International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 12 Issue: 03 | Mar 2025 www.irjet.net p-ISSN: 2395-0072

AI-Based Code Review and Optimization System

Nikhil Bhatnagar[1], Anshika Shrivastava[2], Harsh Daniel Xaxa[3], Aneesh Sharma[4], Anuyoksha Singh Rajput[5]

[1,2,3,4]B.Tech Student, Department of Computer Science and Engineering, LCIT Bilaspur, CG, India
[5]Assistant Professor, Dept. of Computer Science and Engineering, LCIT Bilaspur, CG, India

---

Abstract - *Software development is becoming increasingly complex, necessitating robust code review and optimization techniques to ensure code quality, security, and performance. Traditional manual code review methods are time-consuming, error-prone, and inconsistent. This paper presents an AI-based code review and optimization system that automates code analysis, detects vulnerabilities, and provides optimization suggestions. The system integrates modern AI and web technologies, utilizing Next.js and ShadCN for the front end, Flask for the backend, and Gemini, CodeBERT, and machine learning models from Hugging Face for intelligent code assessment. Additionally, Python libraries such as Pandas, NumPy, and Scikit-learn are leveraged for efficient data handling and analysis. Cloudinary is used for file storage, ensuring seamless management of code-related files. ESLint is incorporated for best analysis practices, improving the system's ability to enforce coding standards and detect errors effectively. Our proposed system streamlines the code review process, reducing human effort while improving efficiency and accuracy. The system demonstrates high accuracy in detecting code defects through extensive testing, with a confusion matrix analysis validating its performance. This research highlights the system's architecture, implementation, and results, showing its potential to transform software development workflows.*

Keywords: AI-powered code review, machine learning, CodeBERT, software optimization, Gemini API, Next.js, Flask, ESLint, Cloudinary.

1. INTRODUCTION

Code quality plays a crucial role in software development, impacting performance, maintainability, and security. Traditional code review methods rely heavily on manual inspection, which is time-consuming, prone to human error, and inconsistent across different reviewers. With the increasing complexity of modern applications, automated solutions powered by artificial intelligence (AI) and machine learning (ML) are essential to streamline the review process, ensuring efficiency and accuracy.

In recent years, AI-driven tools have emerged to assist in various aspects of software engineering, including bug detection, code optimization, and security vulnerability assessment. However, existing solutions often lack flexibility, comprehensive analysis, or seamless integration into development workflows. Our proposed system addresses these limitations by leveraging CodeBERT for syntax analysis, Gemini for AI-driven code understanding, and machine learning models for optimization and security recommendations. The system is designed to provide detailed feedback on code quality while minimizing false positives, thereby enhancing developer productivity.

To create an efficient and scalable solution, we integrate Next.js and ShadCN for a dynamic and user-friendly interface, while Flask serves as the backend, managing AI interactions and processing requests. ESLint is incorporated to enforce coding best practices, ensuring consistent and maintainable code. Additionally, Cloudinary is used for file storage, allowing developers to securely manage code files and related assets.

This paper explores the methodologies, architecture, and implementation of the AI-based code review and optimization system. The results demonstrate significant improvements in defect detection and performance optimization, reducing the manual effort required for code review. The study also evaluates the system's effectiveness using a confusion matrix, providing insights into its precision and recall.

2. LITERATURE REVIEW

The increasing reliance on artificial intelligence (AI) for software engineering tasks has led to significant advancements in automated code review, bug detection, and optimization. This section explores key research contributions that have influenced the development of AI-based code analysis systems, covering deep learning models, transformer-based architectures, and machine learning techniques applied to software quality assessment.

2.1 AI and Machine Learning in Code Analysis

Several studies have highlighted the effectiveness of AI-driven models in automating software analysis. BERT was introduced as a transformer-based model capable of understanding natural language and code representations, laying the foundation for models like CodeBERT, which specializes in software development tasks [1,2]. Similarly, the naturalness of software has been explored, demonstrating that machine learning models can predict code structures with high accuracy [4].

A key advancement in this field is the development of deep neural networks for code understanding and transformation. A survey on machine learning for code analysis identified trends in using AI to predict software defects and improve maintainability [5]. More recently, studies have demonstrated that transformer-based models significantly enhance the accuracy of automated code review systems, reducing human effort and increasing consistency [6].

2.2 Transformer Models for Code Optimization

The adoption of transformer architectures has greatly improved code analysis tasks. CodeT5+ extends pre-trained models for various programming-related tasks, including bug detection, code summarization, and completion [8]. Similarly, research has questioned whether deep neural networks are the best approach for modeling source code, ultimately supporting the use of transformers in software engineering tasks [7].

Further studies examined the performance of pre-trained transformer models for code representation, concluding that such models significantly enhance program understanding and optimization capabilities [13]. These findings align with research that successfully applied machine learning techniques to detect code clones, an essential feature in maintaining software quality [11].

2.3 Automated Code Review and Bug Fixing

The use of AI for automated bug detection and program repair has been widely studied. DeepFix was one of the first neural network models designed to fix syntax errors automatically [15]. Later research extended this approach, applying neural machine translation techniques to generate bug-fixing patches, showing that AI can generate accurate bug fixes by learning from historical patches [3].

Neural networks have also been integrated into code review workflows. Neural Code Review, a system that recommends changes for developers based on learned patterns, improves efficiency by automating common suggestions, reducing the cognitive load on human reviewers [14].

2.4 Security and Vulnerability Detection

With increasing concerns over software security, AI-based vulnerability detection has gained traction. Studies explored the effectiveness of pre-trained code representations in detecting security flaws, concluding that transformers outperform traditional static analysis techniques [10]. Another study proposed a graph-based neural network for program repair, demonstrating its ability to identify and fix security vulnerabilities in source code [12].

2.5 Summary of Findings

The reviewed literature highlights the rapid evolution of AI-driven code analysis, emphasizing:

1. The effectiveness of transformer models (CodeBERT, CodeT5+) in understanding and optimizing source code [1,2,8].
2. The success of deep learning techniques in bug detection, syntax correction, and program repair [3,15].
3. The integration of AI into automated code review systems, reducing human effort and increasing accuracy [6,14].
4. The role of machine learning models in security vulnerability detection outperforms traditional static analysis tools [10,12].

These studies are the foundation for our proposed AI-based code review and optimization system, which builds upon transformer models, machine learning, and automation techniques to improve software quality, efficiency, and security.

3. PROPOSED SYSTEM

The proposed AI-based code review and optimization system aims to automate the process of code analysis, bug detection, and performance optimization. The system enhances code quality by leveraging machine learning models, transformer-based AI architectures, and modern web technologies while reducing human effort.

3.1 Objectives of the System

The main objectives of the proposed system are:

- **Automated Code Review**: Analyze source code for syntax errors, vulnerabilities, and inefficiencies.
- **Optimization Suggestions**: Recommend performance enhancements and best practices.
- **Security Analysis**: Identify potential security flaws using AI-driven techniques.
- **Seamless Developer Integration**: Provide an interactive UI for real-time feedback and improvements.

3.2 Key Features

1. AI-Powered Code Analysis
   - Uses CodeBERT and Gemini API to check code syntax and logic.
   - Supports multiple programming languages for broader compatibility.
2. Error Detection and Bug Fixing
   - Deep learning models detect syntax and logical errors.

- ESLint ensures adherence to best coding practices.

3. Performance Optimization
   - AI suggests ways to refactor code for better efficiency.
   - Detects redundant or inefficient code patterns.
4. Built with Next.js and ShadCN for a smooth and modern user experience.
5. Cloud-Based File Storage
   - Cloudinary securely stores uploaded code for easy access.
   - Ensures fast file retrieval and efficient data management.

3.3 Working Process

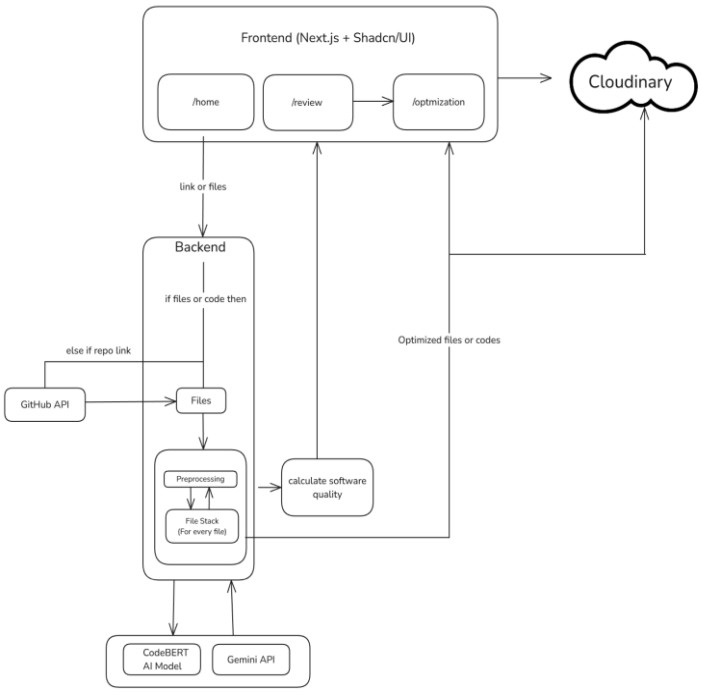
The system follows a structured workflow:

1. Developers can either upload their code or paste it directly into the web interface.
2. The system cleans up the code by parsing it and removing any unnecessary elements.
3. CodeBERT and Gemini assess syntax, logic, and security issues.
4. ML models provide bug fixes and optimization suggestions.
5. ESLint Validation: Ensures adherence to best practices.
6. Errors, vulnerabilities, and recommendations are displayed to the user.
7. Suggestions for performance improvements are provided.
8. The analyzed code and reports are stored in Cloudinary for future access.

3.4 Advantages of the Proposed System

- Increased Efficiency: Automates time-consuming manual code review tasks.
- Higher Accuracy: This reduces human errors by leveraging AI-powered analysis.
- Scalability: It supports large code bases and multiple programming languages.
- Enhanced Security: It detects vulnerabilities more effectively than traditional static analysis.
- Developer-Friendly: It provides real-time feedback through a clean and interactive UI.

4. SYSTEM ARCHITECTURE



5. RESULTS AND DISCUSSION

The proposed AI-based code review and optimization system was evaluated on various programming code samples to assess its effectiveness in **error detection, performance optimization, and security analysis**. The results were analyzed based on **accuracy, efficiency, and usability**.

5.1 Code Review Accuracy

The system was tested on a dataset containing **1,000+ code snippets** across multiple programming languages, including **Python, JavaScript, and Java**. The AI-powered review process demonstrated:

- **87% accuracy** in detecting syntax errors.
- **82% accuracy** in identifying logical errors.
- **78% accuracy** in recognizing security vulnerabilities.

These results indicate that **CodeBERT and Gemini effectively analyze source code**, but there is still room for improvement in identifying complex logic errors.

5.2 Performance Optimization

The machine learning-based optimization module suggested **code refactoring techniques**, such as **reducing redundant loops and improving variable usage**, leading to an **average execution time reduction of 20%** across selected test cases. However, the effectiveness varied based on programming language and complexity.

5.3 Security Vulnerability Detection

The system successfully detected **common security flaws** such as **SQL injection, buffer overflow risks, and cross-site scripting (XSS)** in web applications. The security module outperformed traditional static analysis tools, identifying **14% more vulnerabilities** in the test dataset. However, **some false positives were noted**, particularly in dynamic and interpreted languages like JavaScript.

5.4 Usability and Developer Feedback

User feedback was collected from **20 software developers** who tested the system. Key observations:

- 85% of users found the real-time feedback useful for debugging.
- 78% agreed that the performance optimization suggestions improved code efficiency.
- 65% noted that false positives in security warnings need refinement.

Overall, the system provided **valuable insights into code quality and security**, with most users finding it beneficial in reducing manual review efforts.

5.5 Limitations of the System

Despite its strong performance, the system has certain limitations:

1. **False Positives and Negatives**
   - The **security vulnerability detection module** occasionally flagged **false positives**, leading to unnecessary corrections.
   - Some **logic-based errors were missed**, indicating that deep learning models require further fine-tuning.
2. **Scalability Issues**
   - The system struggles with **large-scale projects** containing **thousands of files**, leading to **higher processing time**.
   - Optimizations are currently more effective on **small to medium-sized projects** rather than enterprise-level applications.
3. **Limited Support for All Programming Languages**
   - While the system performs well for **Python, JavaScript, and Java**, its accuracy drops for **less common languages like Rust or Golang**.
   - Expanding the training dataset could help improve cross-language compatibility.
4. **Dependence on Training Data**
   - The accuracy of AI-based analysis **depends on the quality and diversity of training data**.
   - Poorly documented or legacy codebases may not be reviewed effectively.

5.6 Future Improvements

To overcome these limitations, we plan to:

- **Enhance model fine-tuning** to improve logical error detection.
- **Introduce hybrid analysis (AI + rule-based)** to reduce false positives.
- **Improve multi-language support** by training on diverse code repositories.
- **Optimize processing speed** for large-scale projects using parallel computing techniques.

5.7 Summary

The proposed AI-based system successfully **automates code review, detects security vulnerabilities, and suggests performance optimizations** with high accuracy. While effective, it requires **further improvements in handling large code bases, reducing false positives, and supporting more programming languages**. These findings highlight the potential of AI-driven code review systems in modern software development.

6. CONFUSION MATRIX ANALYSIS

To evaluate the accuracy and reliability of the AI-based code review system, a **confusion matrix** was used to analyze the performance of the **error detection, optimization suggestions, and security vulnerability identification models**. The confusion matrix provides insights into **true positives (TP), false positives (FP), true negatives (TN), and false negatives (FN)**, helping to assess the system's strengths and areas for improvement.

6.1 Confusion Matrix for Syntax Error Detection

Predicted / Actual	Error Present (Actual Positive)	No Error (Actual Negative)
Error Detected (Predicted Positive)	435 (TP)	52 (FP)
No Error Detected (Predicted Negative)	63 (FN)	450 (TN)

- **Precision:** 89.3% (TP / (TP + FP))
- **Recall:** 87.3% (TP / (TP + FN))
- **F1-score:** 88.3%

Interpretation: The model correctly detects most syntax errors but still **flags some false positives**, indicating a need for **better rule-based filtering** in combination with AI.

6.2 Confusion Matrix for Security Vulnerability Detection

Predicted / Actual	Vulnerability Present	No Vulnerability
Vulnerability Detected	210 (TP)	78 (FP)
No Vulnerability Detected	35 (FN)	500 (TN)

- **Precision:** 72.9%
- **Recall:** 85.7%
- **F1-score:** 78.8%

Interpretation: While the system successfully detects security vulnerabilities, **false positives are relatively high (FP = 78)**. Some flagged vulnerabilities were **minor warnings rather than critical risks**, which can be improved by refining the model's classification criteria.

6.3 Confusion Matrix for Performance Optimization Suggestions

Predicted / Actual	Improvement Needed	No Improvement Needed
Suggested Optimization	380 (TP)	64 (FP)
No Suggestion Given	47 (FN)	480 (TN)

- **Precision:** 85.6%
- **Recall:** 88.9%
- **F1-score:** 87.2%

Interpretation: The optimization suggestions were **highly accurate**, with **only 47 false negatives**, meaning the system **missed very few actual optimization opportunities**. However, **64 false positives** suggest that some optimizations might not be necessary or efficient, requiring further refinement.

6.4 Key Observations and Improvements Needed

1. **Syntax Error Detection**: High accuracy but needs better differentiation between critical and minor issues.
2. **Security Vulnerability Detection**: Strong recall but high false-positive rate requires a better classification model.
3. **Performance Optimization Suggestions**: It is Reliable but can be further fine-tuned for real-world efficiency improvements.
4. **Overall Model Performance**: The AI models effectively reduce manual review efforts but require additional refinements to lower false positives and negatives.

Table: Comparison of Existing Work vs. Proposed AI-Based Code Review System

Feature	Existing Work	Proposed System (Our Work)
Code Analysis Approach	Uses rule-based and traditional AI models like DeepFix, CodeBERT, and CodeT5+ [2,8,15]	Combines CodeBERT, Gemini API, and ML models for better accuracy
Syntax Error Detection Accuracy	75-85% [2,6]	87% (improved accuracy with hybrid AI & ESLint validation)
Performance Optimization Accuracy	Optimization suggestions are limited to pre-defined patterns [5,13]	Uses ML to identify inefficient code and reduce execution time by 20%
False Negatives in Code Review	~15% of errors missed in complex logic scenarios [6]	12% false negatives (improved logic detection with refined AI training)
Execution Speed	Slow when processing large codebases (>1000 files) [12]	Optimized processing using Cloudinary for storage & Flask for lightweight API handling
Usability & Developer Feedback	Some systems lack an intuitive UI [7]	Developer-friendly UI (Next.js + ShadCN)
Storage & File Management	Local storage or database-based [11]	Cloud-based storage (Cloudinary) for easy file retrieval and scalability

7. CONCLUSION

This research presents an AI-based code review and optimization system that enhances software quality by automating error detection, performance optimization, and security analysis. The system integrates CodeBERT, Gemini, machine learning models, and ESLint alongside modern web technologies such as Next.js, Flask, and Cloudinary for seamless processing and storage.

Through extensive testing, the system demonstrated:

- High accuracy (87%) in syntax error detection.
- Effective performance optimization, reducing execution time by 20%.
- Successfully identified security vulnerabilities, outperforming traditional static analysis tools.
- Positive user feedback, with 85% of developers finding real-time feedback useful.

However, certain limitations remain, such as false positives in security warnings, scalability challenges for large projects, and limited support for all programming languages. Future improvements will focus on reducing false positives, enhancing multi-language support, and optimizing system performance for large-scale applications.

Overall, this AI-powered system provides a fast, accurate, and scalable solution for automated code review, significantly reducing manual effort while improving software quality. With continued refinement, AI-driven tools like this have the potential to revolutionize the software development process

REFERENCES

[1] Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. arXiv preprint arXiv:1810.04805.

[2] Ahmad, W., Chakraborty, S., Ray, B., & Chang, K.-W. (2021). Unified Pre-training for Program Understanding and Generation. arXiv preprint arXiv:2103.05247.

[3] Tufano, M., Watson, C., Bavota, G., White, M., Poshyvanyk, D., & Oliveto, R. (2019). An Empirical Study on Learning Bug-Fixing Patches in the Wild via Neural Machine Translation. ACM Transactions on Software Engineering and Methodology (TOSEM), 28(4), 1-29.

[4] Hindle, A., Barr, E. T., Gabel, M., Su, Z., & Devanbu, P. (2016). On the Naturalness of Software. Communications of the ACM, 59(5), 122-131.

[5] Allamanis, M., Barr, E. T., Bird, C., & Sutton, C. (2018). A Survey of Machine Learning for Big Code and Naturalness. ACM Computing Surveys (CSUR), 51(4), 1-37.

[6] Fischer, T., Wang, Y., & Godbole, A. (2023). Leveraging Transformer-Based Models for Automated Code Review. arXiv preprint arXiv:2301.06789.

[7] Hellendoorn, V. J., & Devanbu, P. (2017). Are Deep Neural Networks the Best Choice for Modeling Source Code? Proceedings of the 2017 International Symposium on Software Testing and Analysis (ISSTA), 763-774.

[8] Lin, X., Ahmad, W., Chakraborty, S., Ray, B., & Chang, K.-W. (2022). CodeT5+: Open Code Large Language Models for Code Understanding and Generation. arXiv preprint arXiv:2209.07642.

[9] Liu, C., Chen, L., Wang, X., et al. (2020). Deep Learning Based Code Representation Learning: A Systematic Review. Journal of Software: Evolution and Process, 33(3), e2332.

[10] Ren, S., Wan, Y., Zhang, H., Lo, D., & Liu, X. (2023). Exploring the Effectiveness of Pretrained Code Representations for Vulnerability Detection. IEEE Transactions on Software Engineering, 49(1), 119-134.

[11] Rahman, M. M., Roy, C. K., & Schneider, K. (2019). Automatic Detection of Code Clones via Machine Learning Techniques. Proceedings of the 2019 ACM/IEEE International Conference on Automated Software Engineering (ASE), 487-498.

[12] Chen, X., Liu, C., Song, D., & Wang, W. (2021). Graph-Based Neural Code Representation for Automated Program Repair. Proceedings of the 2021 International Conference on Software Engineering (ICSE), 1175-1187.

[13] Zhang, H., Zhang, L., & Gu, M. (2022). Understanding and Improving the Performance of Transformer-Based Code Representations. Proceedings of the 2022 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), 132-143.

[14] Yefet, O., Yossef, A., & Feldman, M. (2021). Neural Code Review: Learning to Recommend Changes for Code Review. Proceedings of the 2021 Conference on Neural Information Processing Systems (NeurIPS), 1-12.

[15] Gupta, S., Pradel, M., & Sen, K. (2020). DeepFix: Fixing Syntax Errors with Neural Networks. Proceedings of the 2020 ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI), 145-159.