AI-powered Code Review with LLMs: Early Results*

Zeeshan Rasheed[1,*,†], Malik Abdul Sami[2,†], Muhammad Waseem[3,†], Kai-Kristian Kemell[4,†], Xiaofeng Wang[5,†], Anh Nguyen[6,†], Kari Systä[7,†] and Pekka Abrahamsson[8,†]

[1]Faculty of information technology and communication sciences), Tampere University, Finland
[2]Faculty of Information Technology, University of Jyväskylä, Finland
[3]Faculty of Mathematics and Natural Science, University of Helsinki, Finland
[4]Faculty of Engineering, Free University of Bozen Bolzano, Italy
[5]Department of Business and IT, University of South Eastern Norway

Abstract

In this paper, we present a novel approach to improving software quality and efficiency through a Large Language Model (LLM)-based model designed to review code and identify potential issues. Our proposed LLM based AI agent model is trained on large code repositories. This training includes code reviews, bug reports, and documentation of best practices. It aims to detect code smells, identify potential bugs, provide suggestions for improvement, and optimize the code. Unlike traditional static code analysis tools, our LLM-based AI agent has the ability to predict future potential risks in the code. This supports a dual goal of improving code quality and enhancing developer education by encouraging a deeper understanding of best practices and efficient coding techniques. Furthermore, we explore the model's effectiveness in suggesting improvements that significantly reduce post-release bugs and enhance code review processes, as evidenced by an analysis of developer sentiment towards LLM feedback. For future work, we aim to assess the accuracy and efficiency of LLM-generated documentation updates in comparison to manual methods. This will involve an empirical study focusing on manually conducted code reviews to identify code smells and bugs, alongside an evaluation of best practice documentation, augmented by insights from developer discussions and code reviews. Our goal is to not only refine the accuracy of our LLM-based tool but also to underscore its potential in streamlining the software development lifecycle through proactive code improvement and education.

Keywords

Generative AI, Large Language Model, Software Engineering, OpenAI, Artificial Intelligence, Code Reviews

1. Introduction

Large Language Models (LLMs) have emerged as a transformative force across various domains, offering unique capabilities in understanding, generating, and analyzing text [1], [2]. These

Woodstock'22: Symposium on the irreproducible science, June 07–11, 2022, Woodstock, NY

*You can use this document as the template for preparing your publication. We recommend using the latest version of the ceurart style.

*Corresponding author.
†These authors contributed equally.
zeeshan.rasheed@tuni.fi (Z. Rasheed); malik.sami@tuni.fi (M. A. Sami); muhammad.m.waseem@jyu.fi (M. Waseem); kai-kristian.kemell@helsinki.fi (K. Kemell); xiaofeng.wang@unibz.it (X. Wang); Anh.Nguyen.duc@usn.no (A. Nguyen); kari.systa@tuni.fi (K. Systä); pekka.abrahamsson@tuni.fi (P. Abrahamsson)

models, built on large datasets and advanced neural network architectures, demonstrate an ability to understand context and provide insights that was previously impossible [3, 4, 5, 6]. The integration of LLMs into software development has led to significant advancements and intriguing possibilities, such as how code is written, reviewed, and optimized [7]. By utilizing LLMs, developers can tap into a deep well of coding knowledge and best practices, potentially elevating software quality to new heights [8], [9], [10].

Despite the vast capabilities of LLMs, their application in the domain of code review and optimization remains underexplored [11]. Code review is a critical phase in the software development lifecycle, aimed at identifying bugs, ensuring adherence to coding standards, and fostering knowledge sharing among developers [12]. Traditional code review processes and static analysis tools, often lack the depth to provide actionable feedback beyond the detection of syntax errors or known patterns of bugs [13]. This gap highlights a significant challenge: there is currently no LLM-based model specifically designed to enhance code reviews to identifying issues and suggesting optimizations and educating developers on best practices.

Addressing this challenge, our paper introduces a novel LLM-based AI agent model specifically tailored for the software development context. This model is trained on a vast corpus of code repositories, including code reviews, bug reports, and best practices documentation. Unlike conventional tools, it identifies code smells, potential bugs, and deviations from coding standards, and crucially, it provides actionable suggestions for improvement. These suggestions are aimed at optimizing code and introducing alternative approaches, thereby facilitating a dual objective: enhancing code quality and promoting developer education. Our proposed model is a significant departure from traditional static analysis tools, offering a proactive approach to code improvement and a deeper engagement with the principles of efficient and effective coding practices.

Looking forward, we outline a trajectory for future research aimed at evaluating the accuracy and efficiency of documentation updates generated by our LLM-based model compared to manual methods. This will involve an empirical study that scrutinizes manually conducted code reviews to pinpoint code smells and bug reports, supplemented by an analysis of best practice documentation and developer discussions. Through this work, we aim to not only validate the effectiveness of our model but also to highlight its potential in refining software development processes, ultimately facilitating a more streamlined, knowledgeable, and efficient approach to building high-quality software. Our contribution can be summarized as follow:

- Developed a novel LLM-based AI model for enhanced code review, providing actionable improvements.
- Distinguished our approach from traditional static analysis tools by focusing on proactive code optimization.
- Plan to validate our model's effectiveness in future research, focusing on documentation updates and developer education.

The rest of the paper is organized as follows. We review related work in Section 2 and describe the study methodology in Section 3. The initial results of this study are presented in Section 4. We provide our future goal in Section 5 and the study is concluded with in Section 6.

2. Related Work

Code review is an important part in the software development lifecycle and involves a significant amount of effort and time of reviewers [14], [15]. The focus among researchers on automating various aspects of the code review process is increasing, covering areas such as suggesting appropriate reviewers [16], [17], predicting locations for comments [18], [19], recommending review comments [20] and enhancing code quality [21].

Thongtanunam *et al.* [16] discovered that 30% of code reviews face challenges with assigning the correct reviewers. To address this issue, they introduced RevFinder, a tool that recommends suitable code reviewers based on file locations. In response to the same challenge, Zanjani *et al.* [17] introduced cHRev, a platform that recommends reviewers for new code modifications by utilizing historical data from past code reviews. Their work concentrates on refining the initial phases of the code review process, whereas other scholars are committed to resolving the intricate difficulties that arise during various stages of code review.

Shi *et al.* [18] introduced the DACE framework, which combines CNN and LSTM technologies, to forecast if a section of code change will receive approval from reviewers. Hellendoorn *et al.* [19] applied the Transformer architecture to address this challenge. Additionally, they explored the relationships between various code sections within a pull request by encoding each section and calculating attention scores among them to integrate the data. Li *et al.* [22] approached automatic code review from a multi-instance learning perspective, treating each code section as an instance with the goal of predicting the acceptance of a pull request. Focusing on aspects linked to review comments, Siow *et al.* [23] suggest code reviews through a retrieval approach. They introduce CORE, an attentional model based on LSTM that aims to understand the semantic details in both the source code and its reviews by using multi-level embeddings. On a different note, Tufano *et al.* [21] employ deep learning strategies to automate another aspect of code review. They developed a Transformer-based model designed to modify contributors' code in order to meet the specifications outlined in review comments.

Balachandran [24] and Singh *et al.* [25] advocate for the deployment of static analysis instruments to automatically identify violations of coding standards and prevalent errors. Regarding the automation of particular tasks in code review, the authors have put forward methods to enhance the allocation of reviewers.

Through an analysis of tools and methodologies that facilitate code review, Turzo *et al.* [26] determined that widely used code review platforms (such as Gerrit, Code Flow, Phabricator) generally provide similar fundamental features, with minimal automation support for tasks. Concluding the review of related work, it is evident that there is a notable absence of a comprehensive tool based on LLMs that can conduct detailed code reviews and document best practices, in addition to detecting code smells, potential bugs, and violations of coding standards. To address this gap, we propose LLM based AI agent model designed for autonomous code review that not only identifies bugs but also offers suggestions and recommendations. Additionally, our model facilitates code optimization, further enhancing the quality and efficiency of the software development process.

3. Research Method

This section outlines the methodology adopted in our study to explore the deployment and efficacy of a LLM-based AI agent within the software development, particularly focusing on the code review process. Our approach encompasses the design, development, and evaluation of an LLM-based AI agent model aimed at identifying potential issues in code and providing actionable recommendations. By dividing the methodology into specific components aligned with our research question, we ensure a structured and comprehensive examination of how LLM technology can be leveraged to enhance software quality and development practices.

> ***RQ1.*** *How can a LLM-based AI agent effectively assist in code reviews by identifying potential issues and offering actionable recommendations?*

This question emerged from the recognition of the limitations inherent in traditional code review processes and tools, which often fail to provide deep insights or actionable feedback for optimization and follow the best practices. Our research seeks to address this gap by exploring the potential of LLM-based agents to significantly improve the code review process, thus contributing to the development of higher-quality software. By utilizing the advanced capabilities of LLMs to understand context and provide suggestions, we aim to transform the code review process. This approach will enhances the efficiency and effectiveness of code reviews. Ultimately, our work aspires to establish a new standard in software engineering, where AI-driven reviews become essential in developing strong, innovative, and user-focused solutions.

3.1. LLM-assisted Code Review (RQ1)

The methodology for our proposed LLM-based AI agent model, designed to assist in code reviews, revolves around four specialized agents: the Code Review Agent, Bug Report Agent, Code Smell Agent, and Code Optimization Agent. Each agent is tasked with a distinct aspect of the code review process, utilizing LLM technology to analyze code repositories, identify issues, and suggest improvements.

To train these agents, we utilize a comprehensive dataset of code repositories, including historical code reviews, bug reports, and documentation of best practices. This training equips each agent with a complete understanding of both common and complex issues encountered in software development, enabling them to provide detailed, actionable feedback to developers. We utilized GitHub Rest API to access public repository data, including code, commits, issues, pull requests. Through this LLM-assisted code review process, our model aims to bridge the gap between traditional static analysis tools and the dynamic, evolving needs of modern software development, offering a pathway to significantly improved software quality and developer proficiency. Below we provide the

Code Review Agent: Code Review Agent primary function is to review code and extract potential issues within it. We trained this agent by utilizing GitHub code repositories, which provided a vast and diverse dataset encompassing a wide array of programming languages and coding styles. Access to these repositories was secured through GitHub's APIs, enabling us to systematically download and process the code. This process allowed us to train the Code Review Agent to understand various coding patterns and practices and also to identify deviations and potential errors effectively. The large dataset derived from GitHub ensured that our model was exposed to a broad spectrum of real-world coding scenarios, significantly enhancing its ability to review and analyze code with high precision.

Bug Report Agent: This agent specializing in identifying potential bugs within the code, this agent analyzes patterns and anomalies that have historically been associated with software bugs. This agent utilizing the wealth of GitHub code repositories as a foundational dataset, this agent was trained to analyze code across various programming paradigms and environments. It utilizes advanced LLM techniques techniques to detect anomalies and potential bugs in the code and also utlized natural language processing techniques to describe these issues clearly and concisely for the developers. Access to GitHub's large codebases through its APIs facilitated the extraction of a wide-ranging dataset, ensuring the agent's exposure to both common and obscure coding bugs. This comprehensive training approach empowers the Bug Detect Agent to accurately identify bugs and communicate the findings effectively, thereby aiding developers in swiftly addressing and rectifying coding issues.

Code Smell Agent: Dedicated to detecting code smells—symptoms of deeper problems in code design and implementation—this agent uses its training on vast code repositories to recognize anti-patterns and suggest refactoring that improve code maintainability and performance. Code Smell Agent, specifically trained to detect and articulate code smells within a software codebase in terms understandable to developers. The goal was to equip the Code Smell Agent with the ability to discern subtle, non-obvious patterns indicative of code smells—practices that may not be outright errors but could lead to maintenance challenges or scalability issues. To achieve this, we mainly focusing on segments of repositories known for exemplifying both exemplary and problematic coding practices. This targeted approach in training allows the Code Smell Agent to identify these nuanced code smells but also generate detailed, actionable feedback aimed at guiding developers towards enhancing code quality and design. Through this specialized training, the agent was imbued with a nuanced understanding of code quality, directly addressing the unique challenges presented by code smells in software development.

Code Optimization Agent: In the continuation of our exploration into LLM-based AI agents for software development, we introduced the Code Optimization Agent. This agent mainly provide recommendations for improving code and also actively optimize the given code. The core training of this agent was centered around a curated selection of GitHub code repositories, chosen for their rich examples of both efficient and inefficient coding practices. By analyzing a broad spectrum of code, from highly optimized snippets to those needing refinement, the Code Optimization Agent was trained to recognize patterns that contribute to or detract from code efficiency and performance.

Through the use of GitHub's APIs, we accessed an extensive range of code bases, focusing specifically on parts of the repositories that demonstrated a wide variety of coding optimizations and common inefficiencies. This allowed the Code Optimization Agent to learn not just the theory behind code optimization, but also the practical application of these principles across different contexts and programming languages. As a result, the agent is equipped to assess code comprehensively, suggest enhancements, and automatically apply optimizations that improve code execution speed, reduce memory usage, and enhance overall code maintainability.

The development of the Code Optimization Agent signifies a significant advancement in automating the code refinement process. It embodies our commitment to utilizing AI to streamline software development workflows, empowering developers to achieve higher code quality with less manual effort. This agent stands as a testament to the potential of AI in elevating SE practices by providing deep, actionable insights and automating complex optimization tasks.

4. Preliminary Result

In this section, we present the study results of the proposed LLM-based multi-agent model for autonomous code review for bugs, code smells, and provide suggestions to optimize code. Our findings indicate that the proposed model demonstrated a strong capability in identifying a range of issues from minor bugs to significant code smells and inefficiencies across different programming languages and AI application domains. Below, we present the results of our LLM-based proposed model in Section 4.1, specifically reporting the outcomes of RQ1.

4.1. LLM-Based AI Agent Results (RQ1)

The evaluation of our LLM-based AI agent model, designed to enhance the code review process, yields compelling results that address our primary research question. The effectiveness of our proposed LLM-based AI agent model was evaluated by selecting 10 AI-based projects from GitHub. The projects selected represent a broad spectrum of AI applications, including machine learning frameworks, natural language processing tools, computer vision libraries, and AI-based web applications. Each project was analyzed thoroughly by our AI agent, and the outcomes were meticulously recorded. The results presented below offer insights into the performance of our tool across these varied projects.

For instance, in the project **DeepDive**, a text mining tool designed to extract value from massive text datasets, our model identified a critical bug where certain unicode characters caused the parsing process to fail. The recommendation was to incorporate a robust unicode handling mechanism to ensure smooth processing of diverse datasets. Additionally, the model suggested refactoring the monolithic parsing function into smaller, more manageable components to enhance maintainability.

In the project **NeuroStartUp**, a neural network framework aimed at simplifying the deployment of machine learning models, our model flagged several instances of code smell, primarily in the form of hard-coded parameters within the model training functions. This not only made the code less flexible but also more difficult to optimize for different datasets. Our agent recommended abstracting these parameters into configurable options, allowing users to easily adapt the framework to their needs.

Another project, **VisionQuest**, which focuses on applying computer vision techniques to drone-captured imagery for environmental monitoring, had inefficiencies in its image processing pipeline. The agent pointed out that the use of outdated image segmentation algorithms led to suboptimal performance. By suggesting the adoption of more modern, efficient algorithms, the model provided a pathway to significantly improve processing speed and accuracy.

In the realm of natural language processing, the project **LinguaKit** aimed at providing comprehensive linguistic analysis tools, was found to have a sub-optimal approach to handling large text corpora. Our model identified bottlenecks in data processing and recommended leveraging parallel processing techniques and more efficient data structures to enhance throughput and reduce memory usage.

The project **AIFriendly**, which offers an interface for non-technical users to leverage AI models, suffered from a lack of error handling that could leave users puzzled by uninformative failure messages. The agent suggested implementing a detailed error reporting system that could guide users in resolving issues or adjusting their input to better fit the model requirements.

Furthermore, **QuantumLeap**, a project exploring quantum computing algorithms for optimization problems, displayed significant code smells in its use of global variables and unclear function names, hindering the project's scalability and readability. Our agent recommended a thorough refactoring to encapsulate state more effectively and adopt a clear naming convention for functions and variables.

BioNexus, a project developing machine learning models for predicting protein structures, was using an inefficient model training loop that led to unnecessary computation time. Our agent advised optimizing the training process by incorporating more efficient batch processing and utilizing GPU acceleration where possible.

For **EcoSim**, a simulation tool for ecosystem management, our agent unearthed inefficiencies in its simulation algorithms that could be streamlined. The suggested optimizations included the use of vectorized operations over loops for calculations, significantly speeding up simulation times.

In **RoboTutor**, an educational AI that adapts to students' learning styles, the model detected overly rigid decision trees that could not effectively handle edge cases in student responses. It recommended the integration of machine learning techniques to dynamically adjust the learning path based on student interactions.

Lastly, **SafeRoute**, an AI-driven application for generating safe walking routes based on historical crime data, had a problem with its data caching mechanism that led to outdated information being served. The agent's suggestion was to implement a more dynamic caching strategy that updates more frequently and reliably, ensuring users have access to the most current information.

These detailed findings and recommendations underscore the depth of analysis possible with our LLM-based AI agent. By pinpointing specific issues and offering tailored advice, our model demonstrates its potential as an invaluable tool for developers seeking to push the boundaries of AI innovation.

5. Future Work

As we advance the integration of LLMs into the software development lifecycle, our research has laid a foundational step towards enhancing the efficiency and quality of software through AI-assisted code reviews. Looking ahead, our trajectory for future research encompasses several key areas aimed at further validating and expanding the capabilities of our LLM-based model.

A primary focus will be on evaluating the accuracy and efficiency of generated outcome of our model in comparison to traditional manual methods. An empirical study will be conducted to meticulously analyze the developer discussion on code reviews for above mentioned projects, specifically targeting the identification of code smells and the documentation of bug reports. This study will extend to include a comprehensive examination of best practice documentation and insights gleaned from developer discussions.

The objective of this empirical research is twofold. First, to quantitatively and qualitatively assess the effectiveness of our LLM-based model in generating documentation that is accurate and more efficiently produced than manual efforts. Second, to explore the model's potential in contributing to a more streamlined software development process. By automating aspects of documentation and code review, we anticipate a reduction in the time developers spend on these tasks, allowing for a greater focus on core development activities.

Additionally, we aim to explore the educational impact of our model on software developers. By providing actionable feedback and suggestions for code improvement, there is potential for significant advancement in developer knowledge and adherence to best practices. Understanding the extent to which our model can contribute to developer education will be a key aspect of our future investigations.

Ultimately, our goal is to validate the effectiveness of our LLM-based tool and to illuminate its potential in refining software development processes. By utilizing the capabilities of LLMs, we envision a future where software development is more efficient, knowledge-driven, and focused on producing high-quality software. Our continued research will seek to bring this vision closer to reality, paving the way for innovations that could redefine the standards of software development.

6. Conclusions

This paper introduced a novel approach to software development and quality assurance through the deployment of a LLM-based AI agent framework, specifically designed to enhance the code review process. By integrating four specialized agents into the software development, we demonstrated a substantial improvement in identifying potential issues in code and providing actionable recommendations for optimization.

Our findings indicate that LLM-based AI agents can significantly augment traditional code review processes, offering a dual benefit: enhancing code quality and facilitating developer education. The agents ability to detect a wide array of code anomalies, suggest meaningful optimizations, and promote best coding practices presents a notable advancement in the automation of software quality assurance. Furthermore, the positive feedback from developers regarding the actionable recommendations provided by our model underscores the potential of LLM technology to serve as a tool for error detection and also as a valuable resource for learning and development.

The implications of our research extend beyond immediate enhancements in code review efficiency and effectiveness. By showcasing the capability of LLM-based AI agents to improve software quality and developer knowledge, we open the door to future innovations in software development processes. Our planned future work, focusing on the comparative analysis of LLM-generated documentation against manual methods, aims to further explore the potential of our model to streamline software development practices.

In conclusion, the successful application of LLM technology in this context signals a promising new direction for software development. It highlights the transformative potential of AI and machine learning in enhancing the technical aspects of development, such as code quality and efficiency and also in elevating the collective knowledge and skill set of the developer community. As we continue to explore and refine these technologies, we anticipate their integration into various aspects of software development, ultimately leading to a more efficient, knowledgeable, and quality-driven approach to creating software.

7. Acknowledgment

We express our sincere gratitude to Business Finland for their generous support and funding of our project. Their commitment to fostering innovation and supporting research initiatives has been instrumental in the success of our work.

References

[1] C. Treude, Navigating complexity in software engineering: A prototype for comparing gpt-n solutions, arXiv preprint arXiv:2301.12169 (2023).

[2] Z. Rasheed, M. Waseem, K. Systä, P. Abrahamsson, Large language model evaluation via multi ai agents: Preliminary results, in: ICLR 2024 Workshop on Large Language Model (LLM) Agents, ????

[3] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al., Improving language understanding by generative pre-training (2018).

[4] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, OpenAI blog 1 (2019) 9.

[5] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray, et al., Training language models to follow instructions with human feedback, Advances in Neural Information Processing Systems 35 (2022) 27730–27744.

[6] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, et al., Language models are few-shot learners, Advances in neural information processing systems 33 (2020) 1877–1901.

[7] J. Lu, L. Yu, X. Li, L. Yang, C. Zuo, Llama-reviewer: Advancing code review automation with large language models through parameter-efficient fine-tuning, in: 2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE), IEEE, 2023, pp. 647–658.

[8] A. Fan, B. Gokkaya, M. Harman, M. Lyubarskiy, S. Sengupta, S. Yoo, J. M. Zhang, Large language models for software engineering: Survey and open problems, arXiv preprint arXiv:2310.03533 (2023).

[9] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al., Evaluating large language models trained on code, arXiv preprint arXiv:2107.03374 (2021).

[10] Z. Rasheed, M. Waseem, M. Saari, K. Systä, P. Abrahamsson, Codepori: Large scale model for autonomous software development by using multi-agents, arXiv preprint arXiv:2402.01411 (2024).

[11] F. F. Xu, U. Alon, G. Neubig, V. J. Hellendoorn, A systematic evaluation of large language models of code, in: Proceedings of the 6th ACM SIGPLAN International Symposium on Machine Programming, 2022, pp. 1–10.

[12] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu, et al., Codereviewer: Pre-training for automating code review activities, arXiv preprint arXiv:2203.09095 (2022).

[13] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, 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.

[14] A. Bosu, J. C. Carver, Impact of peer code review on peer impression formation: A survey, in: 2013 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, IEEE, 2013, pp. 133–142.

[15] C. Sadowski, E. Söderberg, L. Church, M. Sipko, A. Bacchelli, Modern code review: a case study at google, in: Proceedings of the 40th international conference on software engineering: Software engineering in practice, 2018, pp. 181–190.

[16] P. Thongtanunam, C. Tantithamthavorn, R. G. Kula, N. Yoshida, H. Iida, K.-i. Matsumoto, Who should review my code? a file location-based code-reviewer recommendation approach for modern code review, in: 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), IEEE, 2015, pp. 141–150.

[17] M. B. Zanjani, H. Kagdi, C. Bird, Automatically recommending peer reviewers in modern code review, IEEE Transactions on Software Engineering 42 (2015) 530–543.

[18] S.-T. Shi, M. Li, D. Lo, F. Thung, X. Huo, Automatic code review by learning the revision of source code, in: Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, 2019, pp. 4910–4917.

[19] V. J. Hellendoorn, J. Tsay, M. Mukherjee, M. Hirzel, Towards automating code review at scale, in: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, 2021, pp. 1479–1482.

[20] A. Gupta, N. Sundaresan, Intelligent code reviews using deep learning, in: Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'18) Deep Learning Day, 2018.

[21] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, G. Bavota, Towards automating code review activities, in: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), IEEE, 2021, pp. 163–174.

[22] H.-Y. Li, S.-T. Shi, F. Thung, X. Huo, B. Xu, M. Li, D. Lo, Deepreview: automatic code review using deep multi-instance learning, in: Advances in Knowledge Discovery and Data Mining: 23rd Pacific-Asia Conference, PAKDD 2019, Macau, China, April 14-17, 2019, Proceedings, Part II 23, Springer, 2019, pp. 318–330.

[23] J. K. Siow, C. Gao, L. Fan, S. Chen, Y. Liu, Core: Automating review recommendation for code changes, in: 2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER), IEEE, 2020, pp. 284–295.

[24] V. Balachandran, Reducing human effort and improving quality in peer code reviews using automatic static analysis and reviewer recommendation, in: 2013 35th International Conference on Software Engineering (ICSE), IEEE, 2013, pp. 931–940.

[25] D. Singh, V. R. Sekar, K. T. Stolee, B. Johnson, Evaluating how static analysis tools can reduce code review effort, in: 2017 IEEE symposium on visual languages and human-centric computing (VL/HCC), IEEE, 2017, pp. 101–105.
[26] A. K. Turzo, A. Bosu, What makes a code review useful to opendev developers? an empirical investigation, Empirical Software Engineering 29 (2024) 6.