Beyond More Context: Evaluating Static and Agent-Based Repository Context in LLM-Based Code Review

Jonas Bernardino de Lima
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
João Pessoa, Paraíba, Brazil
jonas.bernardino@academico.ifpb.edu.br

Danyllo Albuquerque
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
João Pessoa, Paraíba, Brazil
danyllo@copin.ufcg.edu.br

Emanuel Dantas Filho
Instituto Federal de Educação, Ciência e Tecnologia de Pernambuco
Jaboatão dos Guararapes
Pernambuco, Brazil
emanuel.filho@jaboatao.ifpe.edu.br

Felipe Ramos
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
João Pessoa, Paraíba, Brazil
felipe.ramos@ifpb.edu.br

Juliana Medeiros
Instituto Federal de Educação, Ciência e Tecnologia da Paraíba
João Pessoa, Paraíba, Brazil
juliana.medeiros@ifpb.edu.br

Abstract

Large Language Models (LLMs) have been explored for automated code review tasks, but it remains unclear whether their performance benefits from simply adding more context or from how such context is accessed. This paper investigates repository context in LLM-based automated code review by comparing three conditions: the original static ContextCRBench condition, static concatenation of additional repository context, and agent-based access to repository context through tools. We evaluate these conditions on the merge-decision task using a balanced subset of 144 pull requests and analyze predictive performance, response validity, latency, and tool-call traceability. Results show that richer benchmark-provided context achieved higher F1-score than minimal context. However, neither static concatenation nor agent-based repository access outperformed the corresponding benchmark condition, while repository-aware strategies increased latency. Although agent-based access did not improve predictive performance, it enabled inspectable repository access through tool-call logs. These findings suggest that effective context use in LLM-based code review depends less on simply providing additional information and more on its relevance, access strategy, and operational trade-offs.

Keywords

Large Language Models for Software Engineering, Automated Code Review, Pull Requests, Repository Context, AI Agents, Software Analytics, AI4SE

1 Introduction

Code review is a central practice in modern software engineering, particularly in collaborative workflows based on Pull Requests (PRs). Reviewers inspect proposed changes to identify defects, discuss design decisions, disseminate knowledge, and maintain coding and architectural conventions [2, 4, 5, 9]. Beyond defect detection, code review is a context-dependent activity in which reviewers must understand the intent and impact of a change within the broader software project.

A PR diff alone may be insufficient for this task. Reviewers often rely on issue descriptions, related files, documentation, dependencies, change history, and repository conventions [3, 6, 8, 12]. This creates an information-access problem, as relevant evidence is distributed across the repository and associated artifacts.

Recent advances in Large Language Models (LLMs) have motivated their use in automated code review tasks [1, 11]. However, LLM-based review systems still face limitations related to precision, reliability, hallucinations, and insufficient understanding of project-specific intent. Addressing these limitations requires determining not only which contextual information should be provided, but also how models should access it during inference.

Context-aware benchmarks such as ContextCRBench evaluate automated code review under different levels of enriched context [7]. However, benchmark-provided context is primarily presented as static input, whereas repository knowledge is broader and distributed across multiple artifacts. Additional repository context can be concatenated directly into the prompt, which is simple but may introduce irrelevant information and increase inference cost. Alternatively, tool-mediated access allows an agent to consult repository information on demand, potentially supporting selective and inspectable access at the cost of additional complexity and latency.

Despite growing interest in context-aware LLMs and repository-level agents, limited empirical evidence compares these strategies in automated code review. It remains unclear whether additional repository context improves performance, whether tool-mediated access is more effective than static concatenation, and what operational trade-offs these strategies introduce. Addressing this gap is relevant to Intelligent Software Engineering because repository-aware LLM systems depend not only on model capabilities, but also on mechanisms for accessing and tracing external evidence.

To investigate this gap, we conduct an empirical study based on ContextCRBench comparing the original static benchmark condition, static concatenation of additional repository context, and agent-based access to the same context. We evaluate these conditions on the merge-decision task using a balanced subset of 144 PRs and analyze predictive performance, response validity, latency, and tool-call traceability.

The results show that C8 achieved higher F1-scores than C1 across all strategy groups, whereas additional repository context did not outperform the corresponding benchmark conditions. Repository-aware strategies also increased latency, while agent-based access produced inspectable tool-call logs. These findings indicate that the effectiveness of repository context depends on its relevance, access strategy, and operational trade-offs.

This paper makes three contributions. First, it compares static concatenation and agent-based access to repository context in LLM-based automated code review. Second, it provides a ContextCRBench-based evaluation setup integrating repository context, tool-mediated access, and traceability. Third, it provides evidence that additional repository context does not necessarily improve merge-decision performance, motivating more selective and task-oriented context mechanisms.

2 Background And Related Work

Code review evaluates changes before their integration into the main codebase. In PR-based workflows, reviewers assess correctness, behavioral impact, maintainability, adherence to project standards, and potential risks [2, 5]. Since the diff may not provide sufficient evidence, reviewers often consult related artifacts, project constraints, and repository conventions [4, 9, 12]. For LLM-based review, context selection is therefore critical: insufficient context may omit relevant evidence, whereas excessive context may introduce noise and increase cost.

LLMs have supported software engineering tasks such as code generation, bug fixing, program comprehension, and code review. Prior work has investigated review comment generation, defect localization, and PR assessment. CodeReviewer explores pre-trained models for code review, while AICodeReview evaluates LLM-based support for review activities [1, 11, 14]. Although these studies demonstrate the feasibility of LLM-based review, they also report limitations such as generic feedback, incomplete reasoning, and misalignment with change intent.

ContextCRBench evaluates merge decision, problematic line localization, and review comment generation through eight predefined context configurations (C1–C8), constructed from different combinations of pull-request and code information [7]. C1 contains only the changed diff hunk, whereas C8 combines all context components available in the benchmark. This study preserves the benchmark's task definition, formats, and evaluation logic. While Retrieval-Augmented Generation (RAG) incorporates external information during generation [10], agent-based approaches allow models to actively access such information through tools. ReAct and Toolformer demonstrate the integration of reasoning and tool use [13, 17]; SWE-agent applies this approach to repository-level bug fixing [16]; and RepoCoder uses iterative retrieval for repository-level code completion [18]. However, limited empirical evidence compares agent-based repository access with static context concatenation for PR-based code review. This study addresses this gap through a controlled comparison.

3 Study Design

This study investigates whether additional repository context improves LLM-based automated code review and whether different context-access strategies affect predictive performance, latency, and traceability. We compare three experimental conditions while keeping the benchmark, task, dataset subset, model configuration, and evaluation metrics fixed across conditions. The study is guided by three research questions (Section 3.1) and follows a controlled experimental procedure (Section 3.2) comprising dataset and repository-context preparation, model inference, and the analysis of predictive and operational outcomes. The artifacts supporting the study and its replication are available in the Artifact Availability section.

3.1 Research Questions

The study addresses three research questions:

RQ1. Does statically concatenated repository context improve LLM performance over the corresponding ContextCRBench condition?

RQ2. Does agent-based repository access improve LLM performance over static concatenation and the corresponding benchmark condition?

RQ3. What are the operational costs and traceability implications of the evaluated strategies regarding response validity, latency, and tool-call logging?

RQ1 examines context availability, RQ2 compares context-access strategies, and RQ3 evaluates their operational implications.

3.2 Experimental Procedure

Figure 1 summarizes the experimental workflow. Each item is evaluated under three conditions: the original benchmark (*Static-Ck*), static repository-context concatenation (*Static-Ck+Repo*), and tool-mediated repository access (*Agent-Ck+Repo*). Inference produces structured JSON outputs and, for the agent condition, tool-call logs.

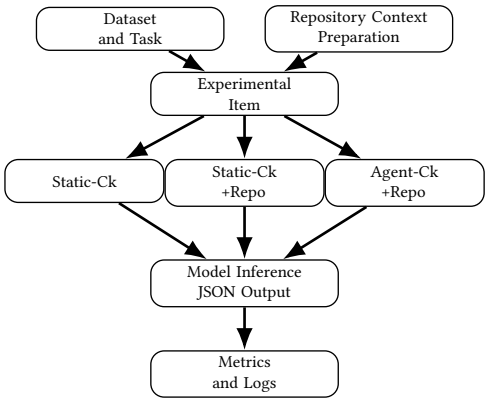

Figure 1: Methodological workflow.

Dataset and Task. We evaluate ContextCRBench Task 1, in which the model predicts whether a pull request (PR) should be merged. This binary outcome enables evaluation using standard classification metrics.

The balanced *task1_mini_144* subset comprises 144 PRs: 72 merged and 72 unmerged. It covers four temporal groups (2021, 2022, 2023, and 2024+) and nine languages: C, C#, C++, Go, Java, JavaScript, Python, Rust, and TypeScript. Each temporal group–language combination contains four items, two merged and two unmerged.

Benchmark Context Levels. A context level represents a predefined combination of information about the pull request and the affected code. ContextCRBench defines eight levels (C1–C8) from five context components: the changed diff hunk (*Base*), issue description (*I*), pull-request description (*P*), code before the change (*B*), and code after the change (*A*). These levels are not strictly cumulative; instead, they combine different subsets of these components.

C1 contains only the changed diff hunk and represents the minimum benchmark context. C2 adds the issue description to the diff, whereas C3 adds the pull-request description. C4 combines the diff with both the issue and pull-request descriptions. The code-oriented levels provide the diff together with the code before the change (C5), the code after the change (C6), or both versions (C7). Finally, C8 combines all five components: the diff, issue description, pull-request description, and code before and after the change, representing the most comprehensive benchmark context.

We evaluate C1 and C8 as the two boundary configurations to examine the effect of repository-level context when the model receives either the minimum or the most comprehensive information available in the benchmark. Intermediate levels (C2–C7) were excluded to keep the experimental scope and computational cost manageable. Therefore, our findings should not be generalized to those intermediate context configurations.

Repository Context Preparation. Each repository was associated with a semi-structured Markdown file describing its structure, technologies, architectural conventions, review guidelines, and testing expectations. The files contain general repository knowledge and exclude PR-specific labels and post-hoc information. They were normalized, mapped from *owner/repo* to *owner_repo.md*, and identified by a SHA-256 hash to ensure version traceability.

Context-Access Conditions. We evaluate three conditions:

- *Static-Ck*: uses the original benchmark prompt and Ck context;
- *Static-Ck+Repo*: concatenates repository context to the benchmark prompt; and
- *Agent-Ck+Repo*: preserves the benchmark prompt and provides on-demand repository access through predefined tools.

For the agent condition, the system constructs the same base prompt used by the static conditions and exposes repository context through tools that may be invoked before the model produces the required JSON output. Each tool call records the execution and item identifiers, task, context level, repository, tool name, input, output preview and size, execution time, timestamp, and content hash. These logs show which information was consulted but not whether it improved the prediction.

Model Inference and JSON Output. All conditions used *GPT-5.4* at temperature 0 with identical prompt templates, maximum output sizes, and expected JSON schemas. Each item was executed once as an initial controlled comparison rather than an estimate of model variance. The model was instructed to return only the JSON required by the merge-decision task; non-parseable or schema-nonconforming responses were classified as invalid. Temperature 0 reduces but does not eliminate nondeterminism.

Metrics and Logs. Predictive performance is evaluated using accuracy, precision, recall, and F1-score. Operational analysis considers valid response rate, average latency per item, and, for the agent condition, tool-call traceability.

4 Results

This section presents the results for the merge-decision task using the *task1-mini-144* subset. The analysis follows the three RQs. We first examine the effect of statically concatenating additional repository context, then compare agent-based repository access with the static conditions, and finally analyze response validity, latency, and tool-call traceability. Detailed results and supporting experimental data are available in the Artifact Availability section.

4.1 Effect Of Static Repository Context (RQ1)

Table 1 summarizes the predictive performance of the evaluated conditions. The highest F1-score was achieved by *Static-C8*, at 57.14%, followed by *Static-C8+Repo*, at 52.98%, and *Static-C1*, at 52.90%. The lowest F1-score was obtained by *Static-C1+Repo*, at 39.02%.

Table 1: Performance results for Task 1 on *task1_mini_144*.

Condition	Ctx	Acc.	Prec.	Rec.	F1
Agent-C1+Repo	C1	53.47	56.41	30.56	39.64
Static-C1	C1	49.31	49.40	56.94	52.90
Static-C1+Repo	C1	47.92	47.06	33.33	39.02
Agent-C8+Repo	C8	52.78	53.33	44.44	48.48
Static-C8	C8	52.08	51.69	63.89	57.14
Static-C8+Repo	C8	50.69	50.63	55.56	52.98

Comparing the two original benchmark conditions, *Static-C8* achieved an F1-score 4.24 percentage points higher than *Static-C1* (57.14% versus 52.90%). Higher F1-scores at C8 were also observed for the other strategy groups: *Static-C8+Repo* exceeded *Static-C1+Repo* by 13.96 percentage points, while *Agent-C8+Repo* exceeded *Agent-C1+Repo* by 8.84 percentage points. Thus, all three groups achieved higher F1-scores under C8 than under C1.

However, statically concatenating additional repository context did not improve performance over the corresponding original benchmark condition. At C1, the F1-score decreased from 52.90% for *Static-C1* to 39.02% for *Static-C1+Repo*, a difference of 13.88 percentage points. At C8, the decrease was smaller, from 57.14% to 52.98%, corresponding to 4.16 percentage points.

The metric-level results also show that these differences were primarily associated with recall. At C1, adding repository context reduced recall from 56.94% to 33.33%, while precision decreased from 49.40% to 47.06%. At C8, recall decreased from 63.89% to 55.56%, whereas precision changed only slightly, from 51.69% to 50.63%. Therefore, in both context levels, the lower F1-score of static repository augmentation was accompanied mainly by a reduction in the proportion of positive instances correctly identified.

Answer to RQ1. Statically concatenated repository context did not improve predictive performance over the corresponding benchmark condition. The F1-score decreased by 13.88 percentage points at C1 and 4.16 percentage points at C8, with the reductions primarily associated with lower recall.

4.2 Effect Of Agent-Based Repository Access (RQ2)

The agent-based condition also did not outperform the corresponding static benchmark condition. At C1, *Agent-C1+Repo* achieved an F1-score of 39.64%, compared with 52.90% for *Static-C1*, a difference of 13.26 percentage points. At C8, *Agent-C8+Repo* achieved 48.48%, compared with 57.14% for *Static-C8*, a difference of 8.66 percentage points.

The comparison between the two repository-aware strategies shows different results across context levels. At C1, agent-based access achieved an F1-score of 39.64%, marginally higher than the 39.02% obtained with static concatenation. At C8, however, static concatenation achieved 52.98%, exceeding the agent-based condition at 48.48%. Thus, agent-based repository access did not consistently outperform static concatenation.

A closer examination of precision and recall reveals additional differences between the strategies. At C1, the agent-based condition achieved the highest precision among all evaluated conditions (56.41%), but also the lowest recall (30.56%). At C8, agent-based access achieved slightly higher precision than the corresponding static benchmark condition (53.33% versus 51.69%), but substantially lower recall (44.44% versus 63.89%). Consequently, the agent-based conditions produced more conservative predictions, with gains or small differences in precision accompanied by substantial reductions in recall.

These results show that providing tool-mediated access to repository information did not translate into higher overall predictive performance. The observed precision–recall differences further indicate that aggregate metrics alone do not fully characterize the effects of the evaluated context-access strategies.

Answer to RQ2. Agent-based repository access did not outperform the corresponding static benchmark conditions and did not consistently improve over static concatenation. Although the agent-based conditions achieved comparatively high precision, this was accompanied by lower recall and, consequently, lower F1-scores.

4.3 Operational Results And Tool-Call Traceability (RQ3)

All evaluated conditions produced valid JSON responses for all 144 items, with no invalid outputs. Thus, differences in predictive performance were not associated with failures to generate benchmark-compliant responses. Table 2 presents the operational results. The lowest average latency was observed for *Static-C1*, at 4.29 seconds, while the highest occurred for *Agent-C8+Repo*, at 6.43 seconds.

Table 2: Operational results for Task 1 on *task1_mini_144*.

Condition	Total	Valid	Invalid	Latency (s)
Agent-C1+Repo	144	144	0	5.35
Static-C1	144	144	0	4.29
Static-C1+Repo	144	144	0	5.03
Agent-C8+Repo	144	144	0	6.43
Static-C8	144	144	0	5.08
Static-C8+Repo	144	144	0	5.51

At C1, static repository concatenation increased average latency from 4.29 to 5.03 seconds, an increase of approximately 17.2%, while agent-based access increased it to 5.35 seconds, approximately 24.7% above the corresponding static benchmark condition. At C8, static concatenation increased average latency from 5.08 to 5.51 seconds (8.5%), whereas agent-based access increased it to 6.43 seconds (26.6%). Agent-based access therefore exhibited the highest latency at both context levels.

The agent-based condition also produced tool-call logs during inference. For each invocation, the logs record the repository, tool name, input, output preview and size, execution time, timestamp, and hash of the returned content. These records make it possible to inspect which repository information was accessed during inference and provide an operational distinction between agent-based access and static prompt augmentation.

In this study, tool-call logs are treated as traceability evidence rather than evidence of improved correctness. The predictive results show that inspectable repository access did not translate into higher F1-scores. Nevertheless, the logs provide information unavailable in the static conditions by exposing the repository-access operations performed during inference.

Answer to RQ3. All conditions produced valid outputs, while repository-aware strategies increased average latency. Agent-based access introduced the highest latency at both context levels, approximately 24.7% above *Static-C1* and 26.6% above *Static-C8*, while providing tool-call logs that made repository access inspectable.

5 Discussion

The results should be interpreted in light of prior work on context-aware code review, LLM-based review support, and tool-mediated repository access. Overall, our findings reinforce the importance of context in automated code review, but also show that context is not uniformly beneficial.

The higher F1-scores observed at C8 across all strategy groups are consistent with prior evidence that code review is a context-dependent activity. Studies on modern code review have shown that reviewers often need information beyond the diff, including rationale, related artifacts, project conventions, and constraints [2, 4, 9, 12]. However, our results also show that not every additional source of context helps. In our setting, the repository-level Markdown files provided general project knowledge, but may not have contained the localized evidence required to decide whether a specific PR should be merged.

The findings extend the discussion introduced by ContextCR-Bench [7] by distinguishing benchmark-provided task context from additional repository-level context and comparing static concatenation with agent-based access. The results suggest that context should not be treated simply as an amount of information provided to the model. Its usefulness also depends on its alignment with the target decision. This is particularly relevant for merge decisions, which may require specific evidence about changed files, linked issues, tests, dependencies, or localized design constraints.

Our results also relate to prior work on LLM-based automated code review, such as CodeReviewer and AICodeReview [1, 11]. These studies demonstrate the feasibility of LLM-based review

support while highlighting limitations such as generic feedback, incomplete reasoning, and misalignment with project-specific intent. The negative results observed in our repository-aware conditions reinforce the need for more selective and task-oriented context mechanisms. Simply exposing broader project information did not improve merge-decision performance.

Finally, the agent-based results should be considered in relation to prior work on RAG, tool use, and repository-level agents [10, 13, 16–18]. Although these approaches demonstrate the potential of consulting external information during inference, tool-mediated access did not improve F1-score in our setting and increased latency. Tool availability alone is therefore insufficient: an agent must identify relevant information and effectively incorporate it into the final decision. At the same time, tool-call logs made repository access inspectable, providing traceability even without improvements in predictive performance.

6 Implications

The findings of this study have implications for researchers investigating context-aware LLM systems, designers of repository-aware code review approaches, and practitioners adopting these technologies. Across these perspectives, the results suggest that the value of context depends on its availability and relevance to the target task, the strategy used to access it, and the operational trade-offs involved.

Implications for Research. For research on LLM-based software engineering, context should be modeled as a structured experimental variable rather than as a generic prompt extension. Our results distinguish three dimensions that are often conflated: the amount of available information, its relevance to the task, and the mechanism used to access it. The higher performance of C8 over C1, combined with the lack of improvement from additional repository context, suggests that future studies should evaluate not only whether context is available, but also whether it is task-aligned.

This distinction is particularly important for benchmark design. Repository-aware code review benchmarks should distinguish among different context sources, such as architectural guidelines, contribution rules, testing documentation, dependency information, related files, linked issues, and PR-specific evidence. Finer-grained ablations and relevance assessments could help determine which information actually supports model decisions. Agent-based approaches should also be evaluated beyond final task performance. Tool-call logs enable analyses of query specificity, retrieved-context relevance, tool-use frequency, and their relationship with prediction correctness, helping determine whether agents perform targeted repository inspection or merely introduce additional overhead.

Design Lessons for Repository-Aware LLM Review. The findings suggest three main design lessons. First, repository context should be treated as evidence to be selected rather than merely as text to be appended. General project documentation may support the understanding of repository conventions, but merge decisions often require localized evidence, such as affected files, related tests, linked issues, dependency constraints, or prior changes. Second, agent-based access should be evaluated according to the relevance of retrieved evidence, not only final prediction metrics. Tool-call logs can reveal whether the model consulted relevant information or retrieved content unrelated to the target PR. Third, repository-aware systems should expose the relationship between recommendations and supporting evidence. Otherwise, additional context may increase cost and complexity without improving decision quality or reviewer trust.

Implications for Industry. For practitioners and tool builders, the findings suggest caution when incorporating repository context into automated code review systems. Static concatenation is simple and reproducible, but may increase prompt size, latency, and exposure to irrelevant information. Agent-based access provides greater inspectability, but also introduces execution complexity and operational cost. Additional context access should therefore be justified through measurable benefits, such as improved performance, better explanations, stronger auditability, or increased reviewer trust.

In industrial settings, the value of agent-based review may extend beyond immediate performance improvements. Tool-call logs can help developers inspect which repository information was consulted, debug incorrect decisions, identify irrelevant retrieved context, and refine retrieval policies. Overall, repository-aware code review should be designed as a selective evidence-gathering process rather than as a strategy of simply providing the model with more information.

7 Threats To Validity

This section discusses the main threats to the validity of the study, organized according to the categories proposed by Wohlin et al. [15].

Construct Validity. Accuracy, precision, recall, and F1-score are suitable for evaluating predictive performance in the binary *merge* decision task, but they capture only one dimension of code review quality. They do not assess aspects such as explanation usefulness, recommendation actionability, reviewer trust, or the quality of the evidence supporting a decision. Similarly, tool-call logging provides evidence that repository information was accessed, but does not demonstrate that the retrieved information was relevant to or effectively used in the final prediction. In addition, the relevance of repository context was not manually assessed for each Pull Request. Consequently, some context files may have contained general project information with limited relevance to the specific changes under review. The merged/unmerged label represents an observed project outcome rather than an objective measure of technical review quality. A pull request may remain unmerged because of abandonment, duplication, timing, shifting priorities, or project-management decisions. Therefore, the results concern the prediction of historical merge outcomes and should not be interpreted as determining whether a pull request technically deserved acceptance.

Internal Validity. Potential confounding factors include prompt formulation, model nondeterminism, execution parameters, and variations in agent tool-use behavior. To mitigate these threats, the same dataset subset, task definition, model configuration, output format, and evaluation metrics were maintained across conditions. The repository-aware conditions were also derived from the corresponding static benchmark conditions, reducing differences unrelated to context access. Nevertheless, each item was executed only once,

and temperature zero does not eliminate all sources of nondeterminism in LLM inference. Moreover, the agent autonomously decided whether and how to use the available tools, meaning that variations in retrieval behavior may have affected individual predictions.

External Validity. This study considered only the merge-decision task (Task 1) of ContextCRBench and used the balanced *task1-mini-144* subset. The evaluation was also conducted using a single LLM and two benchmark context levels, C1 and C8. Therefore, the findings may not generalize to problematic line localization, review comment generation, other LLMs, repositories, programming languages, context levels, repository-context construction strategies, or industrial code review settings. Furthermore, the repository context consisted primarily of general project documentation, whereas other context sources, such as related files, linked issues, test results, dependencies, and change history, may lead to different outcomes.

Conclusion Validity. The analysis is descriptive and based on 144 Pull Request items, with one execution per item and no statistical significance or effect-size analysis. Therefore, the reported differences should be interpreted as empirical observations within the evaluated setting rather than evidence of the statistical superiority of any context-access strategy. Although the balanced subset supports direct comparison across conditions, repeated runs and larger samples are needed to estimate output variability and assess the stability of the observed trends. Future work should complement predictive metrics with statistical analyses and qualitative inspection of errors and tool-call logs to better understand when additional repository context contributes to, has no effect on, or negatively affects model decisions.

8 Final Remarks

This paper investigated whether additional repository context improves LLM-based automated code review and whether different context-access strategies affect predictive performance and operational aspects. Using ContextCRBench as the experimental foundation, we compared the original static benchmark condition, static concatenation of additional repository context, and agent-based access to the same context. The evaluation focused on the merge-decision task using a balanced subset of 144 PRs.

The results show that C8 achieved higher F1-scores than C1 across all strategy groups, whereas additional repository context did not outperform the corresponding benchmark conditions. Neither static concatenation nor agent-based access improved predictive performance, and both repository-aware strategies increased latency. Although agent-based access did not improve F1-score, it produced tool-call logs that made repository access inspectable.

These findings indicate that effective context use in LLM-based code review depends not simply on providing more information, but on selecting task-relevant evidence and determining how it should be accessed. For researchers, the results highlight the need to evaluate context relevance and access mechanisms separately. For tool designers and practitioners, they suggest that the additional complexity and cost of repository-aware strategies should be justified by measurable benefits in performance, traceability, or other dimensions of review quality.

Future work will extend the evaluation to additional ContextCRBench tasks, including problematic line localization and review comment generation, as well as additional LLMs, larger datasets, and repeated executions. Further studies should also investigate finer-grained repository-context sources and analyze tool-call logs to determine when repository information is relevant, ignored, or introduces noise into model decisions.

Artifact Availability

The supplementary material, including the experimental artifacts and resources supporting the replication of this study, is publicly available at: [1].

Acknowledgements

Generative AI tools were used to support language revision and improve manuscript clarity. All technical content, experimental decisions, interpretations, and final text were reviewed and approved by the authors, who take full responsibility for the manuscript.

References

[1] Y. Almeida, D. Albuquerque, E. Dantas Filho, F. Muniz, K. de Farias Santos, M. Perkusich, H. Almeida, and A. Perkusich. 2024. AICodeReview: Advancing Code Quality with AI-Enhanced Reviews. *SoftwareX* 26 (2024), 101677.
[2] A. Bacchelli and C. Bird. 2013. Expectations, Outcomes, and Challenges of Modern Code Review. In *ICSE*. 712–721.
[3] K. Baciejowski et al. 2023. Are Code Review Smells and Metrics Useful in Pull Request-Level Software Defect Prediction? In *Developments in Information and Knowledge Management Systems for Business Applications*. 27–52.
[4] D. Badampudi, M. Unterkalmsteiner, and R. Britto. 2023. Modern Code Reviews: Survey of Literature and Practice. *ACM Transactions on Software Engineering and Methodology* 32, 4 (2023), 107:1–107:61.
[5] M. Beller, A. Bacchelli, A. Zaidman, and E. Juergens. 2014. Modern Code Reviews in Open-Source Projects: Which Problems Do They Fix?. In *MSR*. 202–211.
[6] E. Dogan and E. Tüzün. 2022. Towards a Taxonomy of Code Review Smells. *Information and Software Technology* 142 (2022), 106737.
[7] R. Hu, X. Wang, X.-C. Wen, Z. Zhang, B. Jiang, P. Gao, C. Peng, and C. Gao. 2025. Benchmarking LLMs for Fine-Grained Code Review with Enriched Context in Practice. *arXiv preprint arXiv:2511.07017* (2025).
[8] F. Khoshnoud et al. 2022. Which Bugs Are Missed in Code Reviews: An Empirical Study on SmartSHARK Dataset. *CoRR* abs/2205.09428 (2022).
[9] O. Kononenko, O. Baysal, and M. W. Godfrey. 2016. Code Review Quality: How Developers See It. In *ICSE*. 1028–1038.
[10] P. Lewis et al. 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. In *NeurIPS*, Vol. 33. 9459–9474.
[11] Z. Li et al. 2022. Automating Code Review Activities by Large-Scale Pre-training. In *ESEC/FSE*.
[12] L. Pascarella, D. Spadini, F. Palomba, M. Bruntink, and A. Bacchelli. 2018. Information Needs in Contemporary Code Review. *Proceedings of the ACM on Human-Computer Interaction* 2, CSCW (2018), 135:1–135:27.
[13] T. Schick et al. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. In *NeurIPS*, Vol. 36.
[14] T. Sun et al. 2025. BitsAI-CR: Automated Code Review via LLM in Practice. In *Proceedings of the ACM International Conference on the Foundations of Software Engineering*. 274–285.
[15] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén. 2012. *Experimentation in Software Engineering*. Springer.
[16] J. Yang, C. E. Jimenez, A. Wettig, K. Lieret, S. Yao, K. Narasimhan, and O. Press. 2024. SWE-agent: Agent-Computer Interfaces Enable Automated Software Engineering. In *NeurIPS*, Vol. 37. 50528–50652.
[17] S. Yao et al. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *ICLR*.
[18] F. Zhang et al. 2023. RepoCoder: Repository-Level Code Completion Through Iterative Retrieval and Generation. In *EMNLP*. 2471–2484.

[1] https://github.com/llmcodereview-cmyk/supplementary-material