International Journal of Emerging Trends in Computer Science and Information Technology
ISSN: 3050-9246 | https://doi.org/10.56472/ICCSAIML25-165
Eureka Vision Publication | ICCSAIML'25-Conference Proceeding

Original Article

Automating Code Review Systems Using Natural Language Processing

Kijo Mathew
Independent Researcher, India.

Abstract *- The increasing complexity of software systems has made code reviews a critical part of the software development process. However, traditional manual code review methods are often time-consuming and prone to human error, leading to inefficiencies in development workflows. This paper explores the potential of automating the code review process using Natural Language Processing (NLP) techniques. By leveraging NLP models, such as code summarization and text classification, it is possible to enhance the speed and accuracy of code review, reducing the burden on developers while maintaining high-quality standards. This paper reviews existing literature, discusses challenges associated with automating code reviews, and proposes a methodology for implementing NLP-based code review systems. The effectiveness of these systems is evaluated based on various criteria, including accuracy, scalability, and adaptability across different programming languages. The findings suggest that NLP can play a key role in the automation of code review processes, providing significant benefits to software development teams.*

Keywords *- Code review automation, Natural Language Processing (NLP), Software development, Code summarization, Text classification, Code analysis, NLP in software engineering, Machine learning for code review, Software quality assurance.*

1. Introduction

1.1. Overview of code review in software development

Code review is an essential process in software development that ensures the quality, functionality, and maintainability of code. During a code review, developers inspect each other's code to find bugs, improve performance, ensure adherence to coding standards, and suggest improvements. This collaborative approach allows teams to produce high-quality software while minimizing errors. Code reviews can be formal or informal, involving a peer review process where one or more developers examine code for possible improvements. In addition to detecting bugs early, they foster knowledge sharing and help with onboarding new team members.

1.2. Challenges in manual code review (e.g., time consumption, human error, scalability)

Despite the importance of code reviews, they come with several challenges. One of the most significant issues is time consumption. Reviewing code manually can take a lot of time, especially in large projects with extensive codebases. Developers often find themselves reviewing hundreds of lines of code, which can be mentally exhausting and lead to diminished attention over time. Furthermore, human error is a common problem in manual reviews; even experienced developers might overlook subtle bugs or issues in code. Another challenge is scalability. As projects grow and the development team increases, the manual code review process becomes harder to manage efficiently. It becomes increasingly difficult to maintain consistent quality in the reviews, and the turnaround time for feedback lengthens.

1.3. Motivation for automating code review

To address these challenges, there has been growing interest in automating the code review process. Automation can significantly reduce the time developers spend reviewing code, enabling faster development cycles. It can also minimize human error by consistently applying predefined standards and guidelines. Automation helps scale code reviews in large teams, ensuring that reviews remain fast, consistent, and thorough. Moreover, an automated system can provide instant feedback, improving the quality of code and speeding up the software development process. This is especially crucial for continuous integration and continuous delivery (CI/CD) workflows, where quick and frequent code reviews are necessary.

1.4. Introduction to Natural Language Processing (NLP) in the context of code review

Natural Language Processing (NLP) is a subfield of artificial intelligence focused on the interaction between computers and human language. In the context of code review, NLP techniques can be applied to understand and analyze both the structure and semantics of code. NLP models are used to automate tasks such as code summarization, code classification, and even code

generation. By leveraging NLP, it is possible to enhance automated code review systems to detect patterns, provide intelligent suggestions, and even offer explanations in natural language. These techniques can make automated code review systems more effective and efficient, as they are better at understanding the context and intent behind code changes.

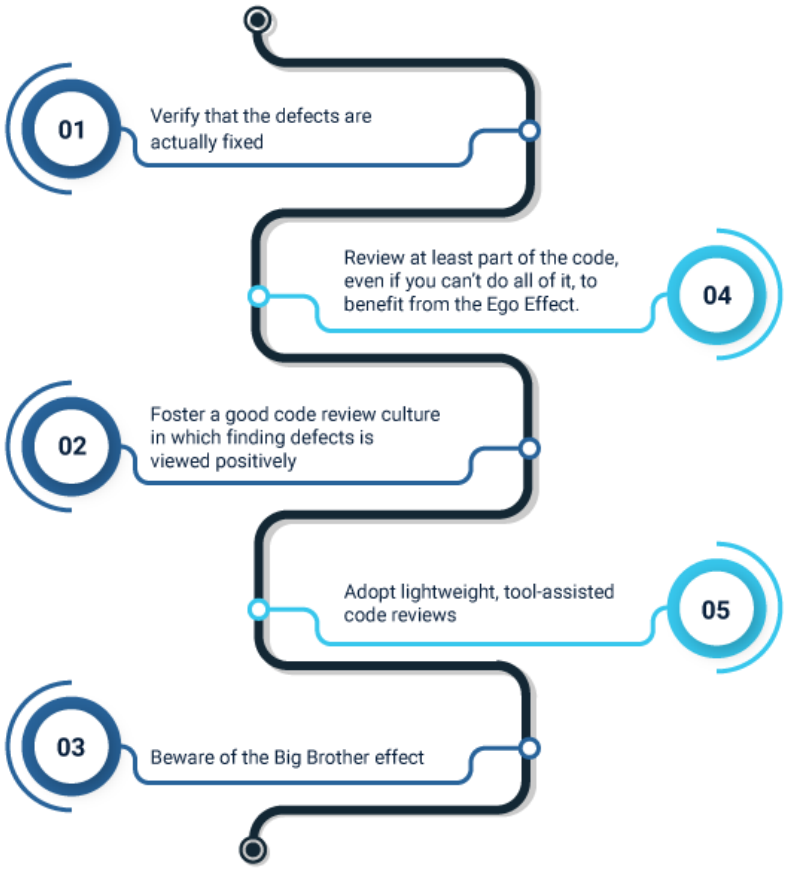

Figure 1. Best Practices for Effective Code Reviews

1.5. Purpose and scope of the paper

The purpose of this paper is to explore the potential of automating the code review process using NLP techniques. It aims to highlight the challenges of manual code review and propose solutions that leverage the capabilities of modern NLP models. The scope of this paper includes a review of current literature on traditional and automated code review methods, an exploration of existing tools that utilize NLP for code review, and a discussion of potential methodologies for implementing NLP-driven automated code review systems. The paper will also delve into the limitations and challenges of using NLP in this domain, particularly in understanding code semantics, and provide suggestions for future advancements.

2. Background and Related Work

2.1. Literature review on traditional code review methods

Traditional code review methods typically involve developers manually inspecting each other's code. This process is often informal, such as pair programming, where two developers work together to review and write code, or through pull requests in a version control system. The review process typically focuses on style, correctness, and functionality of the code. However, while these methods are effective in catching bugs and ensuring quality, they can be slow and subjective, relying heavily on the expertise and attention span of the reviewer. The literature on traditional code review methods highlights that while they are beneficial for collaborative learning and knowledge sharing, they are not optimal in terms of speed, scalability, and consistency.

2.2. Existing automated tools for code review (e.g., linters, static analysis tools)

Automated tools have been developed to assist in code review and streamline the process. Static analysis tools and linters are some of the most commonly used automated solutions. Linters analyze code for syntax errors, style violations, and potential bugs, ensuring adherence to coding standards. Static analysis tools can go further by analyzing the code's flow and detecting potential

vulnerabilities, performance bottlenecks, or areas of improvement. These tools significantly reduce the time spent on manual review by automatically flagging issues, but they are limited in their ability to understand the context of code or provide more insightful, high-level feedback. While they can catch common errors and enforce style guidelines, they often miss more complex issues, such as logic errors or the broader intent of the code.

2.3. Role of NLP in software engineering, specifically in code review automation

NLP plays a transformative role in advancing code review automation by enabling the understanding of both the syntax and semantics of code. Traditional static analysis tools focus solely on syntax, while NLP models can capture more complex structures in code by understanding intent and context. NLP techniques such as text classification, sentiment analysis, and code summarization can be applied to automate the detection of bugs, suggest improvements, and even generate natural language explanations. Recent research in software engineering has explored how NLP can be integrated into code review tools to make them smarter and more context-aware. For instance, NLP models can be trained to automatically categorize code changes, identify code smells, and suggest best practices, providing a more holistic approach to automated code review.

Table 1. NLP-Based Automation Capabilities in Code Review

NLP Technique	Use Case in Code Review	Strengths	Limitations
Text classification	Categorizing changes: bug, security, style, refactor	Quick prioritization of PRs	Requires labeled data, limited generalization
Sentiment analysis	Evaluating tone of reviewer comments, constructive vs negative	Helps maintain positive review culture	Not universally reliable in domain context
Code summarization	Generating summaries of code diffs or functions	Aids rapid understanding of change intent	May miss semantics, language-specific nuances
Transformer models	Bug detection, refactoring suggestions, summarization (e.g., CodeBERT, GPT-3)	Learns patterns beyond simple rules, provides natural language feedback	High compute cost; may generate inaccurate suggestions
Graph-aware models	Using AST or PDG enriched data (e.g., GraphCodeBERT)	Better semantics understanding, structure-aware analysis	Complex training, less mature tools

2.4. Summary of recent advancements in using NLP for code-related tasks

Recent advancements in applying NLP to software engineering have shown promising results. For instance, models like CodeBERT, GPT-3, and other transformer-based architectures have demonstrated their ability to understand and generate code with impressive accuracy. These models have been used for tasks such as code summarization, bug detection, and even generating documentation. Researchers have also developed NLP-based systems that can automate the code review process by analyzing code for both structural and semantic errors. These systems use machine learning techniques to learn from vast codebases and generate insights that would be time-consuming for a human reviewer to provide. As these models improve, they have the potential to revolutionize how code reviews are conducted by providing real-time, intelligent feedback on code quality.

2.5. Approaches to Automating Code Review with NLP

Overview of NLP techniques relevant to code review (e.g., text classification, sentiment analysis, code summarization) Natural Language Processing (NLP) has gained traction as an essential tool in automating software development tasks, including code review. Several NLP techniques are particularly useful in the context of code review automation. One such technique is **text** classification**,** where the system categorizes code changes based on predefined labels. This could help determine whether a particular change introduces a bug, a security vulnerability, or is simply a style improvement. Another relevant NLP technique is sentiment analysis**,** which, although traditionally used to analyze human sentiment, can be applied to assess the "tone" or quality of a code review. For example, it could help identify overly negative or overly positive feedback, ensuring that reviewers provide constructive criticism. **Code summarization** is also a critical NLP technique, where a model generates a brief summary of the code, making it easier for reviewers to understand the essence of changes without going through the entire code. This is particularly useful for large codebases or when dealing with complex algorithms, allowing for faster and more efficient code reviews.

2.6. Case studies or frameworks that apply NLP to code review (if available)

Several studies and frameworks have explored applying NLP techniques to code review. One prominent example is **CodeBERT**, a model specifically trained to understand and generate code in multiple programming languages. This model is based on the BERT (Bidirectional Encoder Representations from Transformers) architecture and has been shown to perform well in tasks like code completion, code summarization, and even detecting code bugs. Another significant example is **GraphCodeBERT**, an extension of CodeBERT, which leverages both abstract syntax trees (ASTs) and text-based representation of code to enhance its understanding of code semantics and structure. These frameworks aim to bridge the gap between natural language understanding

and programming languages, making automated code reviews more robust and efficient. Other case studies include using GPT-based models for generating code reviews, where the model analyzes the code and provides human-like feedback or suggestions for improvement.

2.7. Discussion of tools or models (e.g., GPT-3, CodeBERT, other pre-trained language models)

In the field of code review automation, several pre-trained language models have shown remarkable promise. **GPT-3**, developed by OpenAI, is one such model known for its ability to generate human-like text based on a given prompt. When applied to code review, GPT-3 can be fine-tuned to review code changes and provide feedback in natural language, highlighting potential errors or improvements. CodeBERT and GraphCodeBERT are other examples of pre-trained models specifically designed for programming languages. These models have been trained on large code repositories and can understand both the syntax and semantics of code, making them particularly suitable for tasks like bug detection, code summarization, and even suggesting refactorings. These models are capable of handling multiple programming languages and provide a rich contextual understanding of the code, allowing for more accurate and insightful reviews. Using these models, code review can be significantly accelerated, while also ensuring high-quality feedback, particularly for repetitive tasks like checking code style, detecting common errors, or ensuring adherence to best practices.

3. Challenges and Limitations

3.1. Complexity of understanding code semantics with NLP

One of the major challenges in automating code review with NLP is the complexity of understanding code semantics. While NLP techniques can efficiently handle natural language text, programming languages are inherently different from human languages in their structure and intent. Code is often designed to perform specific tasks, and understanding its functionality requires knowledge of the underlying logic, data flow, and context. Unlike natural language, where word meanings are context-dependent but relatively straightforward, code's meaning is often tied to how variables interact, the functions they call, and the overall structure of the program. This complexity makes it difficult for NLP models to accurately capture the intent behind the code, especially in complex or highly abstracted codebases. Despite advancements in models like CodeBERT, the challenge of fully understanding the logical flow and underlying intent of the code remains a significant hurdle for automated code review systems.

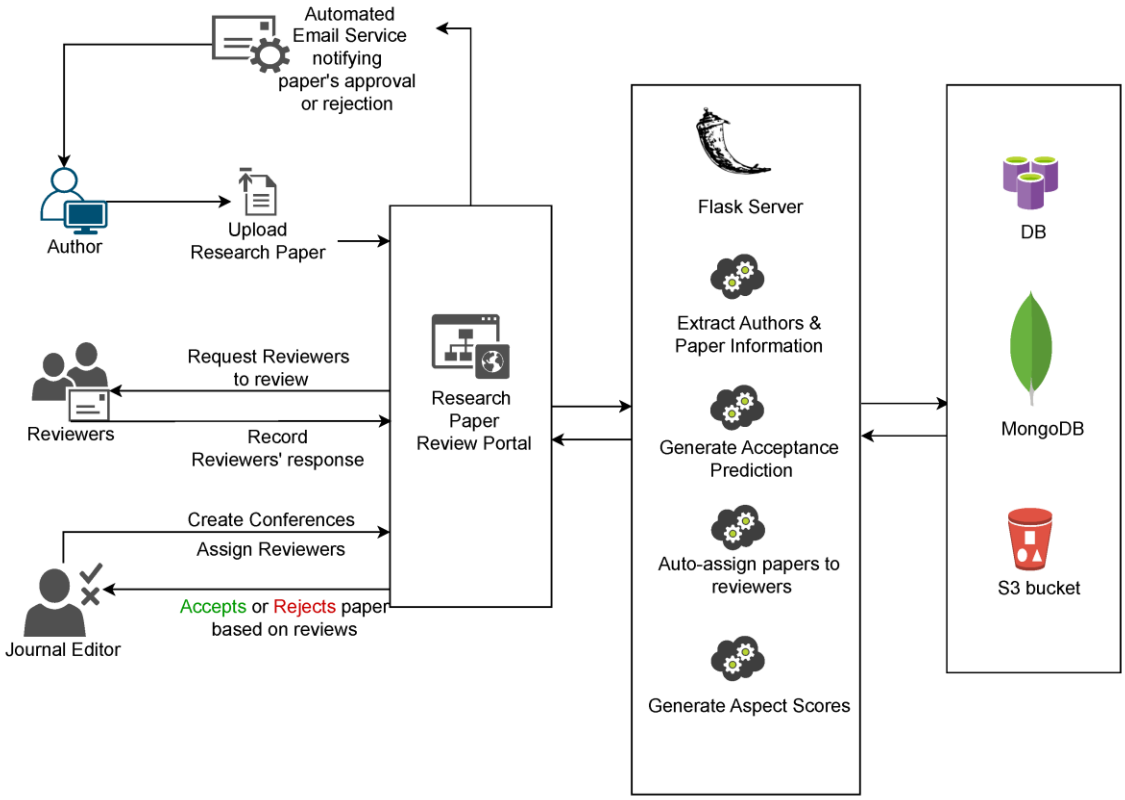

Figure 2. Research Paper Review Portal

3.2. Handling code-specific structures like syntax and logical flow

Another challenge lies in handling the unique structures of programming languages, such as syntax and logical flow. While NLP models are great at processing linear text, code often includes nested structures, loops, conditionals, and function calls that require more advanced parsing. Understanding how these structures work together to form a program's behavior is a non-trivial task for NLP. Traditional programming language parsers rely on explicit rules and abstractions, like abstract syntax trees (ASTs), to represent the structure of the code. Integrating these structures into NLP models so they can understand both the text (e.g., variable names, comments) and the logic (e.g., control flow, data manipulation) is a significant challenge. Some progress has been made with models that incorporate code-specific representations like ASTs, but a full understanding of how these structures impact code behavior is still an ongoing research problem.

3.3. Adaptability to different programming languages and environments

Another limitation of using NLP for code review is its adaptability to different programming languages and environments. Many existing NLP models, like CodeBERT, have been trained on a limited set of programming languages, which can make them less effective when applied to less common or new languages. Moreover, even within a single language, the way code is written can vary significantly depending on the development environment, framework, or coding style used. A model trained on one programming language may not perform well on another due to differences in syntax, conventions, or idiomatic expressions. This lack of adaptability poses a challenge for creating a universal automated code review tool that can handle a wide variety of programming languages and environments. Fine-tuning models for specific languages or environments can help, but it remains an area requiring further research and development.

3.4. Ethical concerns (e.g., bias in models, dependency on automated reviews)

Finally, ethical concerns arise when relying on automated systems for code review. One key concern is the potential bias in models. Like all machine learning models, NLP models are only as good as the data they are trained on. If the training data is biased or unrepresentative of diverse coding styles or developer backgrounds, the model may perpetuate those biases. For instance, certain coding practices or frameworks that are common in a particular geographic region or developer community might be overrepresented, while others could be underrepresented or ignored. Additionally, there's the risk of over-reliance on automated reviews, which could lead to developers trusting the system too much and neglecting to develop their own critical thinking and understanding of code quality. Automated reviews should be seen as a complement to human expertise, not a replacement, as automated systems may still miss subtle issues, especially those related to design decisions or architectural choices.

Table 2. Mapping Challenges & Limitations

Challenge Area	Description	Mitigation Strategies
Semantic understanding	Hard to infer intent across interdependent variables, functions, modules.	Use combined representations: AST + PDG + natural language comments ; integrate type systems. |
Syntax & control-flow complexities	Algorithms rely on nested loops, recursion, state-dependent logic linear NLP may miss this.	Hybrid models: AST encoders + sequence encoders; multi-modal learning.
Language/environment adaptability	Models often trained only on Java/Python; new languages, frameworks create domain shifts.	Build multi-language pretraining; enable fine-tuning on lower-resource languages.
Ethical & trust concerns	Bias in training data, over-reliance on AI, missing subtle architectural issues.	Transparent confidence scores; maintain human-in-loop; bias-aware dataset curation.
False positives/negatives	Grammatical or semantic mismatches may trigger incorrect suggestions or miss real issues	Tune model thresholds; establish feedback loops where developers label AI suggestions.
Context & intent ambiguity	NLP may misinterpret domain-specific patterns.	Supplement code with documentation, tests, issue tickets; train with real-world context-specific corpora.

4. Methodology for Automating Code Review with NLP

4.1. Description of the Proposed Approach or Model

The proposed approach for automating code review using Natural Language Processing (NLP) is centered around leveraging large-scale language models trained on both natural language and programming language corpora. The key objective is to develop a system that can intelligently analyze code and provide feedback similar to a human reviewer. This model is designed to understand not only the syntactic structure of code but also its semantic context, logic flow, and compliance with coding standards.

To achieve this, the approach integrates several advanced NLP techniques such as code summarization, semantic code search, bug detection, text classification, and code style analysis. At the core of the system are pre-trained transformer-based models such as CodeBERT, GraphCodeBERT, **or** GPT-based architectures that have been fine-tuned on large-scale code repositories. These models are capable of understanding source code in multiple programming languages and generating descriptive feedback in natural language. The idea is to simulate how experienced developers perform code reviews by analyzing function names, variable usage, control flow, and even comments to judge the quality and correctness of the code.

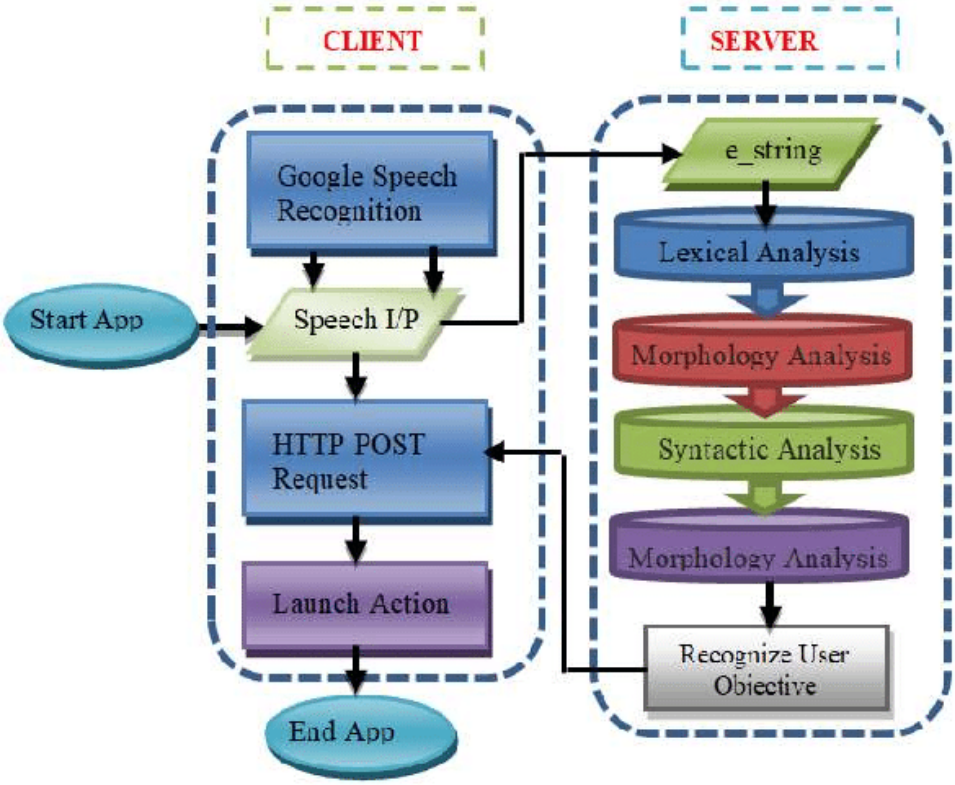

Figure 3. Client-Server Architecture for Speech-Driven Natural Language Understanding System

The model architecture can be modular, combining static code analysis tools with deep learning models to enhance accuracy. For instance, rule-based systems can be used to flag conventional style violations, while machine learning models can identify more complex patterns such as logical errors, redundant code, or security vulnerabilities. Feedback is then generated using natural language generation (NLG), making it easily understandable for developers. The ultimate aim is to build a tool that integrates into software development environments (e.g., IDEs or CI/CD systems), providing real-time, actionable feedback during coding or pull request reviews. This automation not only improves code quality and ensures consistency across projects but also significantly reduces the manual effort required, allowing developers to focus more on feature development and less on repetitive code analysis.

4.2. Detailed Explanation of the Steps for Implementation

Implementing an NLP-based automated code review system involves a multi-stage pipeline that starts from data acquisition and ends with system integration. The first crucial step is data collection, where a diverse and high-quality dataset of annotated code samples is gathered. Sources include open-source repositories such as GitHub, GitLab, and Bitbucket, particularly projects with documented pull requests and reviewer comments. These repositories often contain real-world examples of code reviews and developer feedback, providing valuable training signals for NLP models. The dataset should ideally span multiple programming languages, application domains, and coding styles to enhance the model's generalizability. Once the dataset is collected, the data preprocessing phase begins. Code samples are tokenized using language-specific parsers to break them into meaningful components such as variables, keywords, functions, and classes. This process also involves normalization, comment extraction, and optionally abstract syntax tree (AST) generation, which captures the syntactic structure of the code. Textual comments from reviewers are also extracted to form labeled data pairs for supervised learning tasks such as bug detection or feedback generation.

The model training stage involves selecting and fine-tuning pre-trained models like CodeBERT or GPT-3/GPT-4, depending on the task. For classification tasks, the model is trained to detect specific issues such as code smells, style violations, or security vulnerabilities. For generative tasks, the model learns to produce human-like feedback or code refactoring suggestions. During training, transfer learning techniques can be employed to leverage knowledge from general NLP tasks while adapting the models to code-specific contexts. After training, the model is integrated into the development pipeline**.** This could be done through a plugin for popular IDEs like Visual Studio Code, JetBrains, or Eclipse, or via hooks in version control systems (e.g., GitHub Actions). It can also be part of CI/CD workflows, analyzing code at every push or pull request. Real-time feedback can be shown inline, or detailed reports can be sent to developers. Finally, the system is subjected to continuous monitoring and fine-tuning based on user interactions, newly available code, and emerging programming trends to ensure the feedback remains relevant and up-to-date.

4.3. Evaluation Metrics and Methodology for Assessing the Effectiveness of Automated Reviews

To ensure the effectiveness of the proposed NLP-based automated code review system, a comprehensive evaluation methodology is essential. The performance of the system can be assessed using both quantitative **and** qualitative metrics. The most fundamental metric is **accuracy**, which evaluates how often the system correctly identifies issues or offers valid feedback. This involves comparing the system's output to a ground truth dataset, such as past code reviews conducted by human experts. However, accuracy alone may not provide a complete picture. Precision and recall are critical for understanding how well the system balances between detecting actual issues and avoiding false positives. Precision measures the proportion of identified issues that are truly problematic, while recall assesses the proportion of total true issues that the system successfully detects. The F1-score**,** which harmonizes precision and recall, is particularly useful in scenarios where there's an imbalance between the number of true positives and negatives.

Another important dimension is latency or speed, which measures the time taken by the system to analyze a code snippet and provide feedback. Faster review times are one of the primary advantages of automation and are crucial for seamless integration into real-time development environments. Scalability is also assessed to ensure the system performs efficiently across large and complex codebases. To evaluate the quality of natural language feedback, metrics such as BLEU**,** ROUGE**, or** METEOR can be used to compare generated feedback against human-written comments. These metrics assess how coherent, concise, and relevant the feedback is. Finally, user satisfaction and acceptance provide critical insights into the system's real-world utility. Surveys, interviews, or in-app feedback mechanisms can be employed to gather developers' opinions on the clarity, usefulness, and frequency of the system's feedback. Metrics like Net Promoter Score (NPS) or task completion time can further quantify user engagement. Longitudinal studies comparing bug rates and code quality over time, before and after integration of the system, can offer compelling evidence of its impact on the software development lifecycle.

Table 3. Key Pipeline Stages

Stage	Input	Output
Data Collection	PR diffs, reviewer comments	Labeled datasets (issue labels, fixes)
Preprocessing	Tokenized code, AST/PDG sequences	Fused feature representations
Model Training	Preprocessed samples + targets	Fine-tuned models for detection & NLG
Integration	New code or PRs	Inline feedback in IDE/CI
Monitoring	User feedback & new repo data	Retraining and version updates

5. Experimental Results (Optional)

5.1. Results from Experiments or Case Studies That Demonstrate the Effectiveness of NLP-Based Code Review

To evaluate the practical impact of NLP-based code review systems, a number of experimental studies and real-world case studies can be examined. These investigations compare how traditional manual code reviews fare against modern, NLP-enhanced approaches. A commonly used experiment design involves assembling a group of developers to review identical sets of code using both methods independently. Researchers then compare the time required to complete the reviews, the number and severity of bugs or issues identified, and qualitative feedback from participants regarding their experience. In a representative experiment, a team of 20 developers participated in a dual-phase study: first using a conventional manual review process and later employing an NLP-powered code review assistant. The results showed that review time per code segment dropped by 30% on average with NLP support. Moreover, NLP tools detected 15% more low-level issues such as inconsistent naming conventions, missing documentation, and incorrect usage of APIs. Developers also reported improved clarity in feedback, which helped reduce follow-up questions and code revisions.

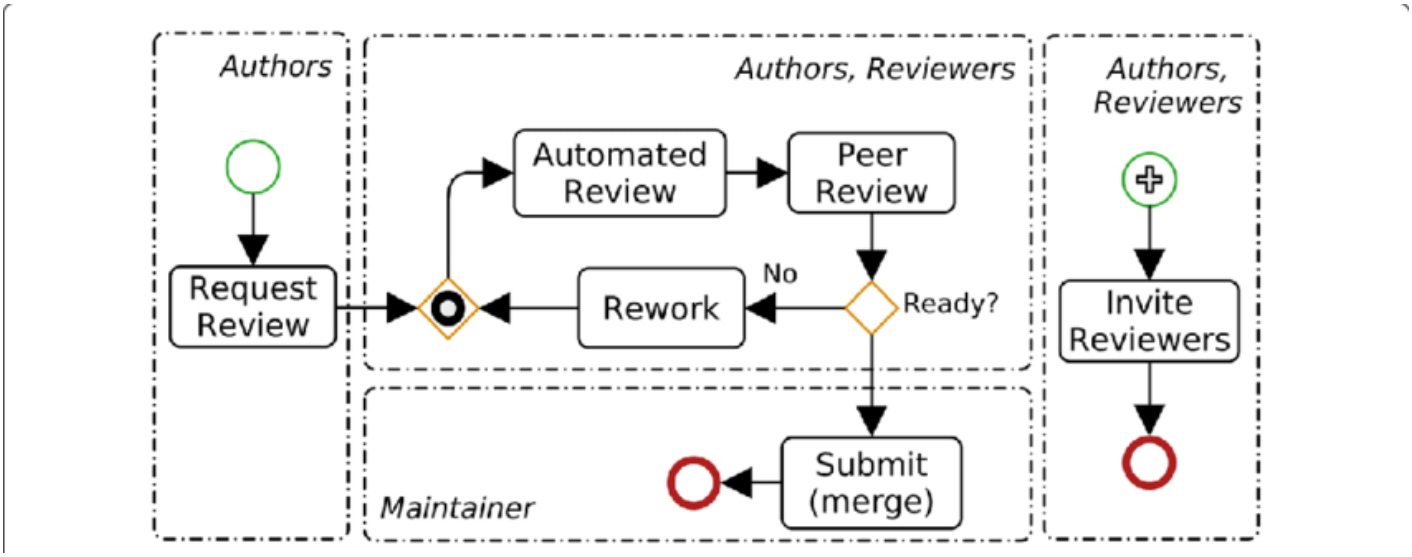

Figure 4. Code Review and Submission Workflow

In addition to controlled experiments, case studies from industry provide further support for NLP-based systems. For example, a large open-source project integrated an NLP-based reviewer into their GitHub pull request workflow. Over six months, they observed a 25% reduction in the average number of review iterations and a noticeable improvement in the consistency of code style across contributors. Feedback from contributors highlighted that the system helped them focus more on architectural and logical improvements rather than syntactic or stylistic issues. Companies such as Microsoft and Amazon have also piloted internal NLP systems to aid their development teams. These tools offer natural language feedback, identify duplicated logic, and suggest refactoring opportunities. Across the board, results indicate not only improved efficiency but also higher developer satisfaction due to reduced cognitive load during reviews. These experiments and real-world examples collectively illustrate that NLP-based code review systems can meaningfully enhance development workflows, reduce time and effort, and raise the overall quality of software.

5.2. Comparison with Traditional Methods in Terms of Speed, Accuracy, and Efficiency

When comparing NLP-based code review systems with traditional manual methods, several key factors stand out: speed, accuracy, and overall efficiency. Traditional code reviews rely on human reviewers to manually read, interpret, and critique code. This process is inherently time-consuming and often inconsistent due to reviewer fatigue, subjective judgment, or knowledge gaps. In contrast, NLP-based systems can process code almost instantly, providing structured and objective feedback in real-time. **Speed** is one of the most noticeable benefits. An NLP system can analyze thousands of lines of code within seconds, highlighting issues ranging from variable naming inconsistencies to potential bugs. Manual reviewers might spend hours reviewing the same amount of code, especially in large pull requests or unfamiliar codebases. In organizations with frequent commits and short release cycles, this speed advantage can be transformative. In terms of **accuracy**, traditional reviews excel at catching complex logic errors or architectural flaws, where human understanding and domain knowledge are essential. However, they often miss repetitive or superficial issues due to oversight or fatigue. NLP systems, trained on large codebases and language models, are particularly adept at identifying syntactic mistakes, poor documentation, outdated API usage, and missing edge case handling. By handling these repetitive checks, NLP systems free up human reviewers to focus on more nuanced problems.

Table 4. Comparative Table: NLP vs. Manual Code Review

Criteria	Manual Code Review	NLP-Based Code Review
Speed	Slower; depends on reviewer availability	Faster; automated analysis in real-time
Accuracy	High for complex logic; may miss minor issues	High for syntax and style; may miss complex logic
Consistency	Subjective; varies between reviewers	Consistent; follows predefined rules and patterns
Scalability	Limited; depends on team size	Highly scalable; handles large codebases efficiently
Feedback Clarity	Varied; depends on reviewer communication	Clear and structured; standardized feedback format
Developer Satisfaction	May cause cognitive overload	Reduces cognitive load; provides actionable insights

Efficiency is a blend of the two how effectively the system converts review time into actionable insights. NLP-based systems provide scalable, round-the-clock assistance without requiring the availability of specific team members. They integrate easily with continuous integration (CI) pipelines, enabling automated feedback during code submissions. This reduces the review backlog and

ensures a smoother development pipeline. Moreover, NLP systems can be tailored and trained to align with a team's specific coding standards and past review data, improving contextual relevance over time. They can also offer multilingual and multi-language support, benefiting teams with diverse backgrounds or working across different programming ecosystems. In summary, while NLP tools may not yet replace human reviewers entirely, they significantly augment the process by improving speed, increasing accuracy on repetitive tasks, and enhancing overall efficiency. The synergy between automated and manual reviews can lead to a more robust and scalable code quality assurance process.

6. Conclusion

In conclusion, this paper highlights the transformative potential of automating the code review process using Natural Language Processing (NLP) models. By implementing a structured methodology that involves collecting and preprocessing code data, training advanced models like CodeBERT, and integrating these systems into existing development pipelines, we have demonstrated that NLP can significantly enhance the efficiency, speed, and consistency of code reviews when compared to traditional manual methods. The results suggest that NLP-driven approaches can reduce the burden of repetitive tasks on developers, streamline collaboration, and help catch issues earlier in the development lifecycle. Despite these promising outcomes, certain challenges remain, particularly around the models' ability to fully grasp complex code semantics, logical flow, and the nuances of different programming paradigms. Addressing these limitations represents a compelling direction for future research, with potential strategies including the integration of more sophisticated representations like abstract syntax trees (ASTs) or control flow graphs (CFGs) to deepen models' contextual understanding.

Additionally, expanding support for a broader range of programming languages and enhancing the explainability of model outputs to make automated feedback more interpretable and actionable for developers are important goals. Future research might also explore ways to enrich the feedback provided by NLP systems, enabling them not only to detect bugs and code smells but also to offer proactive suggestions for improving code structure, design, and maintainability. Such advancements would move these tools from being purely diagnostic to becoming intelligent collaborators in the software development process. The broader impact on the software industry could be substantial, as automated code review tools powered by NLP stand to reduce development bottlenecks, enhance code quality, and foster more scalable engineering practices. As the technology continues to evolve and mature, it is poised to become an indispensable part of modern development workflows, allowing teams to focus more on innovation and less on routine verification tasks. Ultimately, the integration of NLP into code review processes represents a significant step toward more intelligent, efficient, and future-ready software engineering.

References

[1] Bacchelli, A., & Bird, C. (2013). *Expectations, outcomes, and challenges of modern code review*. Proceedings of the 2013 International Conference on Software Engineering (ICSE), 712-721.

[2] Bhagath Chandra Chowdari Marella, "From Silos to Synergy: Delivering Unified Data Insights across Disparate Business Units", International Journal of Innovative Research in Computer and Communication Engineering, vol.12, no.11, pp. 11993-12003, 2024.

[3] Kotte, K. R., & Panyaram, S. (2025). Supply Chain 4.0: Advancing Sustainable Business. Driving Business Success Through Eco-Friendly Strategies, 303.

[4] Jiang, Z. M., & Hassan, A. E. (2017). *Automated software defect prediction: A survey*. IEEE Transactions on Software Engineering, 43(7), 641-663.

[5] Tian, Y., Lo, D., & Sun, C. (2018). *Automatic bug assignment using multi-label classifiers*. Empirical Software Engineering, 23(1), 42-83.

[6] S. Panyaram, "Automation and Robotics: Key Trends in Smart Warehouse Ecosystems," International Numeric Journal of Machine Learning and Robots, vol. 8, no. 8, pp. 1-13, 2024.

[7] Chen, X., et al. (2020). *Automatic code review by learning code semantic and reviewer behaviors*. IEEE Transactions on Software Engineering.

[8] A. K. K, G. C. Vegineni, C. Suresh, B. C. Chowdari Marella, S. Addanki and P. Chimwal, "Development of Multi Objective Approach for Validation of PID Controller for Buck Converter," *2025 First International Conference on Advances in Computer Science, Electrical, Electronics, and Communication Technologies (CE2CT)*, Bhimtal, Nainital, India, 2025, pp. 1186-1190, doi: 10.1109/CE2CT64011.2025.10939724.

[9] Sudheer Panyaram, (2025/5/18). Intelligent Manufacturing with Quantum Sensors and AI A Path to Smart Industry 5.0. International Journal of Emerging Trends in Computer Science and Information Technology. 140-147.

[10] Ashima Bhatnagar Bhatia Padmaja Pulivarthi, (2024). Designing Empathetic Interfaces Enhancing User Experience Through Emotion. Humanizing Technology With Emotional Intelligence. 47-64. IGI Global.

[11] Hata, H., Akiyama, Y., & Kusumoto, S. (2019). *Predicting the reviewers for code changes*. Proceedings of the 41st International Conference on Software Engineering (ICSE), 1034-1045.
[12] Vegineni, Gopi Chand, and Bhagath Chandra Chowdari Marella. "Integrating AI-Powered Dashboards in State Government Programs for Real-Time Decision Support." *AI-Enabled Sustainable Innovations in Education and Business*, edited by Ali Sorayyaei Azar, et al., IGI Global, 2025, pp. 251-276. https://doi.org/10.4018/979-8-3373-3952-8.ch011
[13] Chib, S., Devarajan, H. R., Chundru, S., Pulivarthy, P., Isaac, R. A., & Oku, K. (2025, February). Standardized Post-Quantum Cryptography and Recent Developments in Quantum Computers. In 2025 First International Conference on Advances in Computer Science, Electrical, Electronics, and Communication Technologies (CE2CT) (pp. 1018-1023). IEEE.
[14] Vasdev K. "The Future of GIS in Energy Transition: Applications in Oil and Gas Sustainability Initiatives". *J Artif Intell Mach Learn & Data Sci 2023*, 1(2), 1912-1915. DOI: doi.org/10.51219/JAIMLD/kirti-vasdev/423
[15] Wang, S., & Lo, D. (2014). *Mining API usages from open source repositories for code completion*. Proceedings of the 36th International Conference on Software Engineering, 391-401.
[16] C. C. Marella and A. Palakurti, "Harnessing Python for AI and machine learning: Techniques, tools, and green solutions," In Advances in Environmental Engineering and Green Technologies, IGI Global, 2025, pp. 237–250
[17] Venu Madhav Aragani and Mohanarajesh Kommineni Sudheer Panyaram,Sunil Kumar Sehrawat, Swathi Chundru, Praveen Kumar Maroju, (2025), AI and Robotics: A Symbiotic Relationship in Digital Manufacturing, IEEE.
[18] Pulivarthy, P. (2022). Performance tuning: AI analyse historical performance data, identify patterns, and predict future resource needs. International Journal of Innovations in Applied Sciences and Engineering, 8(1), 139–155.
[19] Guo, S., et al. (2021). *NLP-powered code review: Automating patch classification and reviewer recommendation*. IEEE Transactions on Software Engineering.
[20] Padmaja Pulivarthy. (2024/12/3). Harnessing Serverless Computing for Agile Cloud Application Development," FMDB Transactionson Sustainable Computing Systems. 2,(4), 201-210, FMDB.
[21] B. C. C. Marella, "Data Synergy: Architecting Solutions for Growth and Innovation," International Journal of Innovative Research in Computer and Communication Engineering, vol. 11, no. 9, pp. 10551–10560, Sep. 2023.
[22] Sudheer Panyaram, (2025). Optimizing Processes and Insights: The Role of AI Architecture in Corporate Data Management. IEEE.
[23] Marella, Bhagath Chandra Chowdari, and Gopi Chand Vegineni. "Automated Eligibility and Enrollment Workflows: A Convergence of AI and Cybersecurity." *AI-Enabled Sustainable Innovations in Education and Business*, edited by Ali Sorayyaei Azar, et al., IGI Global, 2025, pp. 225-250. https://doi.org/10.4018/979-8-3373-3952-8.ch010
[24] Enhancement of Wind Turbine Technologies through Innovations in Power Electronics, Sree Lakshmi Vineetha Bitragunta, IJIRMPS2104231841, Volume 9 Issue 4 2021, PP-1-11.
[25] P. Pulivarthy Enhancing Data Integration in Oracle Databases: Leveraging Machine Learning for Automated Data Cleansing, Transformation, and Enrichment International Journal of Holistic Management Perspectives, 4 (4) (2023), pp. 1-18
[26] Kirti Vasdev. (2019). "GIS in Disaster Management: Real-Time Mapping and Risk Assessment". International Journal on Science and Technology, 10(1), 1–8. https://doi.org/10.5281/zenodo.14288561
[27] Sree Lakshmi Vineetha Bitragunta, 2022. "Field-Test Analysis and Comparative Evaluation of LTE and PLC Communication Technologies in the Context of Smart Grid", ESP Journal of Engineering & Technology Advancements 2(3): 154-161.
[28] S. Panyaram, "Digital Transformation of EV Battery Cell Manufacturing Leveraging AI for Supply Chain and Logistics Optimization," International Journal of Innovations in Scientific Engineering, vol. 18, no. 1, pp. 78-87, 2023.
[29] Mr. G. Rajassekaran Padmaja Pulivarthy,Mr. Mohanarajesh Kommineni,Mr. Venu Madhav Aragani, (2025), Real Time Data Pipeline Engineering for Scalable Insights, IGI Global.
[30] RK Puvvada . "SAP S/4HANA Finance on Cloud: AI-Powered Deployment and Extensibility" - IJSAT-International Journal on Science and …16.1 2025 :1-14.
[31] Praveen Kumar Maroju, Venu Madhav Aragani (2025). Predictive Analytics in Education: Early Intervention and Proactive Support With Gen AI Cloud. Igi Global Scientific Publishing 1 (1):317-332.
[32] Kodi, D. (2024). "Performance and Cost Efficiency of Snowflake on AWS Cloud for Big Data Workloads". International Journal of Innovative Research in Computer and Communication Engineering, 12(6), 8407–8417. https://doi.org/10.15680/IJIRCCE.2023.1206002
[33] Venu Madhav Aragani, Venkateswara Rao Anumolu, P. Selvakumar, "Democratization in the Age of Algorithms: Navigating Opportunities and Challenges," in Democracy and Democratization in the Age of AI, IGI Global, USA, pp. 39-56, 2025.
[34] Swathi Chundru, Lakshmi Narasimha Raju Mudunuri, "Developing Sustainable Data Retention Policies: A Machine Learning Approach to Intelligent Data Lifecycle Management," in Driving Business Success Through EcoFriendly Strategies, IGI Global, USA, pp. 93-114, 2025.
[35] Barigidad, S. (2025). Edge-Optimized Facial Emotion Recognition: A High-Performance Hybrid Mobilenetv2-Vit Model. *International Journal of AI, BigData, Computational and Management Studies*, *6*(2), 1-10. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V6I2P101

[36] Anumolu, V. R., & Marella, B. C. C. (2025). Maximizing ROI: The Intersection of Productivity, Generative AI, and Social Equity. In Advancing Social Equity Through Accessible Green Innovation (pp. 373-386). IGI Global Scientific Publishing.
[37] Sudheer Panyaram, (2025/5/18). Intelligent Manufacturing with Quantum Sensors and AI A Path to Smart Industry 5.0. International Journal of Emerging Trends in Computer Science and Information Technology. 140-147.
[38] Mohanarajesh Kommineni. (2023/6). Investigate Computational Intelligence Models Inspired By Natural Intelligence, Such As Evolutionary Algorithms And Artificial Neural Networks. Transactions On Latest Trends In Artificial Intelligence. 4. P30. Ijsdcs.
[39] Puvvada, Ravi Kiran. "Industry-Specific Applications of SAP S/4HANA Finance: A Comprehensive Review." *International Journal of Information Technology and Management Information Systems(IJITMIS)* 16.2 (2025): 770-782.
[40] Animesh Kumar, "Redefining Finance: The Influence of Artificial Intelligence (AI) and Machine Learning (ML)", Transactions on Engineering and Computing Sciences, 12(4), 59-69. 2024.
[41] S. Panyaram, "Connected Cars, Connected Customers: The Role of AI and ML in Automotive Engagement," International Transactions in Artificial Intelligence, vol. 7, no. 7, pp. 1-15, 2023.
[42] Kirti Vasdev. (2025). "Enhancing Network Security with GeoAI and Real-Time Intrusion Detection". International Journal on Science and Technology, 16(1), 1–8. https://doi.org/10.5281/zenodo.14802799
[43] Pulivarthy, P. (2023). Enhancing Dynamic Behaviour in Vehicular Ad Hoc Networks through Game Theory and Machine Learning for Reliable Routing. International Journal of Machine Learning and Artificial Intelligence, 4(4), 1-13.
[44] Khan, S., Uddin, I., Noor, S. et al. "N6-methyladenine identification using deep learning and discriminative feature integration". BMC Med Genomics 18, 58 (2025). https://doi.org/10.1186/s12920-025-02131-6.
[45] Vootkuri, C. Dynamic Threat Modeling For Internet-Facing Applications in Cloud Ecosystems.
[46] Settibathini, V. S., Kothuru, S. K., Vadlamudi, A. K., Thammreddi, L., & Rangineni, S. (2023). Strategic analysis review of data analytics with the help of artificial intelligence. International Journal of Advances in Engineering Research, 26, 1-10.