Review, Review, Review!

Mohamed Hedi Elfkir, Ece Kunduracioglu, Farzad Hallaji Azad, Elyar Esmaeilzadeh

Problem: Iterative Refinement of Code Reviews
Description: Code reviews are fundamental components of the software development lifecycle, providing developers with clear and concise direction for addressing bugs and implementing new features. However, poorly written reviews can significantly hinder developer productivity and progress. Given this motivation, we propose an iterative refinement approach for code reviews that leverages automated quality assessment to systematically enhance review quality and effectiveness.
Target User: Software Teams - Reviewers
Lifecycle Stage: Development (PR Review)
Motivation:

Aspect	**Manual Code Review**	**Automated Code Review**	**References**
Speed & Efficiency	Slow; PR takes 18–30 hrs	Very fast; reviews thousands of lines in minutes	Kodus.io (2024); Bito (2024)
Consistency	Inconsistent; 60% get conflicting feedback	Consistent; applies uniform rules without bias	Kodus.io (2024); Bito (2024); TCM Security (2024)
Scalability	Hard to scale; needs more people	Scalable; handles many PRs and large codebases easily	TCM Security (2024); Aikido Security (2024)
Types of Issues Detected	Complex bugs, logic errors, design flaws, architecture issues, readability, auth issues, crypto flaws	Syntax errors, style issues, common vulnerabilities (SQLi, XSS), performance lags, duplicate code	Sunbytes (2024); Graphite; TCM Security (2024)
Coverage	May miss issues due to reviewer fatigue or oversight; limited by human capacity	Comprehensive; can analyze every line of code including third-party libraries and dependencies	TCM Security (2024); Codiga
Cost	High cost; requires skilled, experienced engineers with years of training	Cost-effective; lower ongoing costs after initial tool setup	Sunbytes (2024); TCM Security (2024); Codegrip (2021)

Methodology: We propose an iterative refinement approach to enhance human-written code reviews using CRScore as a quality assessment and guidance mechanism. The pipeline begins with a human-written review comment for a given code diff. This initial human review is evaluated using CRScore, which provides fine-grained scores across three dimensions—comprehensiveness, relevance, and conciseness—along with detailed feedback identifying potential gaps, missing issues, or areas of improvement. If the CRScore falls below a predefined quality threshold, the human review, along with the original code diff and the CRScore feedback, is passed to a large language model with specific instructions to address the identified deficiencies. The LLM enhances the review by incorporating missing information, removing irrelevant content, or improving clarity while preserving the original intent and valuable insights from the human reviewer. This enhanced review is then re-evaluated using CRScore, and the process repeats iteratively until either the review achieves a score above the threshold or a maximum iteration limit is reached. By grounding the refinement process in CRScore's automated evaluation framework—which combines LLM-generated code claims and static analysis tool outputs—we create a systematic feedback loop that augments human expertise with automated quality improvement. This approach maintains the valuable domain knowledge and contextual understanding of human reviewers while addressing common shortcomings such as incomplete coverage of edge cases, verbosity, or overlooked code smells, ultimately producing higher-quality code reviews that better align with professional standards. (Figure. 1)

Input: A human-written code review comment and its associated code diff. The review includes observations about code quality, bugs, improvements, style, or questions for the author. The diff shows the specific changes made across files.

Output: An improved review that maintains the original insights while addressing quality gaps identified by CRScore. The enhanced version scores higher in comprehensiveness (covering all relevant issues), conciseness (efficient communication without redundancy), and relevance (focusing on pertinent aspects). It preserves the reviewer's expertise while incorporating missing technical details, edge cases, or overlooked best practices.

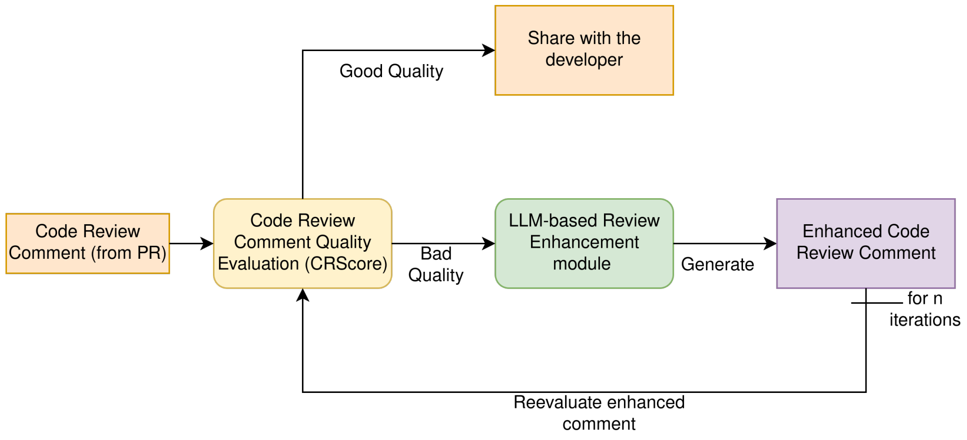
Share with the developer
Good Quality
Code Review Comment (from PR)
Code Review Comment Quality Evaluation (CRScore)
Bad Quality
LLM-based Review Enhancement module
Generate
Enhanced Code Review Comment
for n iterations
Reevaluate enhanced comment

Fig. 1: Our proposed pipeline.

Ground Truth Dataset Availability: RevHelper [1], ChromiumConversations [2], OpenDev [3]
Baselines Available: BM25 Retriever [4], LSTM Reviewer [5], CodeReviewer [6]
Possible novelty: Iterative refinements using LLMs
Possible Tool Support in the Future: Yes, a GitHub bot/Action that pings on new comments, shares scores, and suggests refinements (e.g., "Try this: [text]").
Corner Cases / Scoping: Only works with GitHub projects or with similar structures that have pull requests and reviews. It is mostly useful for teams that have a continuous review process in their development cycle.

References

[1] M. M. Rahman, C. K. Roy, and R. G. Kula, "Predicting usefulness of code review comments using textual features and developer experience," 2018. [Online]. Available: https://arxiv.org/abs/1807.04485
[2] B. S. Meyers, N. Munaiah, E. Prud'hommeaux, A. Meneely, C. Ovesdotter Alm, J. Wolff, and P. K. Murukannaiah, "A dataset for identifying actionable feedback in collaborative software development," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, I. Gurevych and Y. Miyao, Eds. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 126–131. [Online]. Available: https://aclanthology.org/P18-2021/
[3] A. K. Turzo and A. Bosu, "What makes a code review useful to opendev developers? an empirical investigation," 2023. [Online]. Available: https://arxiv.org/abs/2302.11686
[4] S. Robertson and H. Zaragoza, "The probabilistic relevance framework: Bm25 and beyond," *Foundations and Trends in Information Retrieval*, vol. 3, pp. 333–389, 01 2009.
[5] A. Gupta, "Intelligent code reviews using deep learning," 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:52219239
[6] 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," 2022. [Online]. Available: https://arxiv.org/abs/2203.09095