agentic workflows. All scenarios, prompts, and evaluation instruments are publicly available to support replication.

2 Related Work

This study bridges two research streams: LLM-assisted code review and software fairness. Regarding code review, recent studies leverage LLMs to automate tasks such as generating comments and identifying bugs, code smells, or architectural issues [8, 12], with industrial deployments showing promising results [3]; these applications, however, primarily target conventional technical defects rather than socio-technical risks embedded in business logic.

Concurrently, fairness has been established as a critical software quality property [2], and existing research shows that LLMs can inadvertently introduce demographic or social biases during code generation [4, 6], though these works focus on the LLM as a generator of biased code, not as a reviewer of it. Our study shifts this paradigm by investigating an LLM as an analytical reviewer — a specialized triage skill — detecting pre-existing fairness risks within PRs, rather than introducing them.

3 Study Design

A controlled experiment evaluated the capability of an LLM-based reviewer to identify fairness risks in simulated Pull Requests. The design crosses ten scenarios, two code conditions (*biased* vs. *clean*), and two prompting strategies. Each of the 40 combinations was executed five times (200 responses total). The five repetitions represent stochastic realizations of the same combination to support stability analysis (RQ3), rather than independent code changes.

The prompting strategy is the main treatment: a simple prompt (testing prior knowledge) versus an advanced prompt (providing

definitions and structure). The code condition enables responses to be classified against a documented ground truth. Model parameters, language, and metadata structure were kept constant.

3.1 Research Questions and Study Objective

The objective of this study is to investigate whether an LLM-based reviewer is capable of identifying fairness risks in code submitted via Pull Requests, and how the prompting strategy affects its behavior during the review. To support this objective, the following research questions (RQs) and metrics were defined:

RQ1: Is the LLM capable of detecting algorithmic biases within a simulated Pull Request setting, and to what extent does the prompting strategy (T1 vs. T2, detailed in Section 3.3) influence this detection, both globally and at the scenario level? This question is divided into two sub-questions: **RQ1.1:** Is the simple prompt sufficient to detect biases relying solely on the model's prior knowledge? and **RQ1.2:** Does the advanced prompt improve detection performance compared to the simple prompt? **Metric 1:** Recall. **Metric 2:** Precision. **Metric 3:** Accuracy.

RQ2: How often does the LLM incorrectly flag bias when reviewing unbiased code within a simulated Pull Request setting, and how well do flagged responses characterize the concern raised? Metric 4: False Positive Rate on the *clean* code. Flagged responses are further characterized via attribute/domain and code-location identification scores (Section 3.5).

RQ3: Does the LLM maintain stability in its responses when repeatedly reviewing the same Pull Request? Metric 5: Classification consistency across the five executions of each experimental combination, assessed separately per judge: a combination was considered stable if all five executions received the same classification, and a scenario was considered unstable if any of its combinations diverged.

3.2 Scenarios and Dataset

To enable a controlled evaluation, ten synthetic JavaScript scenarios were constructed, each pairing a biased implementation (containing a discriminatory rule) with a clean, non-discriminatory alternative of the same functionality and structure. Each scenario pairs a sensitive attribute with a plausible application domain (e.g., an emergency triage mechanism penalizing people with disabilities, or a content moderation feature disparately treating religion); the complete attribute/domain list, code files, and ground truth are available in the study's repository. Attribute selection was informed by Mehrabi et al. [10], Barocas et al. [1], the NIST generative AI risk profile [11], and the Google Fairness Codelab [5], prioritizing domains where similar risks are discussed in the literature; a digital divide scenario was added to capture technology access inequalities not covered by broader socioeconomic categories.

Scenarios were reviewed against a four-dimension checklist: bias classification (mapping to the taxonomy), change control (differences concentrated in the discriminatory rule), explicit clue control (the sensitive attribute is not directly named), and ground truth quality (documented attribute, proxy variable, code location, and evaluation terms). Construction followed a two-stage AI-assisted process with human validation at each stage: the model first produced a structured bias description reviewed against the checklist, then generated the implementation pair and ground truth, which underwent a final correspondence check. This mitigates, though does not eliminate, the risk of misalignment between the planned bias, the code, and the ground truth. Because fairness judgments depend on legal, social, and domain-specific interpretations, the resulting ground truth serves as an operational reference for this study rather than a universally valid determination of fairness, and responses were evaluated strictly against it.

3.3 Reviewer Configuration and Prompting Strategies

The reviewer was implemented using the Llama 3.3 70B model, in the `llama-3.3-70b-versatile` variant, accessed via the Groq API. The temperature was set to 0.2, aiming to limit stochastic variability without producing completely deterministic responses. This configuration also allows the observation of whether, in RQ3, minor variations between executions alter the classification assigned to the responses.

We compared two prompting strategies, treated as T1 and T2. The complete prompts, including the system and user messages used in each request, are available in the replication package.

The **simple prompting strategy (T1)** contains a brief instruction requesting the analysis of potential fairness issues in the code change and the inclusion of a final section titled "Fairness Analysis". The treatment does not provide an operational definition of the concept, a list of sensitive attributes, or a detailed response schema. T1 serves as a baseline to observe the model's behavior relying primarily on its prior knowledge.

The **advanced prompting strategy (T2)** combines an operational definition of fairness, a list of sensitive attributes, questions regarding proxy variables and disparate impact, analysis tasks, and a structured output format. The response must indicate the identified concern, the potentially involved attributes, the location in the code, and a mitigation suggestion.

The comparison between T1 and T2 estimates the joint effect of these two prompting strategies. Because T2 simultaneously modifies content, role, questions, and output format, the design does not allow potential differences to be attributed to an individual prompt component.

3.4 Experimental Pipeline

The experiment was executed by a programmatic pipeline that simulates the subset of the Pull Request setting selected for the study (Figure 1). Instead of creating branches and PRs in a remote repository, the pipeline locally assembles the request sent to the model. This decision reduces operational costs and allows inputs to be controlled, but it does not reproduce all elements of a real-world review setting, such as repository history, test execution, previous comments, and navigation between related files.

Each request contains: (i) standardized title and description metadata; (ii) the source code of the *biased* or *clean* version; and (iii) the instructions corresponding to T1 or T2. The metadata does not inform the model of the condition label, the expected sensitive attribute, or the classification present in the ground truth.

Each execution included a tracking identifier with no semantic meaning, used to differentiate repetitions and mitigate response

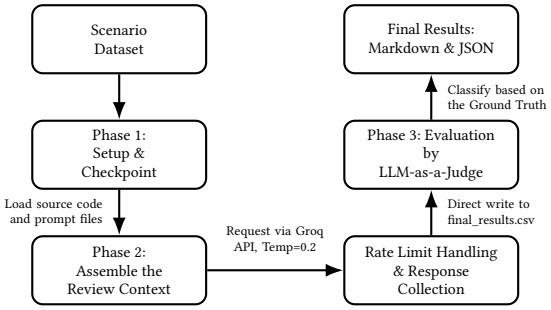

Figure 1: Phases of the experimental pipeline.

reuse by caching mechanisms; it did not encode the scenario, code condition, or expected result. All raw responses, execution parameters, and identifiers were stored prior to the evaluation stage.

3.5 Evaluation Method

The responses produced by the reviewer were evaluated using the *LLM-as-a-Judge* approach. Three distinct models served as judges: Claude Sonnet 4.6, accessed via API, and `deepseek-v4-flash` and `big-pickle`, executed via OpenCode. Complete model identifiers, execution parameters, and evaluation prompts are available in the replication package.

Judge classifications were never combined via majority voting: every metric in this study, for both detection performance and stability, was computed separately per judge and then reported individually or as an arithmetic mean, preserving each judge's assessment and the sensitivity of results to evaluator choice. For each response, the judge received the reviewer's text and the scenario's ground truth, classifying it into one of four categories: True Positive (TP), correctly identifying the risk in a *biased* version; True Negative (TN), correctly withholding a flag in a *clean* version; False Positive (FP), incorrectly flagging a *clean* version; or False Negative (FN), missing the risk in a *biased* version. For RQ1, precision, recall, and accuracy were computed this way per judge; for RQ2, the false positive rate on *clean* versions; and stability (RQ3) was likewise assessed separately per judge, reporting divergence patterns rather than a single aggregated score.

For responses containing a fairness flag (TP or FP), judges additionally rated three qualitative dimensions, each as Yes, Partial, or No: correctness of the identified attribute or domain, accuracy of the identified code location, and format adherence. TN and FN responses had no flag to evaluate and were marked N/A.

Yes, Partial, and No were weighted 1.0, 0.5, and 0.0, respectively. For each judge and prompting strategy, the percentage of a dimension is the mean of these weights across that judge's positive responses; the values reported in Section 4 are the mean of these percentages across the three judges.

To assess whether differences between prompting strategies are robust given the limited number of repetitions, we additionally computed 95% confidence intervals via a block bootstrap (5,000 resamples), resampling executions with replacement and preserving the three-judge classification for each resampled execution, at both the scenario level and pooled across all ten scenarios. No human adjudication was performed on the 200 responses; classifications should therefore be interpreted as contingent on the three LLM judges employed, not as substitutes for human judgment.

4 Results

4.1 RQ1: Fairness Risk Detection and the Effect of the Prompting Strategy

Table 1 summarizes the average metrics from the three judges by prompting strategy, providing the quantitative basis to answer RQ1.

Table 1: Detection metrics and false positive rate by prompting strategy (average of the three judges).

Metric	Simple Prompting Strategy	Advanced Prompting Strategy
Accuracy	0.88	0.80
Precision	0.84	0.74
Recall	0.96	0.91
False Positive Rate (FPR)	0.19	0.31

RQ1.1 — Is the simple prompting strategy sufficient to detect fairness risks? Under the evaluated conditions, the simple prompting strategy obtained a mean judge-assessed recall of 0.96, precision of 0.84, and accuracy of 0.88, indicating that the reviewer flagged most documented fairness risks without an explicit fairness definition or taxonomy. Because no a priori sufficiency threshold was established, this should be read as high descriptive performance within this dataset, not evidence that the simple prompt is generally sufficient.

RQ1.2 — Does the advanced prompting strategy improve detection? Under the evaluated conditions, the advanced prompting strategy produced lower mean detection metrics than the simple strategy: recall decreased from 0.96 to 0.91 and precision from 0.84 to 0.74. This aggregate decrease is not uniform across scenarios (Table 2). Seven of the ten scenarios show identical or near-identical recall under both strategies, while ID-04 (disability) and ID-05 (religion) show substantial drops under the advanced prompt (0.87→0.53 and 1.00→0.60, respectively). ID-02 (gender) shows the opposite pattern (0.73→1.00): the reviewer attributed the injected rule to age/experience under the simple prompt, but the advanced prompt's taxonomy correctly reframed it as gender-related, suggesting the taxonomy helps surface attributes that are indirect and otherwise hard to name from prior knowledge. In ID-04 and ID-05, by contrast, misses under the advanced prompt were not explained by a single consistent alternative reading; responses either raised no concern or attributed the issue to a distinct, plausible but incorrect narrative. These two scenarios are also the primary loci of the RQ3 judge-level instability, suggesting a shared source: scenarios with multiple competing bias narratives appear more likely to push the advanced prompt's structured format toward an incorrect or absent classification. A bootstrap analysis (Section 3.5) confirms only the ID-02 difference is statistically robust (95% CI [0.667–0.867] for T1 vs. [1.000–1.000] for T2); the ID-04/ID-05 drops and the global recall difference ([0.927–0.987] vs. [0.820–0.973]) have overlapping intervals given the small per-cell sample size. Because the analysis is descriptive and scenario-conditioned, these differences should not be interpreted as population-level effects.

Practical implication. The high recall observed with the simple prompting strategy suggests that this configuration may be useful as a high-sensitivity risk sensor within a human-in-the-loop workflow, flagging potentially problematic Pull Requests for human review rather than approving or rejecting changes autonomously. The lower aggregate recall under the advanced strategy does not reflect a uniform cost, but the scenario-specific trade-off described above: a net gain when the discriminatory mechanism requires external naming, and a net cost when competing bias narratives push the structured format toward a premature, incorrect commitment.

4.2 RQ2: False Positives and Characterization of Flagged Responses

RQ2 investigates how often the reviewer flags fairness risks in code classified as clean according to the study's documented ground truth. We additionally examine how the prompting strategies characterize the responses in which a fairness concern was raised.

False positives and potential review effort. For the clean reference versions, the mean False Positive Rate increased from 0.19 with the simple strategy to 0.31 with the advanced strategy, mirrored by the decrease in mean precision from 0.84 to 0.74. This increase is concentrated in a small subset of scenarios rather than distributed evenly. Scenario ID-07 (socioeconomic class) is the primary driver: its false positive count rose from 3 to 14 out of 15 clean executions across the three judges, accounting for most of the total increase between strategies. Scenario ID-05 (religion) shows a related but distinct pattern: its clean version was flagged as a false positive in nearly all judge assessments under both strategies alike (FPR ≈ 1.00), indicating a persistent, strategy-independent tendency to treat its cultural inclusion structures as discriminatory, rather than a behavior induced by the advanced prompt specifically. A bootstrap analysis confirms ID-07 is the only scenario-level effect with non-overlapping confidence intervals ([0.000–0.600] vs. [0.800–1.000]); the global increase, despite the substantial point-estimate gap, has overlapping intervals ([0.100–0.293] vs. [0.193–0.447]). In production, this would likely require additional human review effort; because fairness judgments are context-dependent, both cases also illustrate the evaluation's sensitivity to the study's operational ground truth.

Characterization of flagged responses. Among responses that raised a fairness concern (both true and false positives), the mean attribute/domain identification score rose from 73% to 83% under the advanced strategy, and the mean code-location score from 85% to 94%; format adherence was 100% for both. This indicates that the advanced strategy produced more specific characterizations among flagged responses, though because the analysis includes false positives, it should not be read as evidence of higher diagnostic accuracy over actual fairness risks.

Answer to RQ2 and interpretation. Under the evaluated conditions, the advanced strategy produced a higher false positive rate, but its flagged responses received higher mean characterization scores for attribute/domain and code location, suggesting a trade-off between detection performance and characterization detail rather than one strategy being universally preferable.

Practical implication. The observed trade-off motivates a two-stage design hypothesis, in which a simple prompt performs initial triage and an advanced prompt characterizes flagged cases. Evaluating this requires a dedicated design pairing T1 and T2 executions at the response level and quantifying the resulting human review effort, which falls outside this study's descriptive, per-strategy scope. We therefore report this as a design implication rather than an evaluated result, and identify its validation as a well-defined direction for future work.

Table 2: Recall by scenario and prompting strategy (aggregated across the three judges, 15 executions per cell).

ID	Sensitive Attribute	Simple (T1)	Advanced (T2)
ID-01	Race/ethnicity	1.00	1.00
ID-02	Gender	0.73	1.00
ID-03	Age	1.00	1.00
ID-04	Disability	0.87	0.53
ID-05	Religion	1.00	0.60
ID-06	National origin	1.00	1.00
ID-07	Socioeconomic class	1.00	1.00
ID-08	Dialect/language	1.00	0.93
ID-09	Sexual orientation	1.00	1.00
ID-10	Digital divide	1.00	1.00

4.3 RQ3: Reviewer Stability Under Repetition

RQ3 investigates whether repeated executions of the same experimental combination received consistent classifications, assessed separately for each LLM judge model using the conservative criterion defined in Section 3.1. Across the judge-specific assessments, classification consistency was the dominant pattern.

Where instability occurred. Instability appeared in two distinct forms, affecting different code conditions (Table 3). On the biased side, instability means the reviewer alternates between correctly detecting and missing the injected risk; on the clean side, it means alternating between withholding a flag and raising a false alarm. These forms are not interchangeable: the former affects the reliability of detecting an actual risk, the latter the consistency of the false-positive workload imposed on human reviewers. Divergence agreed upon by all three judges was concentrated in ID-04 and ID-05 under the advanced prompt (biased side), and in ID-02, ID-03, ID-07, and ID-08 on the clean side, as detailed in Table 3. Beyond these unanimous cases, single-judge divergences followed a judge-dependent pattern: Claude Sonnet 4.6 accounted for all additional biased-side cases, while deepseek-v4-flash accounted for most additional clean-side cases, confirming that this sensitivity is not attributable to a single judge across both code conditions.

We did not identify a scenario design characteristic that reliably separates stable from unstable scenarios. All ten scenarios were constructed to avoid explicitly naming the sensitive attribute (Section 3.2), yet four (ID-01, ID-06, ID-09, ID-10) showed no instability under any judge, prompt, or code condition, while the remaining six showed at least one unanimous divergence. Because each attribute and bias mechanism is represented by a single scenario, what distinguishes these two groups remains an open question rather than a conclusion this study can support.

Table 3: Number of judges (out of three) reporting unstable classifications per scenario, code condition, and prompting strategy.

		Biased		Clean	
ID	Attribute	T1	T2	T1	T2
ID-01	Race/ethnicity	0	0	0	0
ID-02	Gender	1	0	1	3
ID-03	Age	0	0	1	3
ID-04	Disability	1	3	1	1
ID-05	Religion	0	3	0	0
ID-06	National origin	0	0	0	0
ID-07	Socioeconomic class	0	0	3	1
ID-08	Dialect/language	0	1	3	3
ID-09	Sexual orientation	0	0	0	0
ID-10	Digital divide	0	0	0	0

Answer to RQ3. Classification consistency remained the dominant pattern, though recurrent divergence occurred in a subset of scenario-prompt combinations across both code conditions. Stability was therefore scenario- and judge-conditioned rather than a general property of the reviewer.

Practical implication. The observed variability reinforces the need to retain human responsibility and define escalation rules for uncertain classifications, which may need to differ by code condition: divergence on biased code (ID-04, ID-05) affects whether a genuine risk is reliably caught, while divergence on clean code (ID-02, ID-03, ID-07, ID-08) affects how consistently reviewers must adjudicate false alarms that were never real. Repeated execution followed by human escalation represents one possible mitigation for both cases, though its effectiveness and operational cost were not evaluated here.

5 Study Design Trade-offs and Validity Implications

Following the trade-off perspective proposed by Robillard et al. [13], we discuss the main design decisions, their alternatives, and their implications for interpreting the findings. **Experimental control versus ecological realism.** We used ten paired synthetic scenarios and a programmatically assembled Pull Request context instead of real repository changes, isolating the discriminatory rule while preserving ground truth. However, the simulated setting excludes repository history, tests, prior comments, related files, and developer interaction. The balanced number of biased and clean versions also differs from production settings, where the low base rate of fairness risks would reduce the positive predictive value; reported precision and alert workload should not be read as production estimates. Each request also included standardized title/description metadata alongside the code, so the design cannot isolate whether detection relies on code logic, metadata cues, or both; an ablation removing the metadata would clarify this.

Component-level evaluation versus full agentic system integration. We evaluated the reviewer as a standalone analytical component rather than as an integrated capability within an orchestrating agentic pipeline, isolating its detection behavior from confounds introduced by tool coordination or multi-step decision-making. This component-level evidence is a precondition for the higher-level design decision of how much autonomy to grant this capability within an agentic or human-in-the-loop workflow; evaluating it within a fully integrated pipeline remains future work.

Focused comparison versus broader generalization. We fixed the reviewer model, temperature, language, metadata, and pipeline to isolate the effect of prompting strategy; a broader design spanning multiple models, languages, and repositories would make attribution more difficult. The findings therefore apply only to the evaluated Llama 3.3 70B configuration and JavaScript scenarios, not to other models, commercial reviewers, or programming ecosystems.

Complete prompting strategies versus component-level attribution. The advanced prompt combines a fairness definition, sensitive-attribute taxonomy, guiding questions, and structured output. We compared complete configurations, representing deployable review strategies, rather than running an ablation to isolate each component's contribution. The results therefore estimate the joint effect of the advanced strategy and cannot attribute differences to any individual prompt element.

Evaluation coverage versus human-grounded judgment. Three LLM judge models separately evaluated all 200 responses, providing complete coverage but no human adjudication. Reporting judge-specific metrics and their mean avoids treating one model as definitive, though it may smooth meaningful disagreements; classifications should be interpreted as contingent on the selected judges, not as substitutes for expert assessment.

Repeatability evidence versus inferential strength. Five executions were collected per experimental combination to observe stochastic variation; these repetitions are realizations of the same scenario, not independent code changes. Stability was assessed conservatively, treating any divergence among the five classifications as instability, a criterion that does not distinguish isolated disagreement from broader dispersion. The bootstrap analysis in Sections 4.1 and 4.2 quantifies the resulting uncertainty; differences between prompting strategies should be interpreted as descriptive, scenario-conditioned trends rather than statistically established population effects.

6 Conclusion and Future Work

This paper presented a controlled experimental evaluation of an LLM-based reviewer for fairness risk triage in Pull Requests, covering ten synthetic scenarios, two prompting strategies, and 200 executions assessed by three LLM judge models. For RQ1, the simple strategy achieved a mean recall of 0.96 without an explicit fairness definition or taxonomy; the advanced strategy produced lower detection metrics but higher characterization scores among flagged responses. For RQ2, mean False Positive Rate increased from 0.19 to 0.31. Bootstrap confidence intervals indicate that these aggregate differences are driven by a small subset of scenarios rather than a uniform effect. For RQ3, classification consistency was the dominant pattern, though recurrent divergence occurred in specific scenario-prompt combinations and varied across judge models.

Taken together, the results provide initial controlled evidence for LLM-based reviewers as risk sensors within human-in-the-loop

workflows, indicating that prompt configurations should be assessed by false positives and stability, not detection metrics alone. The findings do not support autonomous acceptance or rejection of code changes. Future work should evaluate real Pull Requests, expand the number and diversity of independent scenarios, cover additional model families and programming languages, quantify uncertainty at the scenario level, manually annotate a subset of responses to assess agreement between human and LLM judges, conduct a component-level ablation of the advanced prompt to isolate the contribution of its individual elements (fairness definition, taxonomy, guiding questions, and structured output), and experimentally evaluate the proposed two-stage review design.

Artifact Availability

All artifacts supporting this study are publicly available at https://github.com/gfernandes04/artefatos-ic.git, including the ten synthetic scenarios (*biased* and *clean* versions with their respective ground truth documentation), the two prompt instruction sets (simple and advanced), the execution pipeline, and the evaluation prompts for the LLM-as-a-Judge.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior (CAPES), Brasil, Finance Code 001, by the Fundação de Apoio ao Desenvolvimento do Ensino, Ciência e Tecnologia do Estado de Mato Grosso do Sul (FUNDECT, Call No. 42/2024), and by the Federal University of Mato Grosso do Sul (UFMS).

The authors declare that Artificial Intelligence tools based on Large Language Models (LLMs) were used strictly as writing assistants for English translation, grammatical refinement, and LaTeX space optimization. All study conception, experimental design, pipeline execution, data analysis, and scientific conclusions contained in this paper were developed entirely by the human authors, who assume full responsibility for the published content.

References

[1] Solon Barocas, Moritz Hardt, and Arvind Narayanan. 2023. *Fairness and Machine Learning: Limitations and Opportunities.* MIT Press, Cambridge, MA.

[2] Yuriy Brun and Alexandra Meliou. 2018. Software fairness. 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) *(ESEC/FSE 2018)*. Association for Computing Machinery, New York, NY, USA, 754–759. doi:10.1145/3236024.3264838

[3] Umut Cihan, Vahid Haratian, Arda İçöz, Mert Kaan Gül, Ömercan Devran, Emircan Furkan Bayendur, Baykal Mehmet Uçar, and Eray Tüzün. 2025. Automated Code Review in Practice. In *2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 425–436. doi:10.1109/ICSE-SEIP66354.2025.00043

[4] Yongkang Du, Jen tse Huang, Jieyu Zhao, and Lu Lin. 2025. FairCoder: Evaluating Social Bias of LLMs in Code Generation. arXiv:2501.05396 [cs.CL] https://arxiv.org/abs/2501.05396

[5] Google. 2022. Fairness: Types of Bias – Machine Learning Crash Course. https://developers.google.com/machine-learning/crash-course/fairness. Accessed: 2026-07-06.

[6] Dong Huang, Jie M. Zhang, Qingwen Bu, Xiaofei Xie, Junjie Chen, and Heming Cui. 2025. Bias Testing and Mitigation in LLM-based Code Generation. *ACM Trans. Softw. Eng. Methodol.* 35, 1, Article 5 (dec 2025), 31 pages. doi:10.1145/3724117

[7] Haolin Jin, Linghan Huang, Haipeng Cai, Jun Yan, Bo Li, and Huaming Chen. 2025. From LLMs to LLM-based Agents for Software Engineering: A Survey of Current, Challenges and Future. arXiv:2408.02479 [cs.SE] https://arxiv.org/abs/2408.02479

[8] Junyi Lu, Lei Yu, Xiaojia Li, Li Yang, and Chun Zuo. 2023. LLaMA-Reviewer: Advancing Code Review Automation with Large Language Models through Parameter-Efficient Fine-Tuning. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. 647–658. doi:10.1109/ISSRE59848.2023.00026

[9] Qinghua Lu, Liming Zhu, Xiwei Xu, Jon Whittle, and Zhenchang Xing. 2022. Towards a roadmap on software engineering for responsible AI. In *Proceedings of the 1st International Conference on AI Engineering: Software Engineering for AI* (Pittsburgh, Pennsylvania) *(CAIN '22)*. Association for Computing Machinery, New York, NY, USA, 101–112. doi:10.1145/3522664.3528607

[10] Ninareh Mehrabi, Fred Morstatter, Nripsuta Saxena, Kristina Lerman, and Aram Galstyan. 2021. A Survey on Bias and Fairness in Machine Learning. *Comput. Surveys* 54, 6, Article 115 (2021). doi:10.1145/3457607

[11] National Institute of Standards and Technology. 2024. *Artificial Intelligence Risk Management Framework: Generative Artificial Intelligence Profile.* Technical Report NIST AI 600-1. U.S. Department of Commerce. doi:10.6028/NIST.AI.600-1

[12] Zeeshan Rasheed, Malik Abdul Sami, Muhammad Waseem, Kai-Kristian Kemell, Xiaofeng Wang, Anh Nguyen, Kari Systä, and Pekka Abrahamsson. 2025. AI-powered Code Review with LLMs: Early Results. arXiv:2404.18496 [cs.SE] https://arxiv.org/abs/2404.18496

[13] Martin P Robillard, Deeksha M Arya, Neil A Ernst, Jin LC Guo, Maxime Lamothe, Mathieu Nassif, Nicole Novielli, Alexander Serebrenik, Igor Steinmacher, and Klaas-Jan Stol. 2024. Communicating study design trade-offs in software engineering. *ACM Transactions on Software Engineering and Methodology* 33, 5 (2024), 1–10.

A Taxonomy of Sensitive Attributes

Table 4 presents the 10 sensitive attributes covered by the synthetic scenarios, their associated application domains, and the references supporting each attribute selection (Section 3.2).

Table 4: Taxonomy of Sensitive Attributes and Application Domains

ID	Attribute	Domain	Ref.
ID-01	Race/ethnicity	Criminal justice	[10]
ID-02	Gender	Recruitment	[10]
ID-03	Age	Credit approval	[1]
ID-04	Disability	Emergency triage	[10]
ID-05	Religion	Content moderation	[5]
ID-06	National origin	Logistics and delivery	[1]
ID-07	Socioeconomic class	Credit and tax identification	[5, 10]
ID-08	Dialect/language	Sentiment analysis	[11]
ID-09	Sexual orientation	Content recommendation	[1]
ID-10	Digital divide	Engagement scoring	This work