SoftwareX 26 (2024) 101677

Contents lists available at ScienceDirect

SoftwareX

journal homepage: www.elsevier.com/locate/softx

Original software publication

AICodeReview: Advancing code quality with AI-enhanced reviews

Yonatha Almeida [a], Danyllo Albuquerque [c,b,*], Emanuel Dantas Filho [b], Felipe Muniz [b], Katyusco de Farias Santos [b], Mirko Perkusich [c], Hyggo Almeida [c], Angelo Perkusich [c]

[a] Accenture, Paraiba, Brazil
[b] Federal Institute of Paraiba (IFPB), Paraiba, Brazil
[c] Research, Development and Innovation Centre of Federal University of Campina Grande (VIRTUS/UFCG), Paraiba, Brazil

ARTICLE INFO

Keywords:
Artificial Intelligence (AI)
Code review
Automated assessment
Software quality enhancement

ABSTRACT

This paper presents a research investigation into the application of Artificial Intelligence (AI) within code review processes, aiming to enhance the quality and efficiency of this critical activity. An IntelliJ IDEA plugin was developed to achieve this objective, leveraging GPT-3.5 as the foundational framework for automated code assessment. The tool comprehensively analyses code snippets to pinpoint syntax and semantic issues while proposing potential resolutions. The study showcases the tool's architecture, configuration methods, and diverse usage scenarios, emphasizing its effectiveness in identifying logic discrepancies and syntactical errors. Finally, the findings suggest that integrating AI-based techniques is a promising approach to streamlining the time and effort invested in code reviews, fostering advancements in overall software quality.

Code metadata

Current code version	1.1.1
Permanent link to code/repository used for this code version	https://github.com/ElsevierSoftwareX/SOFTX-D-23-00603
Permanent link to Reproducible Capsule	https://plugins.jetbrains.com/plugin/21106-ai-code-review
Legal Code License	Apache License.
Code versioning system used	Git.
Software code languages, tools, and services used	Java and Kotlin.
Compilation requirements, operating environments & dependencies	IntelliJ IDE
Documentation/manual	https://github.com/Yonatha/ai-code-review
Support email for questions	yonathalmeida@gmail.com

1. Motivation and significance

Code Review is a software quality assurance activity where the main checking is done by one or several humans, and at least one of these humans is not the code's author [1]. The checking is performed mainly by viewing and reading the source code, generally executed after its implementation [1]. The humans performing the checking, excluding the author, are called "reviewers" [2].

Every definition aspect establishes boundaries for code review concerning other quality assurance techniques, including static analysis, self-checks, testing, and pair programming [3]. However, these distinctions become less clear in practice [4]. For instance, human reviewers may receive support from static code analysis, occasionally conduct tests or interact with the graphical user interface (GUI), authors may participate in reviewing their code, and collaborative issue resolution between authors and reviewers can essentially resemble pair programming [5].

In light of the particularities involved in code review, it becomes evident that automated support can aid reviewers in their tasks [6]. This assertion aligns with the growing recognition of modern software projects' escalating complexity and scale [7]. The incorporation of automated tools not only addresses the challenges posed by the extensive code volumes [8] but also introduces the potential for increased accuracy, consistency, and efficiency in identifying issues, enhancing the overall quality assurance in software development [9]. Furthermore,

* Corresponding author.
E-mail address: danyllo.albuquerque@virtus.ufcg.edu.br (Danyllo Albuquerque).

https://doi.org/10.1016/j.softx.2024.101677
Received 9 September 2023; Received in revised form 24 February 2024; Accepted 28 February 2024
Available online 5 March 2024

automated support in code review aligns with the broader industry trend towards leveraging technology to streamline and optimize software development practices [10].

The advancement of Artificial Intelligence (AI) has played a significant role in enhancing various processes in software engineering [11], including the identification and resolution of code problems [12]. Several studies have focused on using AI-based techniques to support code review. For instance, Sharma's study Sharma et al. [13] conducted a systematic review revealing the potential of AI and ML in enhancing automated code reviews. Notable tools include Zhou et al.'s DeepCom [14] using neural networks for automatic code comment generation, and Lal et al. [15] analyzed ML-based techniques to support code review. Similarly, Ayewah et al.'s FindBugs [16] and Shi et al.'s DACE [17] used ML and rules to identify code problems. Finally, Albuquerque et al.'s tool [18] combines static analysis and criteria-based decision-making for continuous code smell detection.

Large Language Models (LLMs) and their highlighted advantages come to the forefront in this context. Models such as ChatGPT and Google Bard, among others, exhibit advanced Natural Language Processing (NLP) capabilities, suggesting its potential to automate code review processes [19]. More recently, some studies have explored using LLMs to improve automated support for code review [19–21]. These studies generally share a common concern for improving code review through LLM-based techniques. They demonstrate how LLMs can outperform existing methods, providing empirical evidence that LLM-based support for code review can efficiently (i) analyze and comprehend the syntactic and semantic structure of code, (ii) identify and analyze potential issues, categorizing them, and (iii) prioritize suggestions for issue resolution.

Given the potential of leveraging LLMs in code review, the current study proposes a novel approach called "AICodeReview". It introduces the creation of a plugin designed for IntelliJ IDEA,[1] which utilizes GPT-3.5[2] for the automation of code review. Using this plugin, developers can identify potential code issues and enhance the quality and security of ongoing software development. Consequently, the code review can be automated, saving time and effort while boosting productivity and quality. This study outlines the steps in constructing the "AICodeReview" tool, provides configuration and usage details, and discusses the practical implications of employing such tools in code review.

The remainder of this paper is structured as follows: Section 2 provides a motivating example of using automated support for code review. Section 3 describes the main features of proposed automated support for code review. Section 4 describes results from the preliminary validation of proposed automated support for code review. Section 5 discusses the study's threats to validity and our actions to mitigate them. Finally, Section 6 presents our final remarks and future work suggestions.

2. Illustrative example

Let us explore a hypothetical scenario to provide a balanced evaluation of the AICodeReview tool. Consider two software developers, Alice and Bob, collaborating on a project to implement a new feature. Both developers must review their code, seeking problems such as antipatterns, bad design decisions, errors, and vulnerabilities. While Alice opts for the AICodeReview tool, Bob chooses a manual review without utilizing the tool.

With the tool's assistance, Alice identifies several issues in the code that Bob overlooks during his manual review. For example, the AI-driven mechanism of AICodeReview can help her to detect errors such as (i) a variable not initialized correctly, (ii) a potential memory leak in a code segment, and (iii) a logic error within a conditional structure. AICodeReview promptly suggests solutions, allowing Alice to comprehend and address errors more efficiently.

In contrast, Bob faces challenges in identifying some of these issues during his manual review, necessitating additional time and effort for problem identification and resolution. Consequently, Bob's source code, besides being more time-consuming, exhibits more errors and vulnerabilities compared to Alice's, which was AICodeReview-aided. This introduces errors that may compromise software quality and security, posing potential risks to software use, evolution, and maintenance.

The utilization of AICodeReview can be a promising approach to deliver substantial benefits to developers in code review. The tool might enhance skill and programming knowledge and provide valuable insights into software architecture, programming logic, and coding best practices. Its AI-driven mechanism identifies errors that might be overlooked in a manual review, showcasing precision and thoroughness beyond traditional methods. Moreover, AICodeReview goes a step further by offering insightful suggestions for issue resolutions, proving valuable in projects with tight deadlines or where code quality is paramount for project success. It is noteworthy that while the tool exhibits significant promise, further empirical evidence and rigorous evaluation are essential to substantiate these claims.

3. Software description

This section provides a detailed overview of the AICodeReview tool. First, we briefly introduce the tool (Section 3.1). Subsequently, we outline the tool's operational aspects (Section 3.2). Finally, we describe the architecture and implementation details (Section 3.3).

3.1. Main features

The AICodeReview tool is accessible via the IntelliJ IDE Marketplace.[3] Its first operational version was released in mid-March 2023 and has already been adopted by over 5000 distinct developers. Once the tool is configured within IntelliJ, developers can interact with it to perform one or more of the following described activities:

- *Code review across multiple programming languages*: AICodeReview offers support for code review across several relevant programming languages in the market, such as Java, Kotlin, C++, and Python. This versatility allows the tool to be used in projects developed with various languages without restrictions.
- *Customization in solution suggestions*: Using AICodeReview, developers can fine-tune the precision of solution suggestions for code review to meet specific project needs. This configuration allows developers to adjust the level of detail in suggestions, ranging from general advice to more specific and detailed recommendations.
- *Explanation and reasoning*: AICodeReview's key feature is its ability to suggest code revisions with detailed explanations of the underlying reasoning. This enables developers to understand the rationale behind each suggestion and learn from the recommendations produced by the tool.
- *Language preferences*: AICodeReview supports notifications in multiple languages, including English, Spanish, and Portuguese. This feature enhances tool accessibility for non-English speakers, allowing them to receive notifications in their preferred language.
- *Compatibility with JetBrains family*: AICodeReview is compatible with all products within the JetBrains product family, enabling its use with other tools developed by the same company (e.g., PyCharm, WebStorm, and Android Studio). This integration provides developers with a more unified and efficient development experience across various tools.

[1] https://www.jetbrains.com/idea/
[2] https://openai.com/blog/chatgpt
[3] https://plugins.jetbrains.com/plugin/21106-ai-code-review/

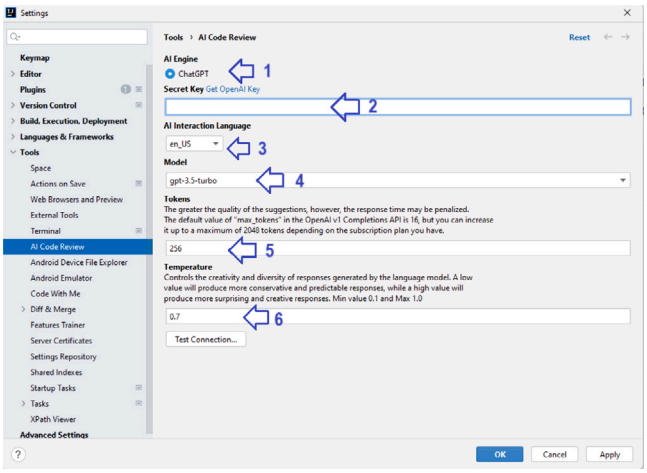

Fig. 1. AICodeReview setup.

In summary, AICodeReview can review code in numerous programming languages, offering customization features for precision and supporting both general and specific suggestions. Moreover, it functions as a learning tool, providing detailed explanations for the reasoning behind code review suggestions. This is particularly valuable in semantic errors requiring a more elaborate debugging and correction process. Overall, AICodeReview provides a straightforward solution for developers engaging in code review, seeking to enhance the quality of their code.

3.2. AICodeReview operation

This section provides a high-level overview of the operation of the AICodeReview tool. In the subsequent sections, we highlight its configuration and usage within IntelliJ.

Configuration. Before utilizing the tool, it is imperative to ensure appropriate configuration. The selection of the AI engine is a pivotal step in tool configuration. Currently, GPT-3.5 serves as the default engine (Fig. 1 - label 1), necessitating an OpenAI key[4] for its utilization (Fig. 1 - label 2). However, due to architectural considerations, other engines might be incorporated. Following the engine choice, language selection is required (Fig. 1 - label 3). The tool supports three languages: Portuguese, English, and Spanish. This ensures that code suggestions are presented in the most suitable developer's language.

Subsequently, a text generation model needs selection (Fig. 1 - label 4). Several models are available, each with its advantages and disadvantages. For instance, the *text-davinci-003* model is recognized for its potential to generate high-quality recommendation text. Token configuration is equally crucial (Fig. 1 - label 5). The default value is 256, but it can be increased up to a limit of 2048. Greater values extend engine response time, potentially leading to more precise and detailed code suggestions. Lastly, the temperature parameter requires configuring within the range of 0.1 to 1.0 (Fig. 1 - label 6). Lower values result in conservative and predictable code suggestions, while higher values foster creativity and unexpectedness. Striking a balance between predictability and creativity is paramount to ensure code suggestions are valuable to developers.

Using. Upon configuring the tool, its utilization follows a straightforward process comprised of four main steps, as depicted in Fig. 2. Each of these steps will be elaborated upon in detail below:

1. *Code Snippet Selection*: The initial step in utilizing the AICodeReview tool involves selecting the code snippet the developer wishes to review. This can be accomplished using the mouse cursor or keyboard shortcuts.
2. *Code Review Request*: The developer must explicitly request a code review following the code snippet selection. This can be done through a keyboard shortcut or a designated button within the tool's interface.
3. *Code Review Process*: Upon the code review request, AICodeReview analyzes the selected code snippet based on the parameters set during configuration. This analysis typically takes only a few seconds, contingent upon the code's complexity and predefined settings. GPT-3.5 is prompted with contextual information about the code under review, objectives, and criteria for improvement. Using its natural language generation, it produces human-like suggestions for code enhancements. The process involves an iterative refinement loop, adapting to user feedback to fine-tune recommendations.
4. *Presentation of Review Results*: Once the analysis is complete, AICodeReview presents the code review results within the user interface. These results encompass a specific suggestion for code enhancement and a comprehensive explanation of the reasoning underpinning the suggestion. GPT-3.5 considers coding best practices and standards, presenting users with actionable suggestions for code improvement in a code review setting. The suggestions assist the developer in comprehending the necessary code improvements and the underlying motivations behind the recommendation.

In summary, AICodeReview streamlines its configuration and utilization stages for simplicity and ease. Notably, the tool conducts code reviews only upon explicit request from the developer, thereby minimizing the potential intrusion into programming activities. Utilizing the tool, developers can simplify the code review process by offering personalized suggestions and elucidating their rationale. Finally, employing the tool saves developers time and effort during code reviews, enhancing the quality of their code.

3.3. Architectural details

The AICodeReview comprises eight distinct components, which are elaborated upon in Fig. 3. Notably, three of these components, namely (i) the IntelliJ Platform, (ii) the Program Structure Interface, and (iii) the ChatGPT API, were provided by JetBrains and OpenAI. These components were integrated to leverage the existing functionalities of the IntelliJ IDE and the ChatGPT API, respectively. The remaining five components were meticulously developed using the Java programming language. This strategic choice allowed for the seamless integration of the GPT-3.5 API into the IntelliJ IDE, enhancing the tool's capabilities and enabling effective code review processes.

1. IntelliJ Platform: It is a set of software development tools created by JetBrains, including IntelliJ IDEA, providing infrastructure for creating desktop, web, and mobile applications across various domains such as finance, healthcare, government, and technology. The platform enables *plugins* developed in either Java or Kotlin languages to access IDE features (e.g., code editor, debugger, menus, and windows) through its SDK.
2. Program Structure Interface (PSI): This component encompasses a range of functionalities, from swift navigation to files, types, and symbols to code inspections and refactoring. Although AICodeReview was created in Java, its utility extends beyond code reviews in this language due to PSI's power and its syntactic and semantic code models, which are compatible with multiple languages.

[4] https://platform.openai.com/account/api-keys

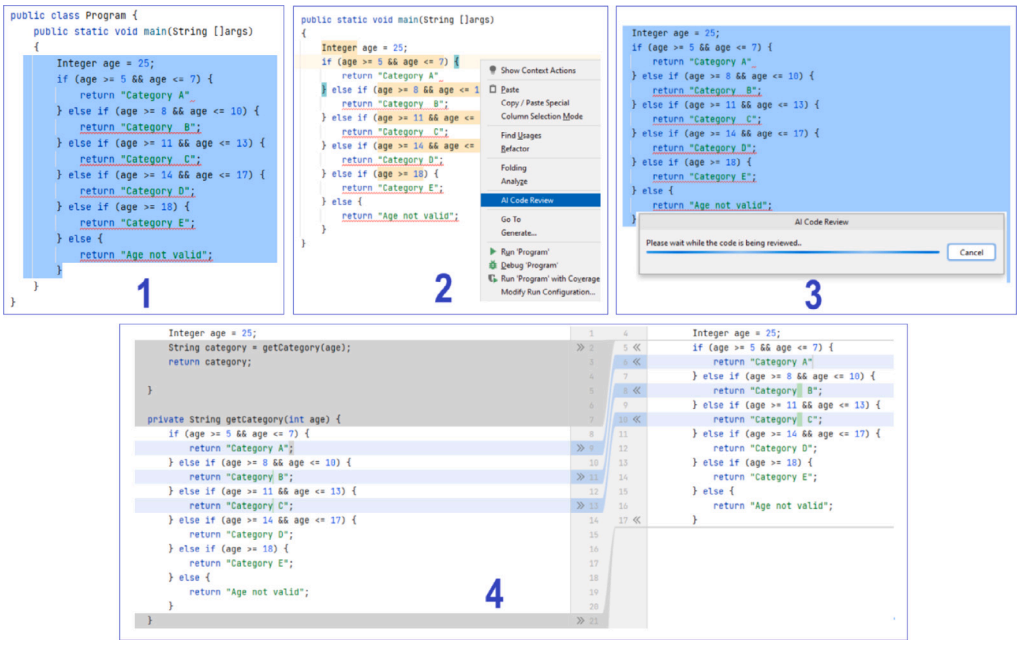

Fig. 2. Operation steps of the AICodeReview.

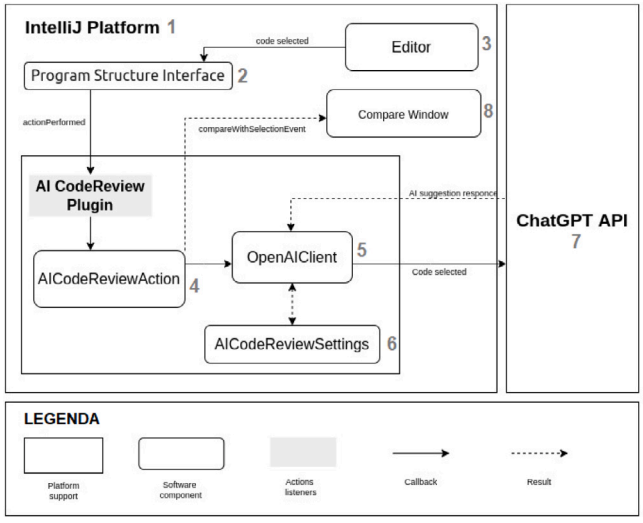

Fig. 3. AICodeReview architecture.

3. Editor: This component is where developers write code using PSI features that interpret the programming language. The developer triggers AICodeReview via the *actionPerformed* function through this component. The *AICodeReviewAction* component, through the *actionListener CodeReviewAction*, is activated by registering it under the *groupid CodeMenu* within the IntelliJ Platform component. This grants access to the code review function in the IDE's main Code menu. Similarly, through an *actionListener* registered with *grupoid EditorPopupMenu3*, users can access the code review function through the context menu of the *Editor* component.
4. AICodeReviewAction: This component represents the action executed by the developer after selecting the code for which they desire Artificial Intelligence suggestions.
5. OpenAIClient: This component, triggered by the *AICodeReviewAction*, uses configuration parameters from the AICodeReviewSettings component to make requests to the external layer of the IntelliJ Platform, described in the GPT-3.5 API component.
6. AICodeReviewSettings: The information shown in Fig. 1 is stored in this component. This component provides predefined configurations such as model types to enable communication between the *OpenAIClient* and the GPT-3.5 API. A diverse set of models with different capabilities powers the OpenAIClient (e.g., gpt-3.5-turbo, text-davinci-003, and code-davinci-002). The developer can customize their models for your specific use case with fine-tuning. We chose the gpt-3.5-turbo over the other predefined models because of its lower cost and improved performance.[5] Besides models, the developer should set the preferred language (e.g., English, Spanish, and Portuguese) for GPT-3.5 API's code review suggestions.
7. GPT-3.5 API: This component represents the external layer of the plugin. It is the service provided by the developer of GPT-3.5 (i.e., OpenAI). This component utilizes a *bearer token*, configured in the *AICodeReviewSettings* under the *Secret key* field. It creates suggestions to improve the selected code and sends details to be shown in the *Compare Window* using the *AICodeReviewAction*.
8. Compare Window: This component compares the user-selected code with the suggested code from GPT-3.5. To accomplish this, PSI resources are employed to perform syntactic analysis and comparison within a single window, as depicted in Step 4 of Fig. 2.

The AICodeReview architecture demonstrates modularity, enabling smooth integration of the tool with the IntelliJ Platform and the GPT-3.5 API (Fig. 3). This modular structure lends itself to simplifying tool maintenance and evolution. Furthermore, the tool preserves the potential for expansion, readily accommodating adaptation to various programming languages and seamless integration with diverse AI-based or ML-based services. This adaptability of the tool enhances the code review process, resulting in improved code quality, productivity, and overall development efficiency.

4. Preliminary evaluation

In this study, we preliminary validated the AICodeReview tool's effectiveness within code review. The study involved a sample of 12

[5] https://platform.openai.com/docs/models/gpt-3-5

Table 1
Average results on experimental tasks.

Group	Time	Detected code smells	Refactored code smells
Experimental	15.2	28	25
Control	22.5	20	13

undergraduate students. All participants had a basic foundation in programming, particularly in Java, and were familiar with using IntelliJ. Furthermore, they demonstrated a pre-established comprehension of code review, code smells, and their importance in software quality. All participants underwent a training session on its features and operation to ensure a consistent understanding of the AICodeReview tool's functionality. This training session aimed to set a baseline proficiency level for both groups, reducing potential differences in tool usage competence during the experiment. Our participant sample was strategically divided into two groups: (i) an *experimental group* comprising six students who leveraged the AICodeReview tool and (ii) a *control group* encompassing six students who executed manual code reviews.

Our investigation primarily focused on evaluating source code segments encompassing around 30 different types of code smells. We deliberately selected two distinct files (files A and B) for analysis to support this assessment. The methodology employed in this study utilized a carefully designed factorial treatment approach. This design aimed to provide both groups an equal chance to review both sets of files while employing their assigned review method (either the AICodeReview tool or manual review). Through this carefully designed experimental arrangement, we aimed to evaluate the efficiency and effectiveness of the two review methodologies. The metrics examined covered various aspects, including (a) the time taken to review each file, (b) the number of detected code smells, and (c) the number of refactored code smells. To provide a more comprehensive understanding of the practical implications of using the AICodeReview tool compared to manual code reviews, we incorporated Cohen's d effect size in our analysis to enhance the depth of our findings. It provides a robust statistical tool to convey the significance of the observed differences in metrics considered in our study, reinforcing the validity and applicability of our research outcomes. We have included detailed information and calculations regarding Cohen's effect size in the supplementary material of this study [6].

The experiment results and the analysis of Cohen's d effect sizes reveal significant and noteworthy differences between the experimental group utilizing the AICodeReview tool and the control group conducting manual code reviews regarding the considered metrics (Table 1). Let us delve into each metric and its corresponding discussion:

Regarding *review time*, the data analysis reveals a notable contrast between the two groups. The experimental group (i.e., using AICodeReview) showed a shorter average review time of 15.2 min, while the control group (i.e., using manual review) took a longer average time of 22.5 min to complete their assessments. This significant discrepancy underscores the AICodeReview tool's capacity to enhance the review process compared to conventional manual reviews. Cohen's d for the "Review Time" task was −1.664, indicating a large effect size, suggesting that the AICodeReview tool significantly reduced the review time compared to manual review. The tool's efficiency in automating the detection of code smells and providing actionable solutions is pivotal in this observed efficiency. By autonomously identifying and suggesting corrections for code smells, the tool reduces the necessity for reviewers to meticulously inspect each code snippet for potential issues, thereby reducing the time compared to manual review. This outcome aligns with the broader advantages of automation in software engineering, where tools can swiftly handle repetitive tasks, liberating human resources to focus on higher-level tasks requiring creative and analytical thinking. This enhancement in review speed can lead to more efficient development cycles and faster integration of code changes, contributing to improved development productivity and overall software quality.

Regarding the *number of detected code smells*, the experimental group (i.e., using AICodeReview) indicated an average detection of 28 code smells. In contrast, the control group (i.e., using manual review) identified an average of 20. This difference suggests that the AICodeReview tool holds a distinct advantage in its capacity for comprehensive code analysis, enabling it to pinpoint a broader spectrum of code smells. Cohen's d for this task was 1.402, indicating a large effect size, implying that the AICodeReview tool was much more effective at detecting code smells than manual review. The tool's algorithmic approach and automation capabilities enhance its ability to analyze code segments for complex, difficult, and less obvious issues that human reviewers might inadvertently overlook. Code smells include various issues, ranging from subtle logic errors to complexities in coding practices. These issues can be challenging for human reviewers to identify, especially in complex codebases.

Concerning *refactored code smells*, a notable difference emerges between the experimental and control groups. The experimental group (i.e., using AICodeReview) achieved an average of 25 refactored code smells compared to the control group (i.e., manual review), which achieved an average of 13 refactored code smells. This difference underscores the AICodeReview tool's potential to support effective code smell refactoring. Cohen's d for this task was 1.612, indicating a large effect size and suggesting that the AICodeReview tool was highly effective at helping developers refactor code smells compared to manual review. The tool's ability to offer tailored, data-driven suggestions assists developers in addressing smells more precisely and efficiently. This can be attributed to the tool's expansive knowledge base, encompassing diverse coding practices and patterns. In contrast, the control group might have faced challenges pinpointing root smell causes and generating precise refactoring actions. By synergistic partnership between AI-driven analysis and human expertise, we conclude that the number of correct refactored smells can be improved.

According to the aforementioned results, the significant effect sizes for all three metrics (Review Time, Detected Code Smells, and Refactored Code Smells) indicate that the AICodeReview tool significantly impacted code review efficiency and effectiveness compared to manual review. The AICodeReview tool's data-driven analysis supports developers with a discerning eye to identify an extended range of potential code smell concerns. This insight underscores the advantage of employing AI-based techniques in code review, as they perform systematic, consistent, and comprehensive assessments, leading to a more robust identification of code smells. The AICodeReview can enhance code quality and software reliability by augmenting human capacities with AI-driven insights. It brings attention to issues that might be ignored in human reviews, making it easier to fix problems early with less time and effort.

It is important to emphasize the observed outcome variations can be attributed to the inherent automation of the AICodeReview tool. This tool can conduct exhaustive analyses and promptly provide consistent correction suggestions. In contrast, the manual review process, as employed in the control group, is more prone to human errors, subjective interpretation differences, and extended time consumption. The results highlight that incorporating the AICodeReview tool into the code review process leads to faster, more thorough, and more effective outcomes than traditional manual reviews. This underlines the potential of AI-based techniques in improving the code review process.

To provide additional analysis, Table 2 presents a comprehensive comparison analysis of the findings from "our study" against those of related research papers in code review practices and tools. This analysis aims to provide a holistic view of how our study's contributions and insights align or diverge with the existing body of knowledge in the

[6] https://github.com/Yonatha/ai-code-review/

Table 2
Comparison analysis of results against related work.

#	Paper	Focus	Participants	Methodology	Metrics	Findings
1	Tufano et al. [9]	investigate the impact of code review on software quality.	involved 12 software developers and engineers.	examines the relationship between code review participation and defect density.	Metrics include defect density and code review participation rates.	Active code review participation is correlated with lower defect density.
2	Lu et al. [19]	relationship between code review processes and team collaboration.	include software development teams.	assesses how code review practices influence collaboration within development teams.	include collaboration scores and code review process data.	Code review processes impact team collaboration positively.
3	Zhou et al. [21]	investigate the effectiveness of static analysis tools in code reviews.	include software developers and engineers.	include collaboration scores and code review process data.	include defect detection rates and code quality improvement.	Static analysis tools enhance defect detection and improve code quality.
4	Guo et al. [20]	examine the impact of code review comments on code quality	include software developers and engineers.	analyzes code review comments and their influence on code changes.	comment sentiment analysis and code change outcomes.	Positive code review comments are associated with improved code quality.
5	Li et al. [7]	evaluate the performance of code review models using pre-training.	Not applicable; the study focuses on model performance.	performance of code review models in code change quality estimation, review generation, and code refinement.	include precision, recall, F1 score, accuracy, and BLEU scores.	Pre-trained models improve code review performance, with CodeReviewer outperforming baselines.
6	Our paper	evaluate the effectiveness of the AICodeReview tool in code reviews.	involved 12 undergraduate students with a basic foundation in programming.	Metrics: Metrics include precision, recall, F1 score, accuracy, and BLEU scores.	included review time, the number of detected code smells, and the number of refactored code smells.	The tool reduced review time, detected more code smells, and supported code smell refactoring.

domain. By juxtaposing key aspects such as research focus, participant demographics, methodologies, metrics, and major findings, this table enables a clearer understanding of our study's unique contributions and implications within the broader context of code review research.

Regarding "Focus", Papers 1, 2, and 4 primarily investigate the impact of code review practices and collaboration within software development teams. Our study and Papers 3 and 5 focus on evaluating tools and models designed for code review tasks. Related to "Participants", our study involved students as participants, serving as a controlled sample. In contrast, Papers 1, 2, 3, and 4 engaged software professionals or teams actively involved in software development processes. Concerning "Methodology", our study adopts a factorial treatment approach in its research methodology, allowing for a structured comparison of code review methods. Other studies employed different methodologies, including correlation, comparative, and sentiment analyses, to investigate code review-related phenomena. Related to "Metrics" used across these studies, including defect density, collaboration scores, defect detection rates, sentiment analysis of comments, and model performance evaluation. Our study included review time, the number of detected code smells, and the number of refactored code smells. Finally, regarding the "Findings", our study's primary finding underscores the efficiency and effectiveness of the AICodeReview tool in streamlining the code review process. It specifically highlights improvements in code smell detection and code smell refactoring. Other papers emphasize various factors, such as the impact of collaboration practices, the role of different tools, and the influence of code review comments on overall code quality. In summary, our study demonstrates how AI-driven code review tools, such as AICodeReview, can significantly enhance the efficiency of code review processes and contribute to improved code quality. Each referenced paper adds valuable insights to the broader understanding of code review practices, their impact on software quality, and the factors influencing effective collaboration within development teams.

5. Threats to validity

In what follows, we present the threats to the validity of this study following the classification schema proposed by Wohlin et al. [22].

Construct validity is maintained through the careful design of the experimental setup, which involves an evaluation of source code segments containing various code smells. The deliberate selection of distinct files for analysis, the use of a factorial treatment approach, and the consideration of multiple metrics contribute to the robustness of the experimental design. Another threat is concerning experimental task settings. This study aimed to perform an initial assessment of the AICodeReview tool's usage rather than an exhaustive evaluation involving tools such as those proposed in [19,20]. The consequence of this choice is a possible incomplete understanding of AICodeReview's performance compared to a broader landscape of automated code review tools. To mitigate this threat, future research should consider including various automated code review techniques for a more holistic assessment.

Internal validity is supported by the strategic division of participants into experimental and control groups, ensuring a controlled environment for evaluating the AICodeReview tool's impact. The baseline proficiency level set through a training session further minimizes potential biases and enhances the reliability of the results. The use of procedures across both groups and the meticulous selection of files for analysis contribute to the study's internal validity.

External validity is addressed by including participants with a basic foundation in programming, particularly in Java, and familiarity with IntelliJ. While the study involves undergraduate students, the findings may be generalized to similar populations with programming knowledge and familiarity with code review concepts. However, caution is warranted in extrapolating the results to broader populations with different skill levels.

Conclusion validity is supported by the comprehensive evaluation of various metrics, including review time, detected code smells, and refactored code smells. The presentation of average results in Table 1 provides a clear summary of the outcomes, contributing to the conclusion validity of the study. Another threat is regarding the reproducibility of results. As with many contemporary studies in AI, our study harnesses the capabilities of state-of-the-art LLM (i.e., GPT-3.5). While leveraging such external services offers significant advantages in NLP and generation, it is essential to acknowledge the potential implications on the reproducibility of our results and the broader landscape of AI-driven research. The dynamic nature of AI technologies, their dependency on third-party providers, and the potential for model updates or service discontinuations create a legitimate concern for the long-term reproducibility of AI-driven research outcomes. To mitigate this threat, we underscore the following measures: (i) include clear references to model versions, APIs, or data sources to facilitate reproducibility; (ii) explore local deployment of critical AI components or preserve access to older model versions to maintain result reproducibility, albeit with additional complexity; and (iii) encouraging collaboration within the research community and with service providers can foster solutions that promote reproducibility while leveraging external AI services' benefits.

6. Impact and conclusions

This paper presented the development of a plugin called "AICodeReview" for IntelliJ IDEA that utilizes the emerging LLM-based technique to simplify and enhance the code review process. We demonstrated that the proposed tool can analyze and comprehend the structure and semantics of code, identifying syntax problems, logic errors, and security vulnerabilities from source code upon the explicit request of the developer. As a result, the tool can provide suggestions for solutions and code improvements, streamlining the code review process and ensuring outcomes for this task.

The preliminary experimental evaluation demonstrates substantial advantages of the AICodeReview tool over manual code reviews. Notably, the AICodeReview group exhibited a significantly shorter average review time of 15.2 min compared to the control group's 22.5 min, emphasizing the tool's efficiency in automating code smell detection and providing actionable solutions. The tool's ability to detect a higher average of 28 code smells compared to the control group's 20 illustrates its superiority in comprehensive code analysis. Additionally, the AICodeReview group's achievement of 25 refactored code smells on average, as opposed to the control group's 13, highlights the tool's potential to support effective code smell refactoring. These results underscore the tool's data-driven analysis, which complements human expertise, contributing to improved code quality, software reliability, and more efficient development cycles.

The main contributions of the paper include:

- Introduction of AICodeReview Tool: This study introduces the AICodeReview tool, leveraging GPT-3.5 for automated code review, providing developers with valuable insights and improvement suggestions;
- Preliminary Evaluation with Undergraduate Students: This study presents a preliminary empirical evaluation involving 12 undergraduate students, demonstrating the tool's efficacy compared to manual code reviews; and
- Insight into AI-based Code Review Techniques: This study contributes valuable insights into the advantages of employing AI-based techniques in code review, emphasizing the systematic, consistent, and comprehensive assessments provided by AICodeReview, enhancing code quality and software reliability.

As prospects for future work, we plan to explore the tool's effectiveness when used by code authors as part of their self-checking process. This investigation will involve assessing whether the tool's assistance is more effective when employed by reviewers during formal code reviews or when utilized by code authors for self-assessment. Understanding the comparative advantages and implications of these different usage scenarios will contribute to a more comprehensive understanding of the tool's capabilities and its potential impact on software development practices. Moreover, future evaluations can incorporate advanced statistical analyses and employ rigorous methodological steps to enhance the robustness of AICodeReview's assessment. This can use hypothesis testing, effect size calculations, and potentially exploring Bayesian analysis to provide a more comprehensive understanding of the tool's performance and effectiveness.

We also intend to incorporate various other LLM-based techniques, including but not limited to Llama, Bard, and Copilot, which have demonstrated their significance in the field. Additionally, we aim to explore the potential integration of the AICodeReview plugin with other widely used code analysis tools such as SonarQube and PMD. This integration could lead to an expanded scope of problem detection, enhancing the tool's utility. Furthermore, our research will delve into the analysis of user-generated data. This approach will enable us to continually refine the AI model underpinning the AICodeReview tool, ultimately enhancing its accuracy and efficiency in real-world software development scenarios. Finally, the tool's alignment with change-based code review processes is an important aspect. We anticipate conducting further research and development efforts to accommodate this particular code review type. These endeavors will enhance the AICodeReview tool's capabilities, ensuring its seamless integration and effectiveness within change-based code review scenarios.

CRediT authorship contribution statement

Yonatha Almeida: Validation, Software. **Danyllo Albuquerque:** Validation, Supervision, Formal analysis, Conceptualization. **Emanuel Dantas Filho:** Project administration, Methodology, Investigation. **Felipe Muniz:** Software, Investigation. **Katyusco de Farias Santos:** Writing – original draft, Methodology, Investigation. **Mirko Perkusich:** Writing – original draft, Project administration, Formal analysis. **Hyggo Almeida:** Visualization, Validation, Data curation. **Angelo Perkusich:** Software, Resources, Investigation, Conceptualization.

Declaration of competing interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Danyllo Albuquerque reports was provided by Federal University of Campina Grande Centre of Teacher Education. Danyllo Albuquerque reports a relationship with Federal University of Campina Grande that includes: non-financial support. Danyllo Albuquerque has patent pending to Assignee. Nothing to declare.

Data availability

Data will be made available on request.

Acknowledgments

The authors acknowledge the support provided by the Virtus Research, Development, and Innovation Center, including the VIRTUS-CC (EMBRAPII VIRTUS Competence Center – Intelligent Hardware for Industry), at the Federal University of Campina Grande. EMBRAPII (Brazilian Company for Industrial Research and Innovation) has made this initiative possible through funding from the Brazilian Ministry of Science and Technology (MCTI) under the PPI HardwareBR program.

References

[1] Baum T, Liskin O, Niklas K, Schneider K. A faceted classification scheme for change-based industrial code review processes. In: 2016 IEEE international conference on software quality, reliability and security. 2016, p. 74–85. http://dx.doi.org/10.1109/QRS.2016.19.

[2] Bacchelli A, Bird C. Expectations, outcomes, and challenges of modern code review. In: 2013 35th international conference on software engineering. IEEE; 2013, p. 712–21. http://dx.doi.org/10.1109/ICSE.2013.6606617.

[3] Thongtanunam P, Pornprasit C, Tantithamthavorn C. Autotransform: Automated code transformation to support modern code review process. In: Proceedings of the 44th international conference on software engineering. 2022, p. 237–48. http://dx.doi.org/10.1145/3510003.3510067.

[4] McIntosh S, Kamei Y, Adams B, Hassan AE. An empirical study of the impact of modern code review practices on software quality. Empir Softw Eng 2016;21:2146–89. http://dx.doi.org/10.1007/s10664-015-9381-9.

[5] The Institute of Electrical and Electronics Engineers. IEEE standard for software reviews. 1998, p. 1–48. http://dx.doi.org/10.1109/IEEESTD.1998.237324, IEEE Std 1028-1998.

[6] Tufano R, Pascarella L, Tufano M, Poshyvanyk D, Bavota G. Towards automating code review activities. In: 2021 IEEE/ACM 43rd international conference on software engineering. 2021, p. 163–74. http://dx.doi.org/10.1109/ICSE43902.2021.00027.

[7] Li Z, Lu S, Guo D, Duan N, Jannu S, Jenks G, et al. Automating code review activities by large-scale pre-training. In: Proceedings of the 30th ACM joint European software engineering conference and symposium on the foundations of software engineering. 2022, p. 1035–47. http://dx.doi.org/10.1145/3540250.3549081.

[8] Thongtanunam P, Tantithamthavorn C, Kula RG, Yoshida N, Iida H, Matsumoto K-i. 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. IEEE; 2015, p. 141–50. http://dx.doi.org/10.1109/SANER.2015.7081824.

[9] Tufano R, Pascarella L, Tufano M, Poshyvanyk D, Bavota G. Towards automating code review activities. In: 2021 IEEE/ACM 43rd international conference on software engineering. IEEE; 2021, p. 163–74. http://dx.doi.org/10.1109/ICSE43902.2021.00027.

[10] Sadowski C, Söderberg E, Church L, Sipko M, Bacchelli A. Modern code review: a case study at google. In: Proceedings of the 40th international conference on software engineering: software engineering in practice. 2018, p. 181–90. http://dx.doi.org/10.1145/3183519.3183525.

[11] Perkusich M, e Silva LC, Costa A, Ramos F, Saraiva R, Freire A, et al. Intelligent software engineering in the context of agile software development: A systematic literature review. Inf Softw Technol 2020;119:106241. http://dx.doi.org/10.1016/j.infsof.2019.106241.

[12] Albuquerque D, Guimarães E, Tonin G, Rodriguez P, Perkusich M, Almeida H, et al. Managing technical debt using intelligent techniques-a systematic mapping study. IEEE Trans Softw Eng 2022. http://dx.doi.org/10.1109/TSE.2022.3214764.

[13] Sharma T, Kechagia M, Georgiou S, Tiwari R, Vats I, Moazen H, et al. A survey on machine learning techniques for source code analysis. 2021, http://dx.doi.org/10.48550/arXiv.2110.09610, arXiv preprint arXiv:2110.09610.

[14] Zhou Y, Zhang X, Shen J, Han T, Chen T, Gall H. Adversarial robustness of deep code comment generation. ACM Trans Softw Eng Methodol (TOSEM) 2022;31(4):1–30. http://dx.doi.org/10.1145/3501256.

[15] Lal H, Pahwa G. Code review analysis of software system using machine learning techniques. In: 2017 11th international conference on intelligent systems and control. 2017, p. 8–13. http://dx.doi.org/10.1109/ISCO.2017.7855962.

[16] Ayewah N, Pugh W, Hovemeyer D, Morgenthaler JD, Penix J. Using static analysis to find bugs. IEEE Softw 2008;25(5):22–9. http://dx.doi.org/10.1109/MS.2008.130.

[17] Shi S-T, Li M, Lo D, Thung F, Huo X. Automatic code review by learning the revision of source code. In: Proceedings of the AAAI conference on artificial intelligence, vol. 33, no. 01. 2019, p. 4910–7. http://dx.doi.org/10.1609/aaai.v33i01.33014910.

[18] Albuquerque D, Guimaraes E, Perkusich M, Almeida H, Perkusich A. ConCAD: A tool for interactive detection of code anomalies. In: Anais do X Workshop de Visualização, Evolução e Manutenção de Software. SBC; 2022, p. 31–5. http://dx.doi.org/10.5753/vem.2022.226597.

[19] Lu J, Yu L, Li X, Yang L, Zuo C. 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. IEEE; 2023, p. 647–58. http://dx.doi.org/10.1109/ISSRE59848.2023.00026.

[20] Guo Q, Cao J, Xie X, Liu S, Li X, Chen B, et al. Exploring the potential of ChatGPT in automated code refinement: An empirical study. 2023, http://dx.doi.org/10.48550/arXiv.2309.08221, arXiv preprint arXiv:2309.08221.

[21] Zhou X, Kim K, Xu B, Han D, He J, Lo D. Generation-based code review automation: How far are we? 2023, http://dx.doi.org/10.48550/arXiv.2303.07221, arXiv preprint arXiv:2303.07221.

[22] Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. Experimentation in software engineering. Springer Science & Business Media; 2012, http://dx.doi.org/10.1007/978-3-642-29044-2.