AUTOMATING CODE REVIEWS WITH CONTEXT: A RETRIEVAL AUGMENTED METHOD

Nguyen Thi Lan[1]

[1]Hanoi School of Artificial Intelligence, VIETNAM

ABSTRACT

Modern software development depends heavily on code review to maintain software quality, detect defects at an early stage, and ensure that coding standards are consistently followed across large and evolving projects. Although manual review remains one of the most reliable quality assurance practices, it is also time-consuming, cognitively demanding, and difficult to scale in contemporary development environments where teams continuously integrate and deploy new code. This study proposes an automated code review framework for Python 3.13 projects that combines Large Language Models with Retrieval-Augmented Generation (RAG) to generate more accurate, context-sensitive, and actionable review comments. The proposed framework first constructs a dataset from GitHub pull requests collected through the GitHub REST API (version 2022-11-28). Review comments are then organized into semantic categories using a semi-supervised Support Vector Machine (SVM) classifier. During inference, the system retrieves the most relevant historical comments from a vector database and provides them as contextual guidance for multiple open-weight language models, including DeepSeek-Coder-33B, Qwen2.5-Coder-32B, Codestral-22B, CodeLlama-13B, Mistral-Instruct-7B, and Phi-3-Mini. The performance of the framework is evaluated through a comprehensive validation methodology that combines conventional text-generation metrics such as BLEU-4, ROUGE-L, and cosine similarity with semantic evaluation using an LLM-as-a-Judge approach and expert human assessment. Experimental findings demonstrate that retrieval augmentation significantly improves review quality for larger models. DeepSeek-Coder, for instance, achieved a 17.9% improvement in alignment score when retrieval depth was set to k = 3. Smaller models such as Phi-3-Mini, however, experienced context collapse when excessive contextual information reduced the quality of generated feedback. To address this limitation, a hybrid expert-routing mechanism was developed to dynamically assign review tasks to the most suitable model according to semantic category. The proposed system achieved an overall improvement of 13.2% over zero-shot baselines while also reducing hallucinations and generating review comments that closely resemble expert human feedback.

KEYWORDS: Automating Code Reviews; Context; Retrieval-Augmented

INTRODUCTION

Code review has become one of the most important practices in contemporary software engineering because it enables developers to detect faults, improve maintainability, enforce coding standards, and prevent vulnerabilities from reaching production systems. In modern development environments, peer review is not merely a quality-control activity but a central process that supports collaboration, knowledge transfer, and long-term software sustainability. Empirical studies conducted at major technology organizations, including Google, have demonstrated that systematic review practices significantly improve software reliability and maintainability over time.

Despite its importance, manual code review presents major practical limitations. As software systems become increasingly large and complex, developers face growing cognitive pressure during the review process. Engineers are often required to inspect extensive code modifications while simultaneously understanding project requirements, implementation details, and architectural implications. Research has shown that developers spend several hours each week performing review-related activities, reducing the time available for feature development and innovation. In addition to the workload problem, manual review quality is strongly influenced by reviewer expertise, domain familiarity, and time constraints. Consequently, subtle logical flaws, maintainability concerns, or security weaknesses may remain undetected, particularly when review practices are inconsistent across teams.

To reduce these limitations, the software industry has long relied on Static Application Security Testing (SAST) tools such as SonarQube and ESLint. These tools automatically identify syntax violations, style inconsistencies, and predefined vulnerability patterns. Although effective for rule-based analysis, traditional static analyzers lack a deeper semantic understanding of source code and business logic. They are unable to interpret project-specific conventions, architectural intent, or developer rationale, which limits their ability to provide meaningful contextual feedback.

Recent advances in Large Language Models have created new possibilities for automating software engineering tasks. Modern code-oriented LLMs can summarize code, explain functionality, generate implementations, and identify defects using natural language reasoning. Nevertheless, applying LLMs directly to automated code review remains challenging because these models generally operate without awareness of project history, organizational conventions, or prior review discussions. In many situations, they produce overly generic recommendations or hallucinated suggestions that appear plausible but are technically incorrect or irrelevant to the project context.

To address these challenges, this study introduces a Retrieval-Augmented Generation framework for automated Python code review. Unlike traditional retrieval systems that treat all code modifications equally, the proposed approach uses a two-stage

architecture that incorporates semantic classification and dynamic expert routing. First, pull requests are classified into semantic categories such as Functional, Refactoring, Documentation, and Discussion in order to determine the reviewer's underlying objective. Next, the system retrieves historically relevant review comments from a vector database and supplies them as contextual guidance to a specialized language model selected according to the semantic category.

The framework employs multiple open-weight language models, each selected because of its strengths in specific review tasks. Larger models such as Qwen2.5-Coder and DeepSeek-Coder are utilized for complex functional analysis and defect identification, while smaller instruction-oriented models are used for lighter tasks such as documentation and readability suggestions. This dynamic routing strategy enables the system to generate more precise, coherent, and context-aware review comments while simultaneously reducing computational overhead.

The primary objective of this research is to evaluate whether retrieval augmentation and expert model routing can improve the quality of automated code review compared to conventional zero-shot language-model approaches. The study specifically investigates the influence of contextual retrieval depth, the specialization capabilities of different LLM architectures, and the effectiveness of combining retrieval-based reasoning with semantic task allocation.

This research addresses several key questions concerning the future of automated software review systems. The first objective is to determine the extent to which Retrieval-Augmented Generation improves the specificity and technical correctness of generated review comments compared to baseline LLM outputs. The second objective is to identify the optimal retrieval depth that balances contextual awareness with information overload. The third objective is to evaluate whether different language-model architectures exhibit specialized strengths across semantic review categories such as functional defect detection and refactoring guidance. Finally, the study investigates whether a dynamic expert-routing mechanism can improve overall review quality by assigning tasks to the most suitable model according to semantic context.

To accomplish these goals, this work makes several important contributions to the field of automated software engineering. First, a curated dataset of Python code reviews was developed using GitHub pull requests and categorized into semantic classes through a semi-supervised SVM classification framework. Second, a Retrieval-Augmented Generation pipeline was designed to retrieve historically relevant review comments and provide contextual grounding for language-model inference. Third, the system was evaluated using both conventional textual similarity metrics and semantic evaluation approaches, including LLM-as-a-Judge and human expert validation. Finally, the study introduces a hybrid expert-routing mechanism that dynamically selects the most

appropriate language model according to review context, thereby improving both elevance and technical accuracy.

The remainder of this paper is organized as follows. Section 2 reviews existing literature related to modern code review practices, automated review systems, and Large Language Models for software engineering. Section 3 describes the dataset construction process, semantic classification methodology, retrieval architecture, and vector database design. Section 4 presents the experimental setup, evaluation metrics, and model configurations. Section 5 discusses the experimental findings and analyzes the impact of retrieval augmentation and expert routing. Section 6 examines threats to validity and practical limitations, while Section 7 concludes the paper and outlines possible directions for future research.

RELEVANT LITERATURE

Research on code review has evolved considerably over the last several decades, progressing from traditional manual inspection techniques to highly sophisticated automated review systems powered by machine learning and Large Language Models. This section examines the historical development of code review automation, the growing role of LLMs in software engineering, and the emergence of Retrieval-Augmented Generation as a mechanism for context-aware code analysis.

EVOLUTION OF CODE REVIEW AUTOMATION

The origins of modern code review can be traced back to Fagan's formal code inspection methodology, which established peer review as a structured process for identifying defects and improving software quality. Manual review quickly became recognized as one of the most effective methods for detecting implementation errors, improving maintainability, and promoting shared understanding among development teams. However, despite its effectiveness, traditional review processes were labor-intensive and often introduced delays because they depended heavily on reviewer availability and expertise.

To address these limitations, the software industry gradually adopted Static Analysis Tools and rule-based verification systems. These tools automated the detection of syntax errors, coding-standard violations, and known vulnerability patterns. While such systems reduced the burden of repetitive inspections, they were limited by their inability to understand the semantic intent of software implementations. Researchers observed that rule-based systems frequently generated false positives and lacked the contextual understanding necessary to evaluate business logic or architectural decisions.

As software engineering practices evolved, researchers introduced Context-Aware Recommender Systems to improve review efficiency. These systems focused primarily on optimizing reviewer assignment and workflow management rather than directly

generating review comments. Studies demonstrated that incorporating contextual factors such as reviewer expertise, project history, and development activity improved reviewer selection and reduced turnaround time in industrial environments.

The next stage in this evolution involved the development of Automated Code Review systems capable of directly analyzing code changes and generating feedback. Early approaches treated code review as a translation problem in which machine-learning models transformed defective code into corrected versions. Subsequent systems introduced specialized neural architectures designed specifically for understanding code modifications and generating review suggestions. Although these systems represented significant progress, they remained constrained by limited contextual understanding and insufficient project awareness.

The emergence of Large Language Models fundamentally changed this landscape by introducing models capable of sophisticated reasoning over natural language and source code simultaneously. These advances created the foundation for intelligent, context-aware review systems capable of producing detailed and human-like feedback.

LARGE LANGUAGE MODELS IN SOFTWARE ENGINEERING

Large Language Models have significantly expanded the capabilities of automated software engineering systems. Unlike earlier deep-learning architectures, modern LLMs can interpret complex programming constructs, reason about implementation details, and generate explanations using natural language. Recent studies have shown that LLMs can detect vulnerabilities, identify code smells, and assist in software maintenance tasks that were previously difficult to automate using conventional static-analysis techniques.

Researchers have also explored collaborative and multi-agent review architectures in which several models cooperate to analyze source code from different perspectives. Such systems improve fault detection by combining the strengths of multiple specialized agents. Despite these advancements, an important limitation remains: most LLMs operate without sufficient awareness of project-specific context. They often fail to understand historical design decisions, repository conventions, and prior review discussions, which can lead to hallucinated or irrelevant feedback.

To mitigate this issue, several researchers have proposed methods for enriching language-model prompts with contextual information extracted directly from repositories. Techniques such as code slicing and dependency extraction have been used to provide models with relevant variable definitions, execution context, and implementation history. Nevertheless, these approaches remain limited because they generally focus on localized code fragments rather than leveraging broader historical review knowledge.

This limitation has motivated the adoption of Retrieval-Augmented Generation techniques, which combine external information retrieval with generative language modeling. By integrating retrieved historical comments and project-specific examples into the generation process, RAG systems provide language models with contextual grounding that substantially improves accuracy, coherence, and relevance.

The integration of Retrieval-Augmented Generation with dynamic expert routing therefore represents a promising direction for the future of automated code review. By combining contextual memory with specialized model selection, such systems can overcome many of the weaknesses associated with traditional static analyzers and standalone language models, enabling more reliable and practically useful review automation.

CONCLUSION

This study presented a Retrieval-Augmented Generation framework for automated code review in Python software projects by integrating Large Language Models with historical review-context retrieval and semantic task classification. The research addressed one of the central limitations of existing automated review systems: the inability of conventional static-analysis tools and standalone LLMs to provide context-aware, project-specific, and semantically meaningful feedback. By combining retrieval mechanisms, semantic categorization, and dynamic expert routing, the proposed framework demonstrated that automated review systems can move beyond rule-based defect detection toward intelligent and context-sensitive software evaluation. The findings of this research show that Retrieval-Augmented Generation substantially improves the quality and relevance of generated review comments when compared with zero-shot prompting approaches. Larger code-oriented language models particularly benefited from contextual retrieval because historical review examples supplied practical guidance that aligned generated comments with project conventions and reviewer expectations. The experiments also demonstrated that retrieval depth must be carefully balanced. While moderate retrieval improved semantic alignment and technical accuracy, excessive context introduced information overload and reduced performance in smaller models. This phenomenon, referred to as context collapse, highlights the importance of adaptive retrieval strategies in future automated review systems. An important contribution of this work is the introduction of a hybrid expert-routing mechanism that dynamically assigns review tasks to the most appropriate language model according to semantic review category. Rather than depending on a single universal model, the system leveraged the specialized strengths of different architectures for functional analysis, refactoring suggestions, documentation improvement, and discussion-oriented feedback. Experimental results showed that this

routing strategy significantly enhanced overall review quality while also reducing hallucinations and irrelevant suggestions. The research also demonstrated the limitations of relying exclusively on traditional evaluation metrics such as BLEU and ROUGE for assessing automated code review quality. Although these metrics measure textual similarity, they fail to fully capture semantic correctness, contextual relevance, and practical usefulness. The incorporation of LLM-as-a-Judge evaluation and focused human assessment therefore provided a more comprehensive and realistic understanding of review quality. The combination of automated and human-centered evaluation proved essential for validating whether generated comments satisfied professional software-engineering expectations. Beyond its experimental findings, this study contributes to the broader field of software engineering automation by illustrating how retrieval systems and language models can be integrated into practical development workflows. The proposed framework aligns closely with modern DevOps and continuous integration practices because it enables scalable, rapid, and context-sensitive review assistance that can support developers during pull-request evaluation. As software systems continue to expand in complexity, intelligent review automation will become increasingly important for maintaining quality while reducing developer workload.

Despite the promising results, several limitations remain. The study focused primarily on Python projects collected from GitHub repositories, which may restrict generalizability across other programming languages and industrial environments. In addition, the quality of retrieval augmentation depends heavily on the availability and consistency of historical review data. Projects with limited review history may therefore obtain smaller benefits from the proposed approach. Computational cost also remains an important challenge, particularly when large language models and vector retrieval systems are deployed in real-time development pipelines.

Future research should explore multilingual and cross-language review systems capable of supporting broader software ecosystems. Further work is also needed to improve retrieval efficiency, reduce context collapse, and develop adaptive context-selection strategies that dynamically optimize retrieval depth according to task complexity and model capacity. Another promising direction involves integrating repository-level knowledge graphs, software architecture representations, and developer interaction histories to provide deeper contextual awareness. Finally, combining Retrieval-Augmented Generation with static-analysis techniques and formal verification methods may lead to hybrid systems capable of producing not only semantically meaningful feedback but also verifiable correctness guarantees.

In conclusion, this study demonstrates that Retrieval-Augmented Generation combined with semantic expert routing provides an effective and scalable approach for intelligent

automated code review. By reducing hallucinations, improving contextual awareness, and generating feedback that closely resembles expert human evaluation, the proposed framework advances the practical application of Large Language Models in software engineering. As AI-assisted development tools continue to evolve, context-aware automated review systems are likely to become an essential component of modern software quality assurance practices.

REFERENCES

[1] Kuntamukkala, N. K., & Thalary, S. (2021). Self-Optimizing Angular Applications: A Novel Framework for AI-Driven Performance Adaptation in Production Environments. *International Journal of AI, BigData, Computational and Management Studies*, *2*(2), 107-117.

[2] Han, B., Susnjak, T., & Mathrani, A. (2024). Automating systematic literature reviews with retrieval-augmented generation: A comprehensive overview. *Applied Sciences*, *14*(19), 9103.

[3] Thalary, S., & Katipelly, A. (2021). CI/CD for Distributed Software Systems: Why Software Architecture Determines Pipeline Complexity. *International Journal of Emerging Research in Engineering and Technology*, *2*(4), 100-111.

[4] Parvez, M. R., Ahmad, W., Chakraborty, S., Ray, B., & Chang, K. W. (2021, November). Retrieval augmented code generation and summarization. In *Findings of the Association for Computational Linguistics: EMNLP 2021* (pp. 2719-2734).

[5] Thalary, S., & Kuntamukkala, N. K. (2022). Operationalizing Software Invariants: A DevOps-Driven Approach to Reliability in Cloud-Native Systems. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(4), 157-168.

[6] Lu, S., Duan, N., Han, H., Guo, D., Hwang, S. W., & Svyatkovskiy, A. (2022, May). Reacc: A retrieval-augmented code completion framework. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (pp. 6227-6240).

[7] Thalary, S. (2022). Cloud Cost, Reliability, and Speed: The Triangle Every Enterprise Struggles With. *International Journal of Emerging Research in Engineering and Technology*, *3*(4), 141-152.

[8] Rani, S. J., Deepika, S. G., Devdharshini, D., & Ravindran, H. (2024, October). Augmenting code sequencing with retrieval-augmented generation (rag) for context-aware code synthesis. In *2024 First International Conference on Software, Systems and Information Technology (SSITCON)* (pp. 1-7). IEEE.

[9] Thalary, S., & Katipelly, A. (2023). Secure-by-Design Cloud Software Delivery: How DevOps and Software Teams Co-Own Security Outcomes. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *4*(1), 131-140.

[10] Gao, X., Xiong, Y., Wang, D., Guan, Z., Shi, Z., Wang, H., & Li, S. (2024, October). Preference-guided refactored tuning for retrieval augmented code generation. In *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering* (pp. 65-77).

[11] Katipelly, A., & Thalary, S. (2023). Cryptographic Identity Propagation in Asynchronous Event-Driven Architectures: Implementing Zero-Trust Envelopes for High-Velocity Payment Streams. *International Journal of Emerging Trends in Computer Science and Information Technology*, *4*(2), 212-222.

[12] Lu, H., & Liu, Z. (2024, October). Improving retrieval-augmented code comment

generation by retrieving for generation. In *2024 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (pp. 350-362). IEEE.

[13] Thalary, S. (2023). Monitoring Isn't Observability: Lessons from Running Enterprise Microservices. *International Journal of Emerging Research in Engineering and Technology*, *4*(2), 139-148.

[14] Andrade, V. B. D. (2024). Evaluating the effect of retrieval augmented generation in Mistral-7b-Instruct-v0. 2's clojure's code review.

[15] Thalary, S. (2024). From Pipelines to Policy: Embedding AI-Ready Governance into Cloud DevOps at Scale. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *5*(1), 200-210.

[16] Li, Y., Zhao, J., Li, M., Dang, Y., Yu, E., Li, J., ... & Tao, C. (2024). RefAI: a GPT-powered retrieval-augmented generative tool for biomedical literature recommendation and summarization. *Journal of the American Medical Informatics Association*, *31*(9), 2030-2039.

[17] Thalary, S., & Katipelly, A. (2024). Cloud-Native Design for Event-Driven Systems: Where Software Architecture Decisions Meet DevOps Reality. *International Journal of AI, BigData, Computational and Management Studies*, *5*(2), 202-212.

[18] Verma, S. (2024). Contextual compression in retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2409.13385*.

[19] Katipelly, A., & Thalary, S. (2024). Semantic Automation of Basel III Liquidity Reporting: Utilizing Ontological Knowledge Graphs for Real-Time Regulatory Compliance and Auditability. *International Journal of Emerging Research in Engineering and Technology*, *5*(2), 147-156.

[20] Wang, Y., Guo, S., & Tan, C. W. (2024). Contextual Augmented Multi-Model Programming (CAMP): A Hybrid Local-Cloud Copilot Framework. *arXiv preprint arXiv:2410.15285*.

[21] Kuntamukkala, N. K., & Thalary, S. (2024). Intelligent Angular Architecture: Machine Learning-Based Component Recommendation Systems for Enterprise-Scale Development. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *5*(4), 276-284.

[22] Bhattarai, M., Santos, J. E., Jones, S., Biswas, A., Alexandrov, B., & O'Malley, D. (2024, September). Enhancing code translation in language models with few-shot learning via retrieval-augmented generation. In *2024 IEEE High Performance Extreme Computing Conference (HPEC)* (pp. 1-8). IEEE.

[23] Thalary, S., & Katipelly, A. (2025). Platform Engineering for Distributed Systems: How Cloud DevOps Enables Scalable, Policy-Driven Software Architectures. *International Journal of AI, BigData, Computational and Management Studies*, *6*(1), 189-197.

[24] Nishikawa, K., Koreki, G., & Kanuka, H. (2024, December). Enhancing Source Code Comment Generation via Retrieval-Augmented Generation with Design Document Term Dictionary. In *2024 31st Asia-Pacific Software Engineering Conference (APSEC)* (pp. 467-471). IEEE.

[25] Katipelly, A., & Thalary, S. (2025). Carbon-Aware Dynamic Batching for Deep Learning Inference: Optimizing the Energy-Latency Trade-off in High-Frequency Transaction Monitoring. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *6*(3), 160-169.