High Agreement, Shallow Reasoning: A Mixed-Method Study of LLMs in Refactoring Reviews

Larisse Amorim
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
larisse.amorim@dcc.ufmg.br

Caique Fortunato
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
caiquefortunato@dcc.ufmg.br

Gustavo Vale
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
gustavovale@dcc.ufmg.br

Eduardo Figueiredo
Federal University of Minas Gerais
Belo Horizonte, Minas Gerais, Brazil
figueiredo@dcc.ufmg.br

Abstract

Evaluating refactoring changes, which lack observable behavioral differences but require judgment on readability, maintainability, and design, is subjective. This study assesses whether large language models (LLMs) can match human decisions in refactoring reviews. By analyzing 146 refactoring code pairs with *Extract Method* and *Rename Variable* from Java projects, we prompted ChatGPT 5-mini and LLaMA 4 Scout to choose between original and refactored code. ChatGPT 5-mini aligned with human judgments in 91.8% of the time, and LLaMA 4 Scout in 82.9%, a statistically significant difference and substantial inter-model agreement. Qualitative analysis of disagreement cases stems from missing context, semantic overgeneralization, and reliance on surface-level stylistic heuristics. These findings suggest that LLMs can often reproduce low-context human preferences but lack the deep reasoning needed for nuanced design evaluation, highlighting both their promise and limitations as refactoring reviewers.

CCS Concepts

• **Software and its engineering** → *Software maintenance tools.*

Keywords

Code Review, Large Language Models, Refactoring Evaluation, Empirical Study, Human-AI Alignment, Software Quality Assessment

ACM Reference Format:
Larisse Amorim, Caique Fortunato, Gustavo Vale, and Eduardo Figueiredo. 2026. High Agreement, Shallow Reasoning: A Mixed-Method Study of LLMs in Refactoring Reviews. In *22nd International Conference on Predictive Models and Data Analytics in Software Engineering (PROMISE'26), July 5, 2026, Montreal, QC, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3803846.3807463

1 Introduction

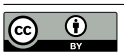
Refactoring is a change in source code to enhance internal quality attributes, such as readability, structure, and maintainability [15]. Refactoring intentions can range from structural improvement to facilitating new features or fixing defects [4, 34]. Due to its subjective nature and inherent trade-offs, manually reviewing a refactored code is time-consuming, requiring syntactic and contextual understanding. Code review is a vital software engineering practice for early defect detection, enforcing standards, and promoting shared understanding of the codebase [6]. Numerous studies [5, 6, 26] confirm that systematic code reviews lead to improved code quality and fewer bugs. However, developers spend over six hours weekly on code reviews [9], a significant project overhead.

Automated tools have been proposed to support code reviews [5]. For instance, Mukadam et al. [27] proposed an open-source Web-based code review tool commonly used in larger Git-based projects. More recently, preliminary studies [49] have investigated the use of Large Language Models (LLMs), such as GPT, Copilot, and Llama, to assist or automate parts of the code review process. Although LLMs have shown promise in software development tasks [29], such as code generation and refinement, we still lack a deep understanding of their ability to judge code quality as human reviewers.

To address this issue, we present an empirical study investigating whether LLMs can effectively evaluate refactoring code reviews by comparing their assessments with human reviewers. We focus on refactoring changes because they maintain program behavior, driving reviewers to judge solely on internal quality attributes, such as readability and maintainability. Refactoring uniquely suited for studying nuanced, subjective judgment, offering a stringent test of whether LLMs genuinely align with human subjective reasoning beyond superficial signals.

In this study, we focus on two widely adopted refactoring types: `Extract Method` and `Rename Variable`. These refactorings are frequent in practice and directly impact core quality attributes evaluated during code reviews. `Extract Method` requires reasoning about abstraction and structural organization, while `Rename Variable` demands semantic interpretation of naming adequacy.

Our empirical analysis relies on the publicly available MaRV dataset [30] containing 693 code pairs extracted from 126 Java repositories on GitHub. From this dataset, we selected 321 code pairs of commits (46.32%) and filtered the refactoring types `Extract Method` and `Rename Variable`, ending up with 146 code pairs to evaluate. Each pair was submitted to two representative LLMs – ChatGPT 5-mini and LLaMA 4 Scout – using a standardized prompt that asked the models to decide whether the original code



PROMISE'26, July 5, 2026, Montreal, QC, Canada

ACM ISBN 979-8-4007-2584-5/26/07
https://doi.org/10.1145/3803846.3807463

(code 1) or the refactored version (code 2) was superior, based on three quality criteria: readability, naming and structure, and maintainability. We then compared the model output with the decisions of actual human reviewers recorded in the MaRV dataset. In addition to quantitative analyses (i.e., effectiveness and agreement), we also conducted a manual qualitative investigation to identify and classify the main reasons for divergence between human and model judgments.

Our results show that LLMs often agree with human reviewers when assessing refactorings: ChatGPT 5-mini agreed with human evaluations in 91.8% of cases, and LLaMA 4 Scout in 82.9%. However, important discrepancies remain. Our qualitative analysis revealed that ChatGPT 5-mini tends to struggle with semantic interpretation, while LLaMA 4 Scout often fails to recognize subtle improvements in code quality. Both models exhibited fragility in reasoning about maintainability, particularly in context-dependent scenarios. These findings underscore the potential of LLMs to support code review activities while highlighting the importance of contextual and architectural information for deeper reasoning.

In summary, this study makes the following contributions.

- We conduct a comparative empirical study evaluating ChatGPT 5-mini and LLaMA 4 Scout on the task of judging code refactorings using real-world commit data.
- We assess the level of agreement between LLMs and human reviewers, revealing the strengths and weaknesses of each model in emulating human evaluative judgment.
- We identify and classify the primary sources of disagreement between LLMs and human evaluators, providing empirical evidence of the contextual and structural challenges.
- We discuss implications for improving prompt design and model performance in code review tasks, highlighting future directions for research and practical deployment.
- We make our data, experimental protocol, and analysis scripts publicly available for verification and replication [3].

The remainder of this paper is organized as follows. Section 2 presents an overview of the related work. Section 3 details our study design and its protocols. Section 4 presents our results. Section 5 discusses our main findings. Section 6 outlines some threats to the study validity and our actions to mitigate them. Finally, Section 7 concludes the paper and points out directions for future work.

2 Background and Related Work

This work relates to three research streams: (i) automation of code review and refinement, (ii) LLM-based assessment of internal code quality attributes, and (iii) alignment between LLM judgments and human evaluation.

Automating Code Review and Refinement. Prior work has explored learning-based approaches to automate review activities, including replicating reviewer edits [44] and supporting multiple tasks such as change recommendation and comment generation through pre-trained models like *CodeReviewer* [23]. More recent studies investigate the use of LLMs for code refinement and review, highlighting their ability to reproduce common improvements as well as their sensitivity to prompting and context [18, 35, 54]. However, these approaches primarily focus on automating review actions or generating feedback, rather than assessing alignment with human review decisions.

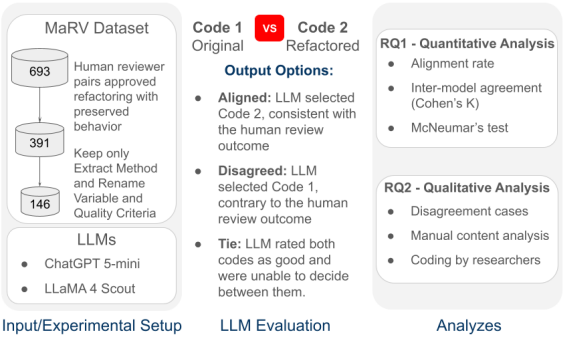


Figure 1: Study Design Overview

LLMs for Code Quality Assessment. Another line of work examines whether LLMs can evaluate internal code quality attributes such as readability and maintainability [2]. Existing studies report moderate to strong correlation between LLM predictions and human judgments across multiple criteria [32, 39]. However, they typically consider isolated quality dimensions and do not capture the comparative and decision-oriented nature of code reviews.

Alignment Between LLMs and Human Reviewers. Recent work has investigated whether LLM evaluations align with human judgments, particularly under the *LLM-as-a-judge* paradigm [48]. While these studies report near-human agreement in selected scenarios, they often rely on synthetic tasks or single-version artifacts, providing limited insight into alignment in realistic review settings.

Despite this progress, it remains unclear whether LLMs can reproduce human review *decisions* when evaluating competing code alternatives under realistic conditions. Refactoring scenarios offer a suitable setting for this investigation, as they preserve behavior while emphasizing subjective internal quality attributes. In this work, we address this gap by analyzing LLM-human judgment alignment in such contexts and examining the factors that drive agreement and disagreement.

3 Study Design

This section presents the design of our study. Section 3.1 introduces the research questions that guided our investigation. Section 3.2 details the configuration and selection of the evaluated models. Section 3.3 describes the dataset. Finally, Section 3.4 explains how we operationalized our analysis to answer each research question. Figure 1 provides an overview of our study.

3.1 Goal and Research Questions

The overarching goal of this study is **to assess whether and how LLMs can effectively evaluate refactoring commits in alignment with human reviewers**. To achieve this goal, we define two complementary research questions (RQs) that address both the *extent* and the *nature* of the alignment between humans and LLMs.

RQ1: *To what extent do LLMs agree with human reviewers when evaluating code refactorings?*

RQ2: *Which factors explain the divergences between LLM and human evaluations?*

RQ1 aims to quantify the level of agreement between LLM assessments and human review outcomes. If models consistently converge with human reviewers, particularly in non-trivial refactoring scenarios, they may serve as trustworthy tools for reducing review effort and promoting greater consistency in software quality assessment. While RQ1 focuses on measuring the strength of agreement, RQ2 investigates the causes of disagreement. Understanding these divergence components provides actionable insights for improving model design, prompt engineering, and integration strategies in practical code review workflows. Together, these research questions enable for a mixed-method investigation.

3.2 LLMs Configuration and Selection

Model Selection. To ensure a fair and representative evaluation of LLMs in code review tasks, we selected two complementary models: ChatGPT 5-mini and LLaMA 4 Scout. We selected these models for their contrasting characteristics and representativeness of the broader ecosystem. ChatGPT 5-mini exemplifies a proprietary, instruction-tuned model optimized for alignment and safety through large-scale human feedback, whereas LLaMA 4 Scout embodies an open, academically accessible model emphasizing interpretability and experimentation. Their comparison allows us to examine whether alignment benefits translate into more human-like judgments in code review scenarios. This contrast also offers insight into how commercial alignment and open research optimization affect evaluative consistency.

Evaluation Settings. We independently queried both models under identical experimental conditions, using the same standardized prompt described as follows. For models in the GPT-5 series (including GPT-5 Mini), classical parameters such as *temperature, frequency_penalty, top_p, presence_penalty, logit_bias, logprobs*, and *even max_tokens* are not supported or applicable. Consequently, it is not possible to define or override these values directly through the API. Model behavior is instead controlled internally by other mechanisms (e.g., *reasoning_effort, verbosity*, among others), or such parameters are simply not exposed to the user. In contrast, the LLaMA Scout model allows explicit parameterization. However, in this work, neither temperature nor *top_p* values were specified. In such cases, the platform typically adopts default settings, commonly around 0.6–0.7 for temperature or 1.0 for *top_p*. All evaluations were executed through the respective public APIs to ensure reproducibility and to reflect realistic usage. We collected and stored the responses in a controlled environment to prevent cross-contamination between the model outputs.

3.3 Dataset and Sampling

Our empirical evaluation is grounded in the *Manually Validated Refactoring Dataset (MaRV)* [30], a curated benchmark comprising 693 manually reviewed code pairs extracted from 126 Java repositories hosted on GitHub. Each pair corresponds to a single commit and includes both the pre-refactoring (original) and post-refactoring versions of a method, and was independently validated by experienced software developers to ensure correctness and behavioral equivalence. This dataset is particularly suitable for our study because it captures real-world refactoring activities accompanied by verified human judgments, providing a robust foundation for evaluating model alignment in code review contexts. MaRV encompasses four refactoring types: *Rename Method*, *Rename Variable*, *Extract Method*, and *Rename Parameter*, offering diversity in both syntactic and semantic transformations.

Filtering and Selection Criteria. To ensure consistency and feasibility for LLM-based evaluation, we applied a five-step filtering process: (i) we retained only cases where both human reviewers agreed on the correctness of the refactoring, yielding 321 verified pairs; (ii) we restricted the dataset to the two refactoring types that preserve semantics while altering structure and identifiers (*Extract Method* and *Rename Variable*), (iii) we excluded partial or truncated snippets lacking complete method bodies, since incomplete code would prevent reliable evaluation by LLMs; (iv) we removed code pairs with parsing or encoding errors to maintain syntatic integrity; and (v) we discarded duplicates and examples exceeding the model input limit (greater than 500 lines of code). After applying these filters, the final sample consisted of **146 valid and analyzable code pairs**, distributed across the two refactoring types (84 *Extract Method* and 62 *Rename Variable*). Two co-authors verified the resulting sample correctness. It maintains project diversity across domains and repository sizes, ensuring representativeness while controlling code length and complexity.

Reproducibility. All filtering scripts, data, and prompt configurations used in this study are included in our replication package [3] to facilitate independent verification and replication.

3.4 Operationalization

We used a sequential explanatory mixed-method design (quantitative then qualitative) to address our research questions.

Quantitative Assessment (RQ1). We evaluated two models on 146 code pairs using a simple prompt that asked which version (original - Code 1 or refactored - Code 2) was superior based on clarity, maintainability, and naming. We found that a more detailed prompt yielded the same results, so we proceeded with the simpler one (presented in Listing 1), simulating a low-context review (e.g. no commit messages). The output was aligned if the model preferred the refactored version; otherwise, it was a disagreement. We measured model alignment using the proportion of cases aligned. Statistical reliability was assessed using *McNemar's test* (to check if models' alignment proportions differed significantly) [25] and *Cohen's* κ (to measure inter-model agreement, correcting for chance) [24]. κ values above 0.60 indicate substantial agreement, and above 0.80 indicate almost perfect agreement [22].

Qualitative Analysis (RQ2). To investigate the reasons behind disagreement cases between LLM outputs and human review decisions, we conducted a qualitative analysis informed by *Grounded Theory* techniques [41, 42]. This qualitative analysis considered all instances in which at least one LLM decision diverged from the human outcome (*Code 1* or *Tie*). Four researchers (three with extensive industry experience) independently examined each instance.

We iteratively applied **open coding** and **axial coding** [12, 42]. During *open coding*, researchers analyzed LLM justifications and code changes, inductively assigning descriptive codes to reasoning strategies and cues, without predefined categories. In the subsequent *axial coding*, the team collaboratively grouped open codes into higher-level categories, identifying causal conditions and reasoning mechanisms underlying disagreement. Disagreements were resolved through discussion. This iterative and comparative process continued until theoretical saturation; i.e., when additional cases no longer yielded new codes or categories. The resulting categories represent divergence mechanisms that explain why LLM judgments depart from human evaluations, grounding our qualitative findings into empirical evidence rather than predefined taxonomies.

```
<prompt>
  <context>
   As a software engineer performing a code review, evaluate
        code_1 and code_2 indicating which version is
       better in terms of clarity, readability, and
       maintainability, and provide a concise justification.

  </context>
  <task>
    Can you rate whether code_1 is better than code_2?
    I would like to know:
      1) Which code is easier to understand.
      2) Consider changes in naming, structure, or style.
      3) Which code is easier to maintain and extend.
  </task>
  <output_format>
    "winner": "code_1" | "code_2" | "tie",
    "reasons": [string, ...]
  </output_format>
</prompt>
```

Listing 1: Prompt Template

4 Results

This section presents the results of our empirical evaluation, organized around the two research questions. Section 4.1 reports the level of agreement between LLMs and human reviewers (RQ1). Section 4.2 analyzes the factors behind the differences (RQ2).

4.1 RQ1 – Human and LLM Agreement

Figure 2 summarizes the distribution of decisions made by the two LLMs. ChatGPT 5-mini favored the refactored version (*Code 2*) in 134 of 146 cases (91.8%), whereas LLaMA Scout preferred it in 121 cases (82.9%). This 9% difference indicates that ChatGPT 5-mini was more assertive in identifying refactoring improvements and was more frequently aligned with human reviewers, who had originally rated the refactored versions as superior. Conversely, ChatGPT 5-mini selected the original code (*Code 1*) in 9 cases (6.2%), while LLaMA Scout did so in 5 cases (3.4%). Finally, ChatGPT 5-mini reported 3 ties (2.1%), compared to 20 (13.7%) by LLaMA Scout, suggesting that the latter model was more conservative when the differences between versions were marginal.

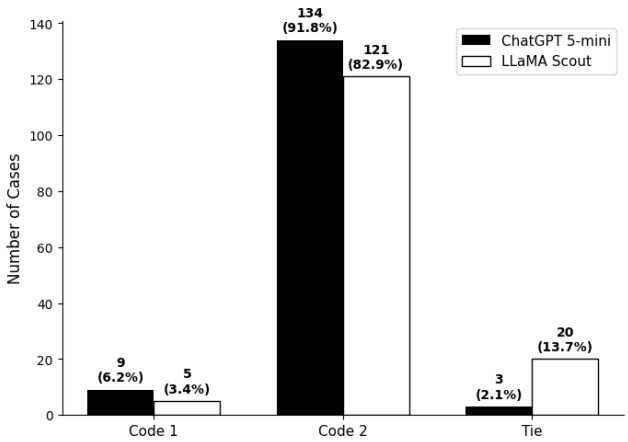


Figure 2: Distribution of Decisions by LLM Models

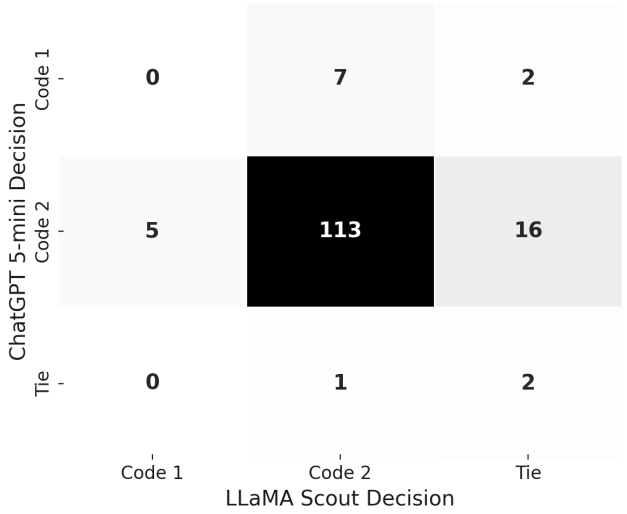


Figure 3: Agreement/Disagreement Matrix

To complement this analysis, Figure 3 shows the *Agreement/Disagreement Matrix* between both models. The diagonal cells represent cases of full agreement. The largest cell (113 cases) indicates that both models agreed that the refactored version was superior, representing 77.4% of the total dataset. Including all diagonal cells (113 Code 2 + 0 Code 1 + 2 ties), the models agreed in 115 of 146 cases, yielding an overall inter-model agreement rate of 78.8%. This substantial overlap demonstrates that both models consistently recognized similar quality improvements after refactoring.

Off-diagonal cells illustrate disagreement patterns. For example, in 7 cases (4.8%), LLaMA Scout preferred *Code 2* while ChatGPT 5-mini favored *Code 1*; in 16 cases (11.0%), ChatGPT 5-mini favored *Code 2* whereas LLaMA Scout reported a tie. These discrepancies occurred primarily in low-impact refactorings involving minor stylistic or structural changes, scenarios where human reviewers also tend to diverge. When ChatGPT 5-mini selected *Code 2*, LLaMA Scout agreed in 84% of cases (113 of 134); in contrast, when LLaMA Scout selected *Code 2*, ChatGPT 5-mini agreed in 93% (113 of 121). This asymmetry suggests that ChatGPT 5-mini

was more decisive, whereas LLaMA Scout exhibited higher uncertainty, possibly reflecting differences in internal calibration and confidence scoring.

To test whether these differences were statistically meaningful, we applied *McNemar's test* to paired binary outcomes (*aligned* vs. *not aligned*), as described in Section 3.4. The resulting $\chi^2 = 5.14$ ($df = 1$, $p < 0.05$) indicates that ChatGPT 5-mini achieved significantly higher aligned with human reviewers than LLaMA Scout. Inter-model consistency, measured using *Cohen's* κ, yielding $\kappa = 0.73$ (95% CI: 0.65-0.81), corresponding to *substantial agreement* according to Landis and Koch [22]. These results collectively demonstrate that while both models exhibit strong alignment with human judgments, the advantage of ChatGPT 5-mini is statistically significant and practically meaningful.

Interpretation. The overall picture suggests that both LLMs captured common refactoring quality heuristics, such as method decomposition and naming clarity, but differed in how confidently they expressed these preferences. ChatGPT 5-mini, with stronger human alignment through RLHF, tended to converge more closely with reviewers' consensus, whereas LLaMA 4 Scout often defaulted to neutral (tie) responses under uncertainty. This observation indicates that alignment training can improve decisiveness and reduce ambiguity in evaluative tasks.

Summary of RQ1. Both LLMs aligned closely with human reviewers in refactoring reviews. ChatGPT 5-mini achieved 91.8% alignment and LLaMA Scout 82.9%, with a statistically significant difference ($p < 0.05$) and substantial inter-model agreement ($\kappa = 0.73$).

4.2 RQ2 – Analysis of Divergent Cases

To answer RQ2, we examined the 33 commits (22.6% of the total sample) in which at least one LLM disagreed with the validated human outcome. In 12 cases only one model diverged, while in 21 both ChatGPT 5-mini and LLaMA Scout differed from the human reviewers. Overall, ChatGPT 5-mini disagreed with humans in 8.3% of all commits and LLaMA Scout in 17.1%. Among the 33 divergent commits, 11 (33.3%) also represented direct disagreements between the two LLMs. Four authors qualitatively analyzed each divergent instance using the open and axial coding techniques. Four distinct **mechanisms of divergence** emerged from this analysis: (i) *contextual blindness* (13 cases), (ii) *semantic overgeneralization* (10 cases), (iii) *heuristic bias toward style and naming* (8 cases), and (iv) *prompt misinterpretation or parsing errors* (2 cases).

Contextual Blindness. Contextual Blindness is the main cause of misalignment. Both models often evaluated extracted methods as self-contained improvements, ignoring the loss of shared state or parameter coupling introduced by refactoring. For example, in commit 50759 (Spring Integration), reviewers noted that GPT's suggestion was flawed because the new method required an external payload. Similarly, in commit 108564 (Terasology), models praised the extracted getter despite its violation of encapsulation by exposing an OpenGL constant. These instances show that LLMs prioritize *local syntactic readability* over global behavioral integrity.

Semantic Overgeneralization. Ten cases showed reasoning errors caused by excessive generalization of code semantics. ChatGPT frequently interpreted relocated or shortened code fragments as deletions, penalizing valid refactorings. For example, in commit 123614 (Graal), ChatGPT labeled the refactoring as "*Bug Free*" and favored the original code, claiming that "*the extracted method may lose functionality*," although the new method simply centralized repetitive checks. An author's note observed that "*GPT confuses method behavior with its declaration.*" Conversely, LLaMA occasionally rewarded simplified structures, asserting superiority "*because the code looks cleaner*," even when behavior was incomplete.

Heuristic Bias Toward Style and Naming. Eight cases revealed that both models equate stylistic neatness with higher quality. They systematically rewarded improved indentation, shorter functions, or naming consistency even when maintainability worsened. In commit 92503 (Cyberduck), LLaMA favored the refactored version "*because the formatting looks cleaner*," while human reviewers rejected it for adding redundancy. Likewise, in commit 108564 (Terasology), ChatGPT preferred the version with improved naming despite duplicated logic. As summarized by one annotation: "*The output values cleaner names but ignore cohesion and encapsulation.*"

Prompt Misinterpretation and Parsing Errors. Two cases originated from incorrect prompt interpretation or token truncation. In commit 28458 (Jetty), ChatGPT evaluated the wrong snippet, generating an unsupported "tie" outcome. Such cases suggest transient parsing or context-window errors rather than reasoning flaws.

Mapping these mechanisms by refactoring type Figure 4) shows that divergences were overwhelmingly concentrated in *Extract Method* refactorings, which represented 24 of the 33 cases (72.7%). With these, contextual blindness (11 cases) and semantic overgeneralization (8 cases) were dominant, suggesting errors are common in refactorings requiring inter-method reasoning and state preservation. Stylistic/Naming biases mainly occurred in *Rename Variable* (5 of 8 cases), rewarding lexical consistency over semantic neutrality. Overall, LLM semantic reasoning is challenged by refactorings altering control flow or dependencies, while purely lexical ones expose over-reliance on style.

Cross-Model Divergence. Direct model-to-model comparison showed 11 opposite preferences between the models. ChatGPT 5-mini better aligned with humans on refactorings emphasizing semantic clarity and naming (e.g. *Rename Variable*) LLaMA Scout performed slightly better on low-level iterative operations where syntactic regularity suggested correctness. This suggests ChatGPT 5-mini relies more on semantic cues, while LLaMA Scout prioritizes structural patterns. In 21 human-disagreement commits, both model's errors stemmed from contextual blindness, evaluating isolated snippets without class-level awareness.

Quality Dimensions Behind Disagreements. Across the 33 divergent commits, ChatGPT 5-mini penalized the refactored version mainly for *maintainability* (7 cases) or *readability* (4 cases), producing 2 ties. LLaMA Scout, in turn, penalized it mainly for *naming* (6 cases) or *readability* (5 cases), and tied in 12 ambiguous cases. These distributions reinforce that the models interpret the three evaluation dimensions unevenly: ChatGPT is more conservative with maintainability, while LLaMA reacts more strongly to surface-level readability and naming cues.

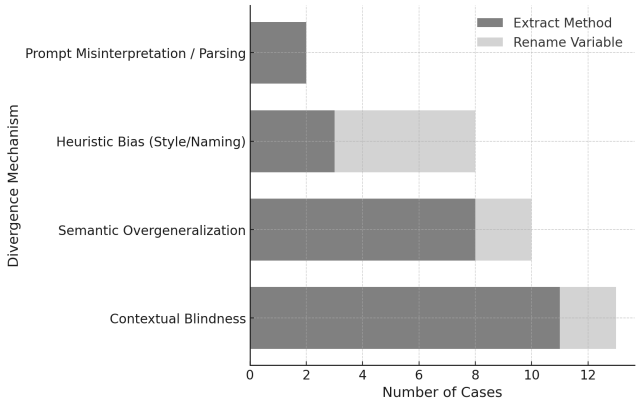


Figure 4: Divergence Mechanisms by Refactoring Type

Interpretation. ChatGPT 5-mini follows a conservative heuristic, penalizing modifications it cannot fully trace. On the other hand, LLaMA Scout applies an optimistic heuristic, rewarding apparent structural simplification without verifying semantic preservation or behavioral equivalence. Both conflate surface readability with deeper design quality and lack the capacity to infer architectural intent beyond the snippet boundaries. This explains their strong performance on simple refactorings and their consistent failures in context-dependent evaluations.

Summary of RQ2. Among the 33 divergent cases, 13 (39.4%) involved contextual blindness, 10 (30.3%) semantic overgeneralization, 8 (24.2%) heuristic bias toward style, and 2 (6.1%) prompt or parsing errors. Contextual and semantic divergences dominated *Extract Method* cases, while stylistic biases prevailed in *Rename Variable*. When Code 2 lost, LLMs primarily penalized maintainability (ChatGPT) or naming/readability (LLaMA), revealing their reliance on syntactic and stylistic proxies rather than contextual or behavioral reasoning.

5 Discussion

Beyond raw agreement scores, this section interprets our results to reveal the cognitive, practical, and methodological boundaries of LLM-based code review. In other words, we explore: *what kind of reasoning do LLMs employ when they agree or disagree with humans?* This exploration outlines the theoretical mechanisms of human–AI misalignment and derives implications for practice, tool design, and future research.

5.1 From Agreement to Understanding

Our findings demonstrate that LLMs can frequently *agree* with human reviewers, particularly in simple and localized refactorings, but agreement alone does not imply shared understanding. ChatGPT 5-mini aligned with humans in 91.8% of cases and LLaMA Scout in 82.9%, yet the qualitative analysis revealed that both models often reached correct conclusions through shallow heuristics rather than semantic reasoning. In other words, models reproduce the *grammar of review reasoning*, favoring shorter, cleaner, or well-named code without necessarily grasping the underlying design intent [7, 29, 36].

This distinction between *syntactic alignment* and *semantic alignment* reflects a well-recognized cognitive limitation of current LLMs [8, 37]. Models excel at mirroring surface-level linguistic and stylistic cues that correlate with quality (e.g., naming consistency, reduced indentation depth), but struggle to interpret behavioral or architectural consequence [33, 38, 47, 53]. From a cognitive-psychology standpoint, they exhibit what Weber [20] called *form-level rationality*: reasoning from observable patterns, rather than from design goals and causal mechanisms. In contract, *goal-level rationality* requires understanding *why* a particular design decision improves maintainability, cohesion, or testability [20]. This explains why the strongest agreements occurred in lexical refactorings (e.g., *Rename Variable*), while the deepest divergences appeared in *Extract Method*—scenarios demanding inter-method reasoning and awareness of shared state.

From a software engineering perspective, current LLMs simulate human *evaluation outcomes* but not human *judgment processes* [43, 48]. They converge on similar decisions when readability is the dominant signal, yet diverge when the task requires understanding hidden dependencies, side effects, or implicit domain conventions. Therefore, high agreement rates should be interpreted as evidence of *behavioral mimicry*, that is the model's ability to reproduce the external behaviors or decision patters of human reviewers without necessarily internalizing their underlying reasoning or intent [8, 21, 28]. In practice, LLM-based evaluators must thus evolve from detecting stylistic conformity to modeling the deeper semantics and causal logic that drive design reasoning.

5.2 The Cognitive Limits of LLM Reviewers

The divergence mechanisms identified in RQ2 expose the cognitive boundaries of the current LLM reasoning. While human reviewers interpret code through a combination of contextual knowledge, intent inference, and shared project norms, LLMs operate within a narrow syntactic window. They approximate the behavior of expert reasoning, but lack the latent models of the system state, the developer intention, and design trade-offs that underpin human judgment [8, 43].

Three patterns are particularly revealing. First, *contextual blindness*, responsible for 39.4% of all divergences, shows that LLMs perceive code as isolated text rather than as a system of interacting entities. When extraction of a method affects the shared state or parameter coupling, both ChatGPT 5-mini and LLaMA Scout fail to propagate this dependency information through their internal representation. In cognitive terms, they reason locally but not relationally. That is, they can "*see*" the structure of the code but not its operational semantics.

Second, *semantic overgeneralization* (30.3%) reveals a complementary bias. Faced with partial evidence, the models extrapolate global properties from superficial cues, penalizing legitimate refactorings that shorten code or centralize logic because they misclassify brevity as loss of functionality. This mirrors the human bias of *attribute substitution* [19, 45], where evaluators unconsciously replace a difficult question ("Does this refactoring preserve behavior?") with

an easier one ("Does it look simpler?"). The result is an illusion of reasoning rather than a grounded semantic judgment.

Third, *heuristic bias toward style and naming* (24.2%) indicates that both models overfit to surface regularities learned during instruction tuning. They treat aesthetic consistency as a proxy for maintainability, rewarding cosmetic coherence even when it degrades design quality. This bias emerges from reinforcement learning with human feedback (RLHF): Models internalize annotator preferences for clean and well-formatted snippets, but receive no penalty for architectural misunderstanding [17, 31, 40]. Consequently, their internal reward landscape is dominated by stylistic cues, not causal correctness.

Together, these mechanisms delineate what we call the *semantic frontier* of current LLM-based code reviewers. Beyond this frontier, models cease to reason about meaning and revert to probabilistic pattern matching [8, 37]. They perform well when quality can be inferred from syntax, but fail when it depends on implicit context, system-level cohesion, or design intent. Recognizing this boundary is crucial, as it defines where automation can safely augment human review and where human expertise remains indispensable.

5.3 Implications for Code Review Practices

Understanding the cognitive limits of LLMs help us reason more precisely about where they can and cannot contribute to software review workflows. Our findings confirm that LLMs can reproduce human-like preferences in low-context, stylistic assessments but remain unreliable when reasoning about semantics or design trade-offs. These findings suggest that their value lies not in full automation, but in carefully scoped collaboration with human reviewers.

LLMs as pre-screeners and conversation starters. Consistent with previous work on automated refinement and "*LLM-as-a-judge*" paradigms [10, 18, 48], LLMs can operate effectively as *first-pass reviewers*. They can highlight obvious naming inconsistencies, redundant code fragments, or low-risk *Extract Method* refactorings before human inspection. In this role, models act as cognitive amplifiers: they surface potential improvements and free reviewers to focus on higher-order reasoning. However, unchecked automation can also normalize stylistic uniformity at the expense of design diversity, an effect observed in generation-based reviewers that overfit to superface regularities [50, 55]. Thus, model-assisted reviews should prioritize prompting diversity and rationale transparency to avoid reinforcing superficial consensus.

Division of labor in human–AI review. A key implication of our results is the need to redefine the boundary between automated and human evaluation. We envision a *division of cognitive labor* in which LLMs handle localized, rule-based assessments, while humans arbitrate on semantic and architectural coherence. Caumartin et al. [10] and White et al. [50] both highlight that hybrid review settings, where AI suggestions are scrutinized by experts, produce higher trust and accuracy than either component alone. In our data, such complementarity would have resolved most "contextual blindness" cases, since human reviewers readily identified dependency losses that escaped the textual reasoning of the models.

Socio-technical alignment and trust calibration. Code review is not a purely technical act; it is a socio-technical negotiation of design norms and shared accountability [5, 6]. Integrating LLMs into this process requires trust calibration: reviewers must understand when to rely on a stylistic precision of the model and when to override it with contextual judgment. Our results imply that confidence scores or rationale-rich explanations could help users gauge when an LLM decision is context-robust versus syntax-bound. Recent studies on explainable review assistance [14, 48] point in this direction, suggesting that transparency, not raw accuracy, is the key to sustainable adoption.

Operational integration in real workflows. From a tooling perspective, our analysis suggests three integration points: (i) *pre-commit evaluation*, where LLMs flag potential readability or naming issues; (ii) *interactive review support*, where they generate alternative rationales for disputed changes; and (iii) *post-review reflection*, where models summarize common reasoning patterns for team retrospectives. These stages reflect different trust gradients and can be adapted to project maturity or criticality, echoing the staged deployment strategies of Guo et al. [18]. Rather than positioning LLMs as standalone reviewers, we propose treating them as adaptive collaborators within a continuous feedback loop.

Toward benchmarked role definitions. The differentiation between *Extract Method* and *Rename Variable* refactorings suggests a pragmatic basis for defining LLM review roles. Semantic refactorings push the model's reasoning frontier and should remain human-led, while lexical refactorings provide a safer automation space. This distinction can guide benchmark design and evaluation metrics, promoting systematic measurement of how semantic complexity affects model reliability, what we term *Refactoring Sensitivity Index*. Such a metric would enable more principled integration of LLMs into industrial review pipelines, ensuring that automation scales where it adds value and defers where it risks error.

5.4 Designing Context-Aware Review Tools

While Section 5.3 defined when to use LLMs, this section considers *how* to design them. Our findings reveal that the main barrier to a reliable LLM-based code review is not model capacity, but contextual isolation. Both ChatGPT 5-mini and LLaMA Scout reasoned well about local syntax but failed when quality judgments required class-level dependencies, shared state, or design intent. This limitation implies that future review tools must move beyond isolated prompt–response paradigms toward architectures that integrate contextual information from the development environment. Based on this our analysis, we outline principles discussed below.

Principle 1: Context amplification through structured metadata. Code rarely carries its rationale within itself. Effective review requires access to complementary artifacts (e.g., commit messages, test coverage, dependency graphs, and design documentation) that inform why a change exists. Augmenting model prompts with such structured metadata can mitigate contextual blindness, enabling LLMs to evaluate consistency across modules rather than within snippets. Recent work on multi-context prompting [50, 52] demonstrates that hierarchical context injection (e.g., including coupled methods or call graphs) substantially improves reasoning accuracy for refactoring and defect detection tasks.

Principle 2: Multi-granular reasoning pipelines. A promising direction is to couple local and global analysis stages, where

lightweight LLM evaluations operate at the method level and higher-level semantic checks are delegated to static or graph-based analyzers. Such hybrid architectures, which combine statistical inference with structural reasoning, have proven to be effective in related domains, such as vulnerability detection [16] and code smell localization [13]. In review scenarios, this approach could detect when a refactoring alters control flow or coupling metrics, prompting the model to re-evaluate its earlier stylistic preference in light of broader system behavior.

Principle 3: Explicit role framing and prompt conditioning. Our results indicate that the absence of clear evaluative structure often leads LLMs to default aesthetic heuristics. Prompt engineering can partially address this problem by explicitly defining the model review objective, for instance, "evaluate maintainability impact" or "assess side-effect safety", rather than generic notions of "better code." Previous studies [10, 55] show that goal-specific framing reduces stylistic bias and encourages reasoning aligned with maintainability and cohesion criteria. Embedding such intent conditioning within integrated development environments (IDEs) could ensure consistency across projects and teams.

Principle 4: Interactive and explainable feedback. Since LLM reasoning is probabilistic and opaque, users must be able to question its rationale. Interactive review interfaces that expose decision rationales ("why Code 2 is preferred?") and confidence levels would allow developers to challenge or refine model judgments. This aligns with emerging practices in explainable AI for SE [14, 48], which emphasize transparency as the foundation for trust calibration. In our study, many false positives could have been corrected if the model had explicitly displayed its assumptions about the variable scope or method dependence.

Principle 5: Continuous human-in-the-loop adaptation. Finally, context-aware review systems should learn from divergence cases over time. When human reviewers override an LLM recommendation, this feedback can serve as a supervised signal for future model adaptation, progressively aligning the system with project-specific conventions [1, 11]. This incremental fine-tuning approach parallels strategies successfully adopted in adaptive testing and defect triage pipelines [18, 50]. By embedding reflective feedback loops into review workflows, tools can evolve from static evaluators to co-adaptive collaborators.

Together, these principles outline a design research agenda for next-generation AI-assisted review environments. Rather than treating prompts as disposable text queries, we advocate for *context-aware ecosystems* that situate LLM reasoning within the full socio-technical landscape of software development. Such ecosystems can transform LLMs from isolated predictors into reliable partners capable of supporting nuanced, semantically grounded code review.

5.5 Research and Industrial Implications

Beyond model benchmarking, this study expands understanding of how human an AI code reviewers co-evolve in software engineering. Next, we discuss implications for research, practice, and the evaluation of future LLM-based review systems.

Implications for research. Our findings emphasize the need for new empirical frameworks that go beyond accuracy metrics and capture the *reasoning alignment* between humans and LLMs. Current evaluation protocols, focused on outcome agreement, hindering whether models reach the same conclusions for the same reasons. Future research should therefore include reasoning trace analysis and discourse-level coding of LLM justifications, enabling researchers to measure *epistemic alignment*: the degree to which model rationales reflect human evaluative heuristics. This observation aligns with the recent shift in artificial intelligence for software engineering (AI-for-SE) studies [48, 50] that treat model explanations, not just answers, as first-class artifacts for analysis. In this sense, divergence mechanisms, such as contextual blindness and heuristic bias, offer a taxonomy for diagnosing failure modes in code reasoning models.

Implications for benchmark design. The contrast between *Extract Method* and *Rename Variable* refactorings highlights the need for more fine-grained benchmarking of semantic versus lexical review challenges. We propose the notion of a *Refactoring Sensitivity Index (RSI)*, a metric that quantifies performance degradation as semantic complexity increases. Such an index would allow the community to systematically measure how reasoning depth, context length, and dependency density affect model reliability, an essential step toward reproducible and interpretable evaluation of code-review capabilities. This proposal complements recent calls for richer, task-specific benchmarks in code reasoning [13, 52], extending them to the social and evaluative dimensions of review.

Implications for industrial adoption. In practical settings, our results suggest a paradigm shift from LLMs as autonomous reviewers to *co-reviewers* integrated into socio-technical workflows. Industrial pipelines could take advantage of LLMs for initial screening and automated comment generation, reserving human oversight for semantically complex or safety-critical code. Several organizations have begun to experiment with such tiered pipelines [10, 18], where models pre-assess differences and humans resolve disagreements. Our findings provide empirical grounding for this approach. Most false positives in our dataset could have been caught by lightweight human verification, while repetitive stylistic checks could safely be automated. To ensure accountability, LLM outputs should be versioned, explainable, and auditable, mirroring existing standards for static analysis and continuous integration.

Bridging academia and practice. The divergences identified in this study open opportunities for cross-sector collaboration. Academia can provide taxonomies and metrics for reasoning alignment, while industry can supply large-scale annotated review logs for fine-tuning and validation. Together, these efforts could yield an open benchmark suite for *LLM-in-the-loop code review*, capturing both stylistic and semantic dimensions. This collaboration would accelerate progress toward *contextually grounded AI reviewers*. That is, systems capable not only of identifying code improvements, but also of explaining their rationale aligned with human reasoning.

In summary, our study positions LLMs not as replacements for human reviewers but as catalysts for reimagining the review process itself. By clarifying the boundaries of their competence, the semantic frontier, and by proposing metrics and design principles for crossing it, we outline a path towards more explainable, context-aware, and human-aligned review ecosystems.

6 Threats to Validity

In this section, we discuss the main threats to the validity of our study [51] and the decisions we made to mitigate them.

Internal validity. First, although we used a fixed and standardized prompt for both models, slight differences in phrasing or formatting details could still bias the way each LLM interprets the evaluation criteria (clarity, readability, and maintainability). Moreover, both models are stochastic by nature; running each evaluation only once may have introduced variability due to non-deterministic sampling. Future replications could mitigate this by performing multiple runs per code pair and aggregating the outcomes. Another potential bias arises from the manual post-processing of responses — the classification of free-text outputs into the categories *Code 1*, *Code 2*, or *Tie* might embed subjective interpretation, especially for borderline or ambiguous cases.

Construct validity. We defined 'alignment' as the match between the model's decision and the preferred version of the human reviewers from the MaRV dataset. This definition assumes that the reviewers' chosen version indeed represents higher quality and that their decisions are consistent across projects. Such assumptions may oversimplify human judgment, as reviewers' acceptance decisions can depend on context, maintainability trade-offs, or project-specific conventions. Additionally, our prompts deliberately excluded contextual metadata (e.g., commit messages or discussion threads), focusing only on code snippets. Although this design isolates code-level reasoning, it under-represents the contextual cues typically available to human reviewers, potentially limiting the ecological realism of the task.

External validity. Our analysis was based exclusively on 146 Java refactoring commits (specifically *Extract Method* and *Rename Variable*) from the MaRV dataset [30]. Hence, the results may not generalize to other refactoring types, programming languages, or organizational review practices. Future studies should replicate this investigation across multiple languages, datasets, and industrial settings to verify whether our observations hold in more diverse code-review environments. Furthermore, since both LLMs were evaluated through public APIs, differences in model versioning or inference infrastructure could also limit reproducibility over time. Explicitly documenting API versions or model checkpoints would help strengthen external validity.

Non-deterministic Given the non-deterministic nature of large language models (LLMs), a single prompt execution may substantially affect the reliability of the results. To mitigate this issue, we selected 10% of the dataset and executed the same prompt five times, enabling an assessment of result stability and the robustness of the reported agreement rates. The analysis shows a high degree of consistency across repeated executions, with the vast majority of classifications remaining unchanged and predominantly assigned to the same label across runs, for both ChatGPT and Llama models. Although we observed minor divergences, such variations were sparse, non-systematic, and limited to a minority of executions. Importantly, no progressive drift or degradation in classification behavior was identified across successive runs. These findings suggest that, despite inherent stochasticity, the applied prompting strategy yields stable outcomes, supporting the robustness of the agreement rates and the validity of conclusions derived from the analysis.

Conclusion validity. Finally, our inferences about model effectiveness and alignment are limited by the scope of statistical and qualitative analyses. Although we employed McNemar's test and Cohen's κ to assess agreement, these metrics capture only categorical consistency and not the underlying strength or quality of reasoning. Hence, the reported differences between LLMs should be interpreted as indicative rather than definitive.

7 Conclusion and Future Work

This paper investigated the extend to which large language models can act as reliable evaluators in refactoring code reviews. By comparing the judgments of two representative models — ChatGPT 5-mini and LLaMA 4 Scout — against 146 human-validated refactoring commits, we provided quantitative and qualitative evidence of their current capabilities and boundaries. Our results show that LLMs can reproduce human-like preferences in low-context refactorings (ChatGPT 5-mini: 91.8% and LLaMA 4 Scout: 82.9%), yet their reasoning remains shallow and heavily biased toward stylistic and naming heuristics. Both models exhibit limited semantic comprehension and fragile maintainability reasoning, underscoring that high agreement does not necessarily imply genuine understanding.

Future work will focus on advancing *context-aware* and *human-aligned* code review systems. This involves expanding evaluation to include more refactoring types, programming languages, industrial datasets, and newer code-specialized LLMs (e.g., Claude, Copilot). We also plan to explore hybrid review pipelines that integrate LLM reasoning with structured metadata, static analysis, and human feedback. We aim to create next-generation code review tools that combine statistical precision with contextual depth [46], moving beyond mimicking human judgment to truly augmenting it.

Acknowledgements. This research was partially supported by Brazilian funding agencies: FAPEMIG (Grant APQ-01488-24), CNPq (Grants 446729/2024-8 and 406089/2025-6), and CAPES.

References

[1] Saleema Amershi, Maya Cakmak, W. Bradley Knox, and Todd Kulesza. 2014. Power to the People: The Role of Humans in Interactive Machine Learning. In *AI Magazine*, Vol. 35. 105–120. https://doi.org/10.1609/aimag.v35i4.2513

[2] Larisse Amorim, Ivandeclei Mendes da Costa, Leticia Alves, and Eduardo Figueiredo. 2025. Bad smell Detection using Google Gemini. In *2025 IEEE 49th Annual Computers, Software, and Applications Conference (COMPSAC)*. 1637–1642.

[3] Larisse Amorim, Caique Fortunato, Gustavo Vale, and Eduardo Figueiredo. 2025. High Agreement, Shallow Reasoning: A Mixed-Method Study of LLMs in Refactoring Reviews. In *Replication Package*. https://github.com/spLari/llm-code-review

[4] Roberta Arcoverde, Alessandro Garcia, and Eduardo Figueiredo. 2011. Understanding the Longevity of Code Smells: Preliminary Results of an Explanatory Survey. In *Proceedings of the 4th Workshop on Refactoring Tools*. 33–36.

[5] Alberto Bacchelli and Christian Bird. 2013. Expectations, outcomes, and challenges of modern code review. In *35th International Conference on Software Engineering (ICSE)*. IEEE, 712–721.

[6] Gabriele Bavota and Barbara Russo. 2015. Four eyes are better than two: On the impact of code reviews on software quality. In *IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 81–90.

[7] David Binkley, Matthew Davis, Dawn Lawrie, Jonathan I. Maletic, Christopher Morrell, and Bonita Sharif. 2013. The impact of identifier style on effort and comprehension. *Empirical Software Engineering* 18, 2 (2013), 219–276. https://doi.org/10.1007/s10664-012-9201-4

[8] Marcel Binz and Eric Schulz. 2023. Turning large language models into cognitive models. (2023). arXiv:2306.03917 [cs.CL] https://arxiv.org/abs/2306.03917

[9] Amiangshu Bosu and Jeffrey C. Carver. 2013. Impact of Peer Code Review on Peer Impression Formation: A Survey. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 133–142.

[10] Genevieve Caumartin, Qiaolin Qin, Sharon Chatragadda, Janmitsinh Panjrolia, Heng Li, and Diego Costa. 2025. Exploring the Potential of Llama Models in Automated Code Refinement: A Replication Study. In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 681–692.

[11] Paul F. Christiano, Jan Leike, Tom Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep Reinforcement Learning from Human Preferences. In *Advances in Neural Information Processing Systems (NeurIPS 2017)*. 4299–4307.

[12] Kattiana Constantino, Mauricio Souza, Shurui Zhou, Eduardo Figueiredo, and Christian Kästner. 2023. Perceptions of Open-Source Software Developers on Collaborations: An Interview and Survey Study. *Journal of Software: Evolution and Process* 35, 5 (2023), e2393.

[13] Jonathan Cordeiro, Shayan Noei, and Ying Zou. 2024. An empirical study on the code refactoring capability of large language models. *ACM Transactions on Software Engineering and Methodology (TOSEM)* (2024).

[14] Ramita Deeprom, Shiyu Yang, Yoshiki Higo, Morakot Choetkiertikul, and Chaiyong Ragkhitwetsagul. 2024. Challenges in Adopting LLaMA: An Empirical Study of Discussions on Stack Overflow. In *CEUR Workshop Proceedings*, Vol. 3864. CEUR-WS, 35–42.

[15] Martin Fowler. 2019. *Refactoring: improving the design of existing code*. Addison-Wesley.

[16] José Gonçalves, Miguel Silva, Bernardo Cabral, Tiago Dias, Eva Maia, Isabel Praça, Ricardo Severino, and Luís Lino Ferreira. 2025. Evaluating LLaMA 3.2 for Software Vulnerability Detection. arXiv:2503.07770 [cs.LG] https://arxiv.org/abs/2503.07770

[17] Everton Guimaraes, Alessandro Garcia, Eduardo Figueiredo, and Yuanfang Cai. 2013. Prioritizing Software Anomalies with Software Metrics and Architecture Blueprints. In *5th International Workshop on Modeling in Software Engineering (MiSE)*. 82–88.

[18] Qi Guo, Junming Cao, Xiaofei Xie, Shangqing Liu, Xiaohong Li, Bihuan Chen, and Xin Peng. 2024. Exploring the potential of chatgpt in automated code refinement: An empirical study. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering*. 1–13.

[19] Daniel Kahneman and Shane Frederick. 2002. Representativeness Revisited: Attribute Substitution in Intuitive Judgment. In *Heuristics and Biases: The Psychology of Intuitive Judgment*, Thomas Gilovich, Dale Griffin, and Daniel Kahneman (Eds.). Cambridge University Press, 49–81.

[20] Stephen Kalberg. 1980. Max Weber's Types of Rationality: Cornerstones for the Analysis of Rationalization Processes in History. *Amer. J. Sociology* 85, 5, 1145–1179. http://www.jstor.org/stable/2778894

[21] David La Barbera, Riccardo Lunardi, Mengdie Zhuang, and Kevin Roitero. 2025. Impersonating the Crowd: Evaluating LLMs' Ability to Replicate Human Judgment in Misinformation Assessment *(ICTIR '25)*. Association for Computing Machinery, New York, NY, USA, 12–21. https://doi.org/10.1145/3731120.3744581

[22] J. Richard Landis and Gary G. Koch. 1977. The Measurement of Observer Agreement for Categorical Data. *Biometrics* 33 (1977), 159–174.

[23] Zhiyu Li, Shuai Lu, Daya Guo, Nan Duan, Shailesh Jannu, Grant Jenks, Deep Majumder, Jared Green, Alexey Svyatkovskiy, Shengyu Fu, et al. 2022. Automating code review activities by large-scale pre-training. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1035–1047.

[24] Marry L. McHugh. 2012. Interrater reliability: the kappa statistic. *Biochem Med (Zagreb)* 22, 3 (2012), 276–282.

[25] Quinn McNemar. 1947. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika* 12, 2 (June 1947), 153–157.

[26] Rodrigo Morales, Shane McIntosh, and Foutse Khomh. 2015. Do code review practices impact design quality? a case study of the qt, vtk, and itk projects. In *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. IEEE, 171–180.

[27] Murtuza Mukadam, Christian Bird, and Peter C Rigby. 2013. Gerrit software code review data from android. In *10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 45–48.

[28] Toshiya Murashige and Takayuki Ito. 2025. Simulating Human Decision-Making in Ultimatum Games using Large Language Models. In *Proceedings of the ACM Collective Intelligence Conference (CI '25)*. 13–19.

[29] Henrique Nunes, Eduardo Figueiredo, Larissa Rocha, Sarah Nadi, Fischer Ferreira, and Geanderson Esteves. 2025. Evaluating the Effectiveness of LLMs in Fixing Maintainability Issues in Real-world Projects. In *IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. 669–680.

[30] Henrique Nunes, Tushar Sharma, and Eduardo Figueiredo. 2025. MaRV: A Manually Validated Refactoring Dataset. In *IEEE/ACM International Conference on AI Foundation Models and Software Engineering (FORGE)*. IEEE, 141–145.

[31] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems* 35 (2022), 27730–27744.

[32] Wendkûuni C. Ouédraogo, Yinghua Li, Xueqi Dang, Pawel Borsukiewicz, Xin Zhou, Anil Koyuncu, Jacques Klein, David Lo, and Tegawendé F. Bissyandé. 2025. Human-Aligned Code Readability Assessment with Large Language Models. arXiv:2510.16579 [cs.SE] https://arxiv.org/abs/2510.16579

[33] Wendkûuni C. Ouédraogo, Yinghua Li, Xueqi Dang, Xin Zhou, Anil Koyuncu, Jacques Klein, David Lo, and Tegawendé F. Bissyandé. 2025. Beyond Surface Similarity: Evaluating LLM-Based Test Refactorings with Structural and Semantic Awareness. arXiv:2506.06767 [cs.SE] https://arxiv.org/abs/2506.06767

[34] Matheus Paixão, Anderson Uchôa, Ana Carla Bibiano, Daniel Oliveira, Alessandro Garcia, Jens Krinke, and Emilio Arvonio. 2020. Behind the intents: An in-depth empirical study on software refactoring in modern code review. In *Proceedings of the 17th International Conference on Mining Software Repositories*. 125–136.

[35] Chanathip Pornprasit and Chakkrit Tantithamthavorn. 2024. Fine-tuning and prompt engineering for large language models-based code review automation. *Information and Software Technology* 175 (2024), 107523.

[36] Daryl Posnett, Abram Hindle, and Premkumar Devanbu. 2011. A simpler model of software readability. In *Proceedings of the 8th Working Conference on Mining Software Repositories (MSR)*. 73–82.

[37] Milena Rmus, Akshay K. Jagadish, Marvin Mathony, Tobias Ludwig, and Eric Schulz. 2025. Generating Computational Cognitive Models using Large Language Models. (2025). arXiv:2502.00879 [cs.LG] https://arxiv.org/abs/2502.00879

[38] Cláudio Sant'Anna, Eduardo Figueiredo, Alessandro Garcia, and Carlos JP Lucena. 2007. On the Modularity of Software Architectures: A Concern-driven Measurement Framework. In *European Conference on Software Architecture (ECSA)*. 207–224.

[39] Igor Simões and Elaine Venson. 2024. Evaluating Source Code Quality with Large Language Models: a comparative study. In *Proceedings of the Brazilian Symposium on Software Quality (SBQS '24)*. 103–113.

[40] Nisan Stiennon, Long Ouyang, Jeffrey Wu, Daniel Ziegler, Ryan Lowe, Chelsea Voss, Alec Radford, Dario Amodei, and Paul F Christiano. 2020. Learning to summarize with human feedback. *Advances in neural information processing systems* 33 (2020), 3008–3021.

[41] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded theory in software engineering research: a critical review and guidelines. In *Proceedings of the 38th International Conference on Software Engineering (ICSE)*. 120–131.

[42] Anselm L. Strauss and Juliet M. Corbin (Eds.). 1997. *Grounded Theory in Practice*. SAGE Publications, Thousand Oaks, CA.

[43] Annalisa Szymanski, Noah Ziems, Heather A. Eicher-Miller, Toby Jia-Jun Li, Meng Jiang, and Ronald A. Metoyer. 2025. Limitations of the LLM-as-a-Judge Approach for Evaluating LLM Outputs in Expert Knowledge Tasks. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*. 952–966.

[44] Rosalia Tufano, Luca Pascarella, Michele Tufano, Denys Poshyvanyk, and Gabriele Bavota. 2021. Towards automating code review activities. In *IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 163–174.

[45] Amos Tversky and Daniel Kahneman. 1974. Judgment under Uncertainty: Heuristics and Biases. *Science* 185, 4157 (1974), 1124–1131.

[46] Gustavo Vale, Claus Hunsen, Eduardo Figueiredo, and Sven Apel. 2021. Challenges of Resolving Merge Conflicts: A Mining and Survey Study. *IEEE Transactions on Software Engineering* 48, 12 (2021), 4964–4985.

[47] Alejandro Velasco. 2024. Beyond Accuracy: Evaluating Source Code Capabilities in Large Language Models for Software Engineering. In *46th International Conference on Software Engineering: Companion Proceedings*. 162–164.

[48] Ruiqi Wang, Jiyu Guo, Cuiyun Gao, Guodong Fan, Chun Yong Chong, and Xin Xia. 2025. Can llms replace human evaluators? an empirical study of llm-as-a-judge in software engineering. *Proceedings of the ACM on Software Engineering* 2, ISSTA (2025), 1955–1977.

[49] Miku Watanabe, Yutaro Kashiwa, Bin Lin, Toshiki Hirao, Ken'Ichi Yamaguchi, and Hajimu Iida. 2024. On the use of chatgpt for code review: Do developers like reviews by chatgpt?. In *Proceedings of the 28th International Conference on Evaluation and Assessment in Software Engineering*. 375–380.

[50] Jules White, Sam Hays, Quchen Fu, Jesse Spencer-Smith, and Douglas C Schmidt. 2024. Chatgpt prompt patterns for improving code quality, refactoring, requirements elicitation, and software design. In *Generative AI for Effective Software Development*. Springer, 71–108.

[51] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer Berlin Heidelberg.

[52] Danning Xie, Mingwei Zheng, Xuwei Liu, Jiannan Wang, Chengpeng Wang, Lin Tan, and Xiangyu Zhang. 2025. CORE: Benchmarking LLMs' Code Reasoning Capabilities through Static Analysis Tasks. *arXiv preprint arXiv:2507.05269* (2025).

[53] Rem Yang, Julian Dai, Nikos Vasilakis, and Martin Rinard. 2025. Evaluating the Generalization Capabilities of Large Language Models on Code Reasoning. arXiv:2504.05518 [cs.SE] https://arxiv.org/abs/2504.05518

[54] Yongda Yu, Guoping Rong, Haifeng Shen, He Zhang, Dong Shao, Min Wang, Zhao Wei, Yong Xu, and Juhong Wang. 2024. Fine-Tuning Large Language Models to Improve Accuracy and Comprehensibility of Automated Code Review. *ACM Transactions Software Engineering and Methodology (TOSEM)* 34, 1, Article 14 (Dec. 2024), 26 pages.

[55] Xin Zhou, Kisub Kim, Bowen Xu, DongGyun Han, Junda He, and David Lo. 2023. Generation-based code review automation: How far are we?. In *IEEE/ACM 31st International Conference on Program Comprehension (ICPC)*. IEEE, 215–226.