What Types of Code Review Comments Do Developers Most Frequently Resolve?

Saul Goldman*, Hong Yi Lin*, Jirat Pasuksmit†, Patanamon Thongtanunam*, Kla Tantithamthavorn†§, Zhe Wang†, Ray Zhang†, Ali Behnaz†, Fan Jiang†, Michael Siers†, Ryan Jiang†, Mike Buller†, Minwoo Jeong‡, Ming Wu‡
*The University of Melbourne, Australia, †Atlassian, Australia. ‡Atlassian, USA. §Monash University, Australia.

***Abstract*—Large language model (LLM)-powered code review automation tools have been introduced to generate code review comments. However, not all generated comments will drive code changes. Understanding what types of generated review comments are likely to trigger code changes is crucial for identifying those that are actionable. In this paper, we set out to investigate (1) the types of review comments written by humans and LLMs, and (2) the types of generated comments that are most frequently resolved by developers. To do so, we developed an LLM-as-a-Judge to automatically classify review comments based on our own taxonomy of five categories. Our empirical study confirms that (1) the LLM reviewer and human reviewers exhibit distinct strengths and weaknesses depending on the project context, and (2) readability, bugs, and maintainability-related comments had higher resolution rates than those focused on code design. These results suggest that a substantial proportion of LLM-generated comments are actionable and can be resolved by developers. Our work highlights the complementarity between LLM and human reviewers and offers suggestions to improve the practical effectiveness of LLM-powered code review tools.**

***Index Terms*—Code Review, Actionable Code Review Comments, Online Experiment**

I. Introduction

Code review is a standard practice in modern software development, with developers typically spending 10–15% of their time on this task [1]–[3]. It plays a vital role in enhancing code quality and promoting team collaboration [4], [5]. A typical code review workflow requires a developer other than the code author to review and provide feedback for new code changes before merging into the main codebase. Despite its benefits, code review can be time-consuming and demands considerable effort from human reviewers [6], [7]. To alleviate this burden, Large Language Model (LLM)-powered code review tools have been developed to automate various code review activities (e.g., generating code review comments, code refinement) [8].

Review comment generation plays a vital role in code review automation as comments can trigger various improvements, contributing varying levels of value to the code review process. To better understand which types of code review comments are being generated, researchers manually classified a generated comment into a category [9], [10]. Understanding this is crucial for advancing LLM-powered code review tools and guiding product design and development, which subsequently enhances code review productivity. In particular, when comments are routinely accepted or resolved with code changes, it demonstrates strong alignment with developers' needs and workflows, thereby driving tool adoption and amplifying their impact on developer productivity.

In this paper, we set out to investigate (1) the comment types generated by LLM reviewers compared to those provided by humans, and (2) the types of LLM-generated comments that are most frequently resolved by developers. To do so, we first developed a taxonomy of comment types based on five categories (i.e., readability, bugs, maintainability, design, and no issue). Then, we developed an LLM-as-a-Judge to automatically categorize review comments based on the aforementioned taxonomy. After confirming the Judge's reliability, we categorized both LLM-generated and human-written comments, enabling us to analyze the differences in issue types discussed. Finally, we classified comments that were previously generated for internal projects at Atlassian to facilitate a triangulated evaluation of resolution rate by comment type. Then, we answer the following RQs:

RQ1) How do the types of code review comments written by humans differ from those generated by an LLM? For Atlassian projects, the LLM reviewer generated a higher proportion of comments related to bugs and maintainability, but fewer comments focused on readability compared to human reviewers. For OSS projects, the LLM reviewer generated more maintainability-related comments but fewer design-related comments than human reviewers did. Nonetheless, bug-related comments are less-frequently generated by an LLM reviewer.

RQ2) What types of LLM-generated code review comments are frequently resolved by developers? Readability, bug, and maintainability-related comments exhibited higher resolution rates compared to those focused on code design. Nevertheless, many of the LLM-generated comments are not resolved by developers. Such unresolved comments could be due to the lack of clarity, relevancy, and simplicity of the comments.

These findings lead us to conclude that the LLM reviewer and human reviewers exhibit distinct strengths and weaknesses depending on the project context. While LLM-generated comments about readability, bugs, and maintainability are more likely to be addressed by developers,

2643-1572/25/$31.00 ©2025 IEEE
DOI 10.1109/ASE63991.2025.00312

bug-related comments remain underrepresented in LLM comments. Balancing the distribution of comment types and improving the quality and clarity of LLM-generated comments is therefore crucial for increasing their impact in code review workflows.

II. Background and Related Work

Review Comment Categorization. Various taxonomies of comment types have been developed based on human-written review comments [11]–[13]. In general, code reviews discuss either functional or evolvability issues. The former encompasses defects that can cause system failures at execution time (e.g., incorrect implementation, logical errors, mistiming), while the latter involves issues that affect the compliance, maintainability and understandability of the codebase (e.g., documentation, refactoring, code readability) [11]. To understand benefits of code reviewing, prior studies explored how different types of human-written comments relate to software quality [14] and developer-perceived usefulness [15], [16]. Despite extensive research on the comment types in human-written comments, the differences in their distribution compared to those in generated comments remain largely unexplored.

Code Review Comment Generation and its Evaluation Recent studies have explored automating review comment generation to reduce the manual effort involved in code reviews. Many approaches use sequence-to-sequence models—such as the original Transformer [17], T5 [18], and CodeT5 [19]—or leverage large language models (LLMs) like LLaMA [20], GPT-3.5 [21], and GPT-4o [22]. Generated comments are typically evaluated against human-written references using surface-level similarity metrics such as BLEU, ROUGE, and Exact Match, which measure lexical similarity. However, these metrics overlook the diversity and semantics. To address this, recent work [9], [10], [23]–[25] have conducted manual classification of comment types to further evaluate the generated review comments. Although recent work has explored the use of an LLM as a judge to classify code review comments, the primary focus has been on curating datasets for training purposes [26]. Despite the valuable insights of comment types, little is known about which types of generated comments developers actually resolve in practice.

III. Research Methodology

In this section, we present the motivation of our research questions and the research methodology.

A. Research Questions

RQ1: How do the types of code review comments written by humans differ from those generated by an LLM?
Motivation. Investigating differences in the distribution of comment types between human-written and LLM-generated code review comments can reveal the distinct strengths and limitations of LLMs in reviewing code changes. By identifying these differences, we gain insight into the comment types that LLMs tend to prioritize and their behavior aligns with the code review practices of human reviewers.

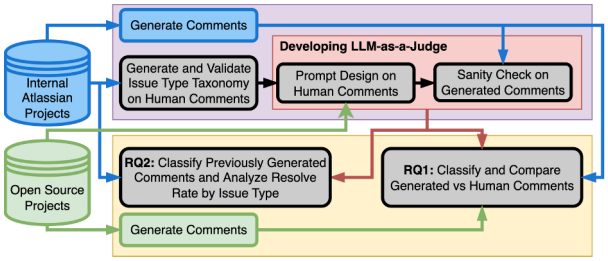

Fig. 1. An overview of our research methodology to answer the research questions.

RQ2: What types of LLM-generated code review comments are frequently resolved by developers?
Motivation. While LLMs can generate a wide range of review comments, it remains unclear which comment types are more likely to lead to comment resolution (i.e., the corresponding code line is changed in the subsequent commit). Understanding the comment types that are most often resolved by developers is crucial for understanding the impact of LLM-generated comments in real-world use cases. Such insights will help inform future efforts to enhance the effectiveness of LLM-powered code review tools.

B. A Taxonomy of Comment Types

To better understand the types of code review comments, we constructed a taxonomy of comment types that are mostly aligned with human-written code review comments at Atlassian. We began with a random sample of 336 human-written comments from the merged pull requests. To ensure the quality of review comments, we selected only comments longer than 20 characters, inserted to a line in a source code or test file (excluding auxiliary files such as compiled bytecode, dependency locks, or static assets), and not written by the pull request author or automated bots [10].

To construct a taxonomy, six Atlassian software engineers conducted card sorting on the sampled review comments to explore the types of code review comments. The annotators iteratively refined the category definitions together until a consensus was reached.[1] Table I presents the six comment types and their descriptions.

C. An LLM Judge for Review Comment Classification

To classify code review comments using our taxonomy, we employed OpenAI's `gpt-4.1-2025-04-14` as an annotator. As recent studies have demonstrated the effectiveness of LLMs-as-a-Judge [26], [27], this approach will enable us to efficiently categorize a large volume of comments. To ensure clarity and consistency, we designed prompts that are succinct, informative, and structured using a JSON format (Figure 2). Each prompt includes four components:

[1]As the card sorting and taxonomy development were done collaboratively by six engineers in a session, an inter-rater agreement could not be calculated.

TABLE I
OUR TAXONOMY OF THE TYPES OF CODE REVIEW COMMENTS.

Categories	Description
Code Readability	Focuses on the textual readability of code that does not impact functionality. Includes naming conventions, syntax, formatting, code duplication, unused variables, etc. Excludes issues related to code comments or documentation.
Code Bugs	Identified faults in software functionality that caused implementation incorrectness. Includes misuse of language features, logical errors, incorrect variable types, validation issues, resource management, timing problems, and interface interactions.
Maintainability	Addresses issues or suggests changes related to the maintainability of the system and the need for proper code comments and documentation. Includes concerns on hard-coding, fragile code, configuration, best-practices, logging, future-proofing, modifying or adding tests, performance optimization, cyclic dependencies, security, DevOps, version control (e.g., git), etc.
Code Design	Concerns regarding code complexity and structure, appropriate implementation patterns, design principles, and architecture. Includes refactoring, code organization, state management, model structure, null handling, additional programmatic checks, etc.
No Issue	Not directly relevant to code readability, code bugs, maintainability or code design. Focuses on raising open-ended discussions, affirmation, humor, or acknowledgment.
Other	Does not fit into the other categories.

an instructional message, a code review comment, the taxonomy, and an example response. We developed and refined the prompt iteratively based on human-written comments in open source projects [28]. The initial prompt simply instructed the LLM to select the most appropriate category based on the definitions without providing additional context. To enhance performance, we introduced a justification prompt, encouraging the model to explain its classification. This implicitly triggers Chain-of-Thought (CoT) reasoning, which has been shown to improve LLM performance in classification tasks [29]. Including justifications also allows us to evaluate whether the model is reasoning about relevant features or guessing, thereby informing prompt refinement. We further extended the prompt to include a confidence score (ranging from 0 to 1) for each prediction, providing insight into how the model assesses its own ability to classify each individual code review comment.

Sanity Check. To check the soundness of our LLM judge, we evaluated the method on 100 comments generated for our internal projects. The comments were split into two subsets and independently labeled by two annotators over two rounds—near perfect agreement was achieved for each round (Cohen's κ =$\{0.80, 0.86\}$). After each set, they resolved all disagreements with a third annotator acting as an arbiter. Finally, we use the full set of comments with human annotations that were agreed upon to measure inter-rater agreement with the LLM judge—moderate agreement was achieved (Cohen's κ =$\{0.42\}$). Corroborating with past studies [26], we deem the performance of the approach sufficient for evaluating code review comments.

Main Prompt Template

 # Classification
Instructions
You are the best code review comment classifier. Classify the given comment into exactly ONE of the categories listed below and provide a brief justification for your classification. ...

Input
{Code Review Comment}

Categories
{Issue Types and Definitions}

Example response: {
"justification": "This comment points out a specific issue with error handling that could lead to system failures",
"category": "category_name",
"confidence": 0.35
}

Response:

Fig. 2. The Main Prompt Template for Review Comment Classification.

IV. RESULTS

(RQ1) How do the types of code review comments written by humans differ from those generated by an LLM?

Approach. To answer this RQ, we aim to examine differences in the distributions of comment types between LLM-generated and human-written comments. To ensure that the results are not bound to specific contexts, we focus on two software development contexts, i.e., enterprise-graded internal software repositories at Atlassian and open-source software (OSS) projects. For internal projects, we used 300 merged pull requests, containing 465 human-written comments. For OSS projects, we used 1k human-written comments from the CuRev dataset—a curated code review dataset [26]. For each code change, we used RovoDev Agent (via Claude 3.5 Sonnet) to generate code review comments (i.e., an *LLM reviewer*). In total, the LLM reviewer generated 1,256 and 702 comments for open source and Atlassian internal projects, respectively. Note that the total number of generated comments are higher than the total number of human-written comments because the LLM reviewer autonomously generates comments for the pull-request. Then, we used our LLM-as-a-Judge (see Section III-C) to classify

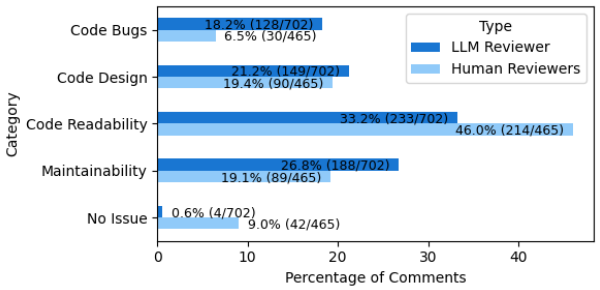

Fig. 3. (RQ1-1) The distribution of human-written vs LLM-Generated code review comment types for Atlassian's internal projects.

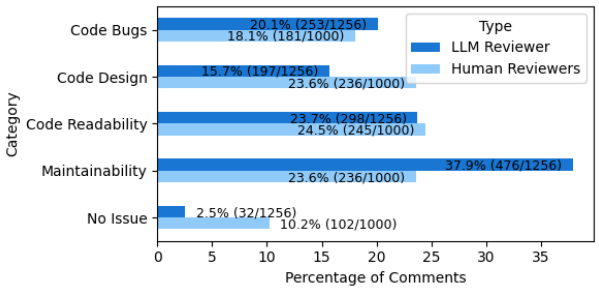

Fig. 4. (RQ1-2) The distribution of human-written vs LLM-Generated code review comment types for OSS projects.

both human-written and LLM-generated comments into our own taxonomy. Figures 3 and 4 present the distribution of comment types by LLM Reviewer and human reviewers for both Atlassian's internal and OSS projects, respectively.

Results. For Atlassian projects, the LLM reviewer generated a higher proportion of comments related to bugs and maintainability, but fewer comments focused on readability compared to human reviewers. When comparing between the LLM reviewer and human reviewers (see Figure 3), we found that, the LLM reviewer generated comments related to code bugs more often than human reviewers did (18.2% [$\frac{128}{702}$] vs. 6.5% [$\frac{30}{465}$]), and also generated more maintainability-related comments (26.8% [$\frac{188}{702}$] for LLM vs. 19.1% [$\frac{89}{465}$] for humans), representing approximately 2.8-fold and 1.4-fold increases, respectively. Conversely, although readability is the most frequently comment type for both groups, human reviewers emphasized it more than the LLM reviewer (46.0% [$\frac{214}{465}$] vs. 33.2% [$\frac{233}{702}$]).

For OSS projects, the LLM reviewer generated more maintainability-related comments but fewer design-related comments than human reviewers did. As shown in Figure 4, maintainability is the most common comment type from the LLM reviewer, comprising 37.9% [$\frac{476}{1,256}$] of its comments, compared to 23.6% [$\frac{236}{1,000}$] from human reviewers. The LLM reviewer generated less no issue comments than human reviewers, with only 0.6% falling into no-issue compared to 9.0% for humans. This could be because developers use code reviews for identifying issues as well as a communication channel to steer the discussion.

These differences across Atlassian's internal and OSS projects highlight that the LLM reviewer and human reviewers exhibit distinct strengths and weaknesses depending on the project context. This is likely because the nature of code review practices vary across different organizations, codebase familiarity, and project goals. For example, in Atlassian's internal projects, the LLM reviewer may be more effective at systematically identifying bugs and maintainability issues due to access to structured guidelines and consistent coding standards, whereas human reviewers might leverage their deeper contextual knowledge to focus on readability and nuanced design concerns. In OSS projects, the diversity of contributors and less standardized practices

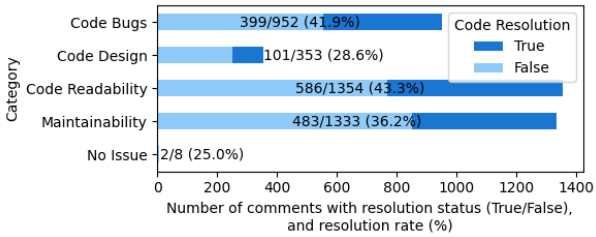

Fig. 5. (RQ2) The percentage of code resolution of the LLM-generated comments for each comment type at Atlassian.

can lead the LLM reviewer to emphasize maintainability, while human reviewers—often more familiar with the project's architecture and community norms—may provide more design-related feedback.

(RQ2) What types of LLM-generated code review comments are frequently resolved by developers?

Approach. To answer this RQ, we aim to examine the association between the comment types and the resolution rate. To examine whether the generated comments are resolved or not, we conduct an online evaluation. Similar to RQ1, we employed the RovoDev Agent in the Atlassian's internal projects. Since code review practices are matured and changed overtime, we opted to focus on the most recent code review comments, spanning across a two-month period (June-July) in 2025. Given a large amount of LLM-generated comments at Atlassian, we randomly selected 4,000 LLM-generated comments that were associated with 3,746 pull requests, spanning across 1,007 repositories. Similar to RQ1, for each comment, we used our LLM Judge (Section III.C) to automatically classify the comment type. A successfully resolved code review comment is determined similar to prior work [15], [30]. We consider *a given code review comment to be resolved* if a subsequent commit modified the exact line where the comment was placed. Then, for each comment type, we measure the code resolution rate as the percentage of the resolved comments to the total of the LLM-generated comments. Finally, we present the percentage of code resolution of the LLM-generated comments for each comment type (see Figure 5).

Results. **Readability, bug, and maintainability-related comments exhibited higher resolution rates compared to those focused on code design.** We observe that comments discussing code readability achieved the highest resolution rate at 43.3% (586/1,354), followed by code bugs at 41.9% (399/952), and maintainability at 36.2% (483/1,333). In contrast, code design had a lower resolution rate of 28.6%.

Nevertheless, many of the LLM-generated comments are not resolved by developers (60%-70%). Such unresolved comments could be due to the lack of clarity, relevancy, and simplicity of the comments. Comments that are less clear or not directly relevant to the code being reviewed are less likely to be acted upon. Additionally, we observed that code design issues often involve more complex, architectural considerations that require deeper understanding and broader changes, making them harder to resolve quickly or automatically. As a result, developers may hesitate to address these suggestions without further discussion or context, leading to lower resolution rates for design-related comments.

V. Discussion

The LLM reviewer and human reviewers are complementary, as they focus on different types of code review comments. Our RQ1 showed that the LLM reviewer and human reviewers exhibited distinct strengths and weaknesses depending on the project context. For instance, in the Atlassian's projects, the LLM reviewer generated a higher proportion of comments related to bugs and maintainability, but fewer comments focused on readability compared to human reviewers. Despite these differences in focus, our RQ2 showed that many of LLM comments were resolved in practices, indicating the value of the LLM comments.

Our results show that LLM reviewers provide different yet acceptable feedback compared to human reviewers. This supports prior work arguing that evaluation should consider comment types rather than relying solely on lexical similarity metrics like BLEU, which may be insufficient [9], [10], [23]–[25]. Future research should explore alternative evaluation aspects, e.g., comment type coverage and actionability, to guide the development of metrics beyond using human-written comments as the sole ground truth.

The LLM reviewer should balance the distribution of comment types and improve the clarity and relevancy. RQ1 showed that the LLM reviewer tends to generate fewer bug-related comments than other types (e.g., 17.8% less than maintainability for OSS projects and 15% less than code readability for Atlassian's internal projects). However, many prior studies [13], [15], [31] showed that code bugs-related comments are among the most useful for human reviewers. Our RQ2 findings support this, showing that bug-related comments have the second-highest resolution rate (5.7% higher than maintainability), indicating that developers are more likely to address them. Nonetheless, we observed that unresolved comments could be due to the lack of clarity, relevancy, and simplicity of the comments. Hence, balancing the distribution of comment types and improving the clarity and relevancy of review comments may improve the effectiveness of LLM-powered code review tools. Future work should address this limitation and explore strategies to increase the practical impact of automated code review.

VI. Threats to the Validity

Threats to Construct Validity may arise from the use of GPT-4.1 as an annotator. The classification of comment types could vary if different large language models (LLMs) are employed. However, we have conducted experiments with other LLMs (e.g., Anthropic Claude), and observed that the results remain consistent.

Threats to Internal Validity may arise from the inclusion of noisy LLM-generated comments in the analysis. To mitigate this, we focused exclusively on comments anchored to specific code lines, as these are likely to be specific and directly related to the code. Since RovoDev Agent is proprietary and its underlying architecture and prompt are confidential, full details cannot be disclosed.

Threats to External Validity concern the generalizability of our findings. Our results are limited to the studied contexts (13 open-source and 1,007 Atlassian's internal projects) and to comments generated by a specific LLM. Additionally, we used one internal LLM reviewer, i.e., RovoDev Agent since the study involved proprietary data (i.e., internal projects). Therefore, the findings may not generalize to other projects, LLMs, or development environments. Future work could explore the use of other LLMs to other projects.

VII. Conclusions

In this paper, we investigate the types of review comments written by humans and LLMs, and the types of LLM comments that are most frequently resolved by developers. Through a study across open-source and Atlassian's internal projects, we found that LLM and human reviewers focus on different types of code review comments, highlighting their complementarity. Comments related to readability, bugs, and maintainability had higher resolution rates than those focused on code design. However, bug-related comments remain underrepresented in LLM comments. To enhance the practical effectiveness of LLM-powered code review tools, future work should 1) explore alternative evaluation aspects beyond using human-written comments as ground truth; 2) improve the balance of comment types, with an increase focus on bug-related comments; and 3) improve the clarity and relevancy of the LLM-generated comments.

Acknowledgment & Disclaimer

We acknowledge the AutoReview (DevAI) team, especially Yaotian Zou, Mohanish Ranade, Andy Wong, Michael Gupta, and Cameron Gregor for their support.

Disclaimer. The perspectives and conclusions presented in this paper are solely the authors' and should not be interpreted as representing the official policies or endorsements of Atlassian or any of its subsidiaries and affiliates. Additionally, the outcomes of this paper are independent of, and should not be constructed as an assessment of, the quality of products offered by Atlassian.

References

[1] A. Bosu, J. C. Carver, C. Bird, J. D. Orbeck, and C. Chockley, "Process aspects and social dynamics of contemporary code review: Insights from open source development and industrial practice at microsoft," *IEEE Trans. Software Eng.*, vol. 43, no. 1, pp. 56–75, 2017. [Online]. Available: https://doi.org/10.1109/TSE.2016.2576451

[2] P. Thongtanunam, R. G. Kula, A. E. C. Cruz, N. Yoshida, and H. Iida, "Improving code review effectiveness through reviewer recommendations," in *Proceedings of CHASE*, 2014, pp. 119–122.

[3] P. Thongtanunam, C. Pornprasit, and C. Tantithamthavorn, "Autotransform: Automated code transformation to support modern code review process," in *Proceedings of the 44th international conference on software engineering*, 2022, pp. 237–248.

[4] P. C. Rigby and C. Bird, "Convergent contemporary software peer review practices," in *Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013*, B. Meyer, L. Baresi, and M. Mezini, Eds. ACM, 2013, pp. 202–212. [Online]. Available: https://doi.org/10.1145/2491411.2491444

[5] S. McIntosh, Y. Kamei, B. Adams, and A. E. Hassan, "An empirical study of the impact of modern code review practices on software quality," *Empir. Softw. Eng.*, vol. 21, no. 5, pp. 2146–2189, 2016. [Online]. Available: https://doi.org/10.1007/s10664-015-9381-9

[6] T. Baum, O. Liskin, K. Niklas, and K. Schneider, "Factors influencing code review processes in industry," in *Software Engineering 2017, Fachtagung des GI-Fachbereichs Softwaretechnik, 21.-24. Februar 2017, Hannover, Deutschland*, ser. LNI, J. Jürjens and K. Schneider, Eds., vol. P-267. GI, 2017, pp. 55–56. [Online]. Available: https://dl.gi.de/handle/20.500.12116/1267

[7] P. W. Gonçalves, E. Fregnan, T. Baum, K. Schneider, and A. Bacchelli, "Do explicit review strategies improve code review performance? towards understanding the role of cognitive load," *Empir. Softw. Eng.*, vol. 27, no. 4, p. 99, 2022. [Online]. Available: https://doi.org/10.1007/s10664-022-10123-8

[8] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu, and N. Sundaresan, "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, ESEC/FSE 2022, Singapore, Singapore, November 14-18, 2022*, A. Roychoudhury, C. Cadar, and M. Kim, Eds. ACM, 2022, pp. 1035–1047. [Online]. Available: https://doi.org/10.1145/3540250.3549081

[9] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, and G. Bavota, "Using pre-trained models to boost code review automation," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 2291–2302. [Online]. Available: https://doi.org/10.1145/3510003.3510621

[10] H. Y. Lin, P. Thongtanunam, C. Treude, and W. Charoenwet, "Improving automated code reviews: Learning from experience," in *Proceedings of the 21st International Conference on Mining Software Repositories*, ser. MSR '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 278–283. [Online]. Available: https://doi.org/10.1145/3643991.3644910

[11] M. Mäntylä and C. Lassenius, "What types of defects are really discovered in code reviews?" *IEEE Trans. Software Eng.*, vol. 35, no. 3, pp. 430–448, 2009. [Online]. Available: https://doi.org/10.1109/TSE.2008.71

[12] M. Beller, A. Bacchelli, A. Zaidman, and E. Jürgens, "Modern code reviews in open-source projects: which problems do they fix?" in *11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India*, P. T. Devanbu, S. Kim, and M. Pinzger, Eds. ACM, 2014, pp. 202–211. [Online]. Available: https://doi.org/10.1145/2597073.2597082

[13] A. K. Turzo and A. Bosu, "What makes a code review useful to opendev developers? an empirical investigation," *Empir. Softw. Eng.*, vol. 29, no. 1, p. 6, 2024. [Online]. Available: https://doi.org/10.1007/s10664-023-10411-x

[14] P. Thongtanunam, S. McIntosh, A. E. Hassan, and H. Iida, "Investigating Code Review Practices in Defective Files: An Empirical Study of the Qt System," in *Proceedings of MSR*, 2015, p. 168–179.

[15] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *12th IEEE/ACM Working Conference on Mining Software Repositories, MSR 2015, Florence, Italy, May 16-17, 2015*, M. D. Penta, M. Pinzger, and R. Robbes, Eds. IEEE Computer Society, 2015, pp. 146–156. [Online]. Available: https://doi.org/10.1109/MSR.2015.21

[16] M. Hasan, A. Iqbal, M. R. U. Islam, A. J. M. I. Rahman, and A. Bosu, "Using a balanced scorecard to identify opportunities to improve code review effectiveness: an industrial experience report," *Empir. Softw. Eng.*, vol. 26, no. 6, p. 129, 2021. [Online]. Available: https://doi.org/10.1007/s10664-021-10038-w

[17] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanykz, and G. Bavota, "Towards automating code review activities," in *Proceedings of ICSE*, 2021, p. 1479–1482.

[18] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, and G. Bavota, "Using pre-trained models to boost code review automation," in *Proceedings of the 44th international conference on software engineering*, 2022, pp. 2291–2302.

[19] R. Tufano, O. Dabić, A. Mastropaolo, M. Ciniselli, and G. Bavota, "Code review automation: strengths and weaknesses of the state of the art," *IEEE Transactions on Software Engineering*, vol. 50, no. 2, pp. 338–353, 2024.

[20] J. Lu, L. Yu, X. Li, L. Yang, and C. Zuo, "Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning," in *Proceedings of ISSRE*, 2023, pp. 647–658.

[21] C. Pornprasit and C. Tantithamthavorn, "Fine-tuning and prompt engineering for large language models-based code review automation," *Information and Software Technology*, vol. 175, p. 107523, 2024.

[22] D. Olewicki, L. Da Silva, S. Mujahid, A. Amini, B. Mah, M. Castelluccio, S. Habchi, F. Khomh, and B. Adams, "Impact of llm-based review comment generation in practice: A mixed open-/closed-source user study," *arXiv preprint arXiv:2411.07091*, 2024.

[23] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, and G. Bavota, "Towards automating code review activities," in *43rd IEEE/ACM International Conference on Software Engineering, ICSE 2021, Madrid, Spain, 22-30 May 2021*. IEEE, 2021, pp. 163–174. [Online]. Available: https://doi.org/10.1109/ICSE43902.2021.00027

[24] R. Tufano, O. Dabic, A. Mastropaolo, M. Ciniselli, and G. Bavota, "Code review automation: Strengths and weaknesses of the state of the art," *IEEE Trans. Software Eng.*, vol. 50, no. 2, pp. 338–353, 2024. [Online]. Available: https://doi.org/10.1109/TSE.2023.3348172

[25] Y. Hong, C. Tantithamthavorn, P. Thongtanunam, and A. Aleti, "Commentfinder: a simpler, faster, more accurate code review comments recommendation," in *Proceedings of ESEC/FSE*, 2022, pp. 507–519.

[26] O. B. Sghaier, M. Weyssow, and H. A. Sahraoui, "Harnessing large language models for curated code reviews," in *22nd IEEE/ACM International Conference on Mining Software Repositories, MSR@ICSE 2025, Ottawa, ON, Canada, April 28-29, 2025*. IEEE, 2025, pp. 187–198. [Online]. Available: https://doi.org/10.1109/MSR66628.2025.00039

[27] W. Takerngsaksiri, J. Pasuksmit, P. Thongtanunam, C. Tantithamthavorn, R. Zhang, F. Jiang, J. Li, E. Cook, K. Chen, and M. Wu, "Human-in-the-loop software development agents," in *Proceedings of the IEEE/ACM International Conference on Software Engineering (ICSE)*, 2025.

[28] N. Davila, I. Nunes, and I. Wiese, "A fine-grained taxonomy of code review feedback in typescript projects," *Empir. Softw. Eng.*, vol. 30, no. 2, p. 53, 2025. [Online]. Available: https://doi.org/10.1007/s10664-024-10604-y

[29] Y. Deng, K. Prasad, R. Fernandez, P. Smolensky, V. Chaudhary, and S. Shieber, "Implicit Chain of Thought Reasoning via Knowledge Distillation," Nov. 2023, arXiv:2311.01460 [cs]. [Online]. Available: http://arxiv.org/abs/2311.01460

[30] M. M. Rahman, C. K. Roy, and R. G. Kula, "Predicting usefulness of code review comments using textual features and developer experience," in *Proceedings of the 14th International Conference on Mining Software Repositories, MSR 2017, Buenos Aires, Argentina, May 20-28, 2017*, J. M. González-Barahona, A. Hindle, and L. Tan, Eds. IEEE Computer Society, 2017, pp. 215–226. [Online]. Available: https://doi.org/10.1109/MSR.2017.17

[31] A. Bosu, M. Greiler, and C. Bird, "Characteristics of useful code reviews: An empirical study at microsoft," in *Proceedings of the 12th Working Conference on Mining Software Repositories*. IEEE, 2015, pp. 146–156.