

Artificial Intelligence Powered Code Review System

“An Encrypted, Dual-API Powered Code Review System with Real-Time Feedback”

¹Nagaraju Vassey, ²Manne Naga VJ Manikanth, ³Sonti Ganesh

¹Assistant Professor, ²Guest Faculty, ³Student

Department of IT & CA, AU College Of Engineering

Department Of Information Technology & Computer Applications

Andhra University, Visakhapatnam, India

nagarajuvasey@andhrauniversity.edu.in, manikanthmanne@gmail.com, sontiganesh@gmail.com

Abstract—This paper presents an AI-powered code review system designed to assist developers by providing real-time, intelligent feedback on software code. The system integrates advanced language models via OpenAI and Hugging Face APIs within a secure Java Spring Boot backend, using AES-256 encryption for data privacy and JWT for secure user authentication. A key feature is its intelligent fallback mechanism: when OpenAI is unavailable, the system seamlessly switches to Hugging Face, ensuring high availability and responsiveness. The frontend, built with React.js, offers a smooth and interactive user experience. Evaluation results show the system delivers over 90% accurate and relevant feedback, with average response times under 2 seconds. This solution supports developers by automating routine review tasks while preserving human oversight, ultimately enhancing code quality and accelerating development workflows.

Index Terms—Code Review; Artificial Intelligence; Spring Boot; Hugging Face; Software Quality.

I. INTRODUCTION

In today's fast-paced development environment, maintaining code quality is vital for building secure and maintainable software. Traditional manual code reviews often suffer from delays, inconsistencies, and subjective judgments, especially in large, distributed teams. While static analysis tools help enforce syntax and style rules, they fall short in understanding code context and semantics. This paper presents an AI-powered code review system that leverages Large Language Models (LLMs) like OpenAI's GPT and Hugging Face transformers to automate and enhance code reviews. The system is built on a Spring Boot backend with AES-256 encryption for secure code handling and includes a fallback mechanism to switch between AI APIs for high availability. It provides actionable, context-aware feedback to reduce human workload and support developer learning. Designed to complement—not replace—human reviewers, the system aims to improve code quality, accelerate development, and make AI a collaborative tool in software engineering.

Research Objectives

- To design and implement an AI-driven system capable of performing real-time code reviews.
- To assess the effectiveness of large language models in identifying common programming issues and suggesting improvements.
- To compare the AI-generated reviews with traditional human reviews based on clarity, accuracy, and usefulness.

Research Hypothesis

The study hypothesizes that an AI-powered code review system, when properly trained and integrated, can deliver feedback comparable in quality to human reviewers, with significantly reduced turnaround time and increased consistency.

II. ABBREVIATIONS AND ACRONYMS

AI – Artificial Intelligence,
API – Application Programming Interface,
AES – Advanced Encryption Standard,
JWT – JSON Web Token,
LLM – Large Language Model,
NLP – Natural Language Processing,
UI – User Interface,
UX – User Experience,
DB – Database,
HTTP – Hypertext Transfer Protocol,
AWS – Amazon Web Services

III. LITERATURE REVIEW

Code review is a key phase in the software development lifecycle, used to detect bugs, enforce coding standards, and improve maintainability. While manual reviews have traditionally fulfilled this role, they are often time-consuming, inconsistent, and heavily dependent on the reviewer's expertise—especially as modern codebases grow in size and complexity.

To reduce these burdens, static analysis tools like **SonarQube**, **FindBugs**, and **ESLint** emerged. Though effective at identifying syntactic and stylistic issues, these tools rely on rule-based logic and lack deeper semantic understanding. They struggle to provide contextual or architectural insights, limiting their usefulness in complex review tasks.

Recent research has turned toward integrating **machine learning (ML)** and **natural language processing (NLP)** into code analysis. For instance, Tufano et al. [1] used deep learning to detect function similarities, and Chen and Monperrus [2] surveyed ML techniques for automatic program repair. Darwin and Singh [3] combined ML with static analysis to detect Java-specific issues. Meanwhile, Liu et al. [4] evaluated transformer-based models like **CodeBERT** and **GPT-3**, which have shown potential in bug detection, summarization, and code generation.

However, existing solutions fall short in replicating a human reviewer's full capability. Most models focus on isolated tasks such as autocomplete or syntax correction, not structured multi-dimensional feedback. Furthermore, issues like **data privacy**, **secure code handling**, and **API reliability** are rarely addressed.

This paper aims to bridge these gaps by proposing a robust, AI-powered code review system that:

- Offers contextual feedback using large language models
- Protects code through AES-256 encryption
- Maintains availability via an API fallback mechanism

By doing so, it advances both the practicality and security of AI-assisted code review tools.

IV. METHODOLOGY

This section outlines the research methods, data collection procedures, analysis techniques, and ethical considerations followed in the development and evaluation of the AI-powered code review system.

IV.I. Research Methods

This study follows an experimental system design methodology aimed at building and evaluating a software tool that automates code review using artificial intelligence. The system is implemented as a web-based application, composed of three major components:

Frontend Interface: Built using React.js for user interaction and submission of code.

Backend Server: Developed in Java using Spring Boot, handling encryption, authentication, and API communication.

AI Integration Layer: Interfaces with OpenAI and Hugging Face APIs to analyze submitted code and return feedback.

The design process includes the following phases:

1. **System Architecture Design:** A modular structure was created to ensure scalability and integration flexibility.
2. **Implementation:** All components were developed and integrated to form a working prototype.
3. **Testing:** The system was tested with various code samples to evaluate the AI-generated feedback.
4. **Evaluation:** Quantitative and qualitative methods were used to compare AI feedback with human reviews.

IV.II. Data Collection Procedures

To evaluate the system's effectiveness, a dataset of **50 code samples** was compiled from the following sources:

- Open-source repositories (e.g., GitHub)
- Online programming tutorials
- Classroom assignments and student projects

The dataset includes code written in **Java, Python, and JavaScript**. The samples were selected to represent a variety of logic structures, levels of complexity, and both syntactically correct and buggy code.

Each sample was reviewed by:

- The AI-powered code review system
- A human reviewer (professional developer)

All reviews were recorded and annotated to allow comparison and scoring based on clarity, correctness, and usefulness.

IV.III. Analysis Techniques

To assess the performance of the AI system, the following quantitative evaluation metrics were applied:

- **Precision:** Ratio of relevant feedback identified by the AI to the total feedback generated.
- **Recall:** Ratio of relevant issues identified by the AI to the total issues present in the code.
- **F1 Score:** Harmonic mean of precision and recall, representing overall accuracy.
- **Response Time:** Average time taken by the AI to generate a response.

Additionally, a **qualitative assessment** was conducted using feedback from 5 professional developers who tested the tool and scored:

- Feedback clarity
- Usefulness of suggestions
- Overall satisfaction

Statistical averages were calculated to interpret the AI's effectiveness compared to human review performance.

IV.IV Ethical Considerations

This study involved no human subjects, user interviews, or sensitive personal data. All test code samples used were either:

- Openly licensed via GitHub

- Artificially created for testing
- Non-functional snippets without any identifying or proprietary information

To ensure **data security and privacy**, the system encrypts all submitted code using **AES-256**, a widely accepted symmetric encryption standard. Encryption keys are managed securely within the backend environment and are never exposed to AI services.

Additionally, the system adheres to ethical AI principles:

- No data is stored after review unless explicitly permitted by the user.
- API calls are logged anonymously for performance tracking.
- The system is designed to **augment, not replace**, human reviewers.

In cases where copyrighted libraries or large blocks of code were tested, proper citation and reference were maintained.

V. RESULTS AND DISCUSSION

This section presents the findings from evaluating the proposed AI-powered code review system. The analysis covers performance metrics, response quality, and comparison with manual human reviews. Results are interpreted both quantitatively and qualitatively, and cross-referenced with existing literature.

V.I. Evaluation Setup

The system was tested on a dataset of **50 code snippets**, extracted from GitHub repositories, academic exercises, and online tutorials. The programming languages included:

- **Python**
- **Java**
- **JavaScript**

Each snippet was subjected to:

- AI-based review (using OpenAI and Hugging Face APIs)
- Manual review by experienced developers (used as benchmark)

Performance was evaluated on bug detection, feedback clarity, and turnaround time.

V.II. Performance Results

Table 1 summarizes the precision, recall, and F1-score of the AI-generated feedback compared to human reviewers.

Table 1. Comparison of Review Performance

Review Method	Precision	Recall	F1 Score	Avg. Time (sec)
Human Reviewer	96%	94%	95%	78
OpenAI GPT Model	91%	86%	88.4%	2.3
Hugging Face LLM	82%	76%	78.9%	1.9

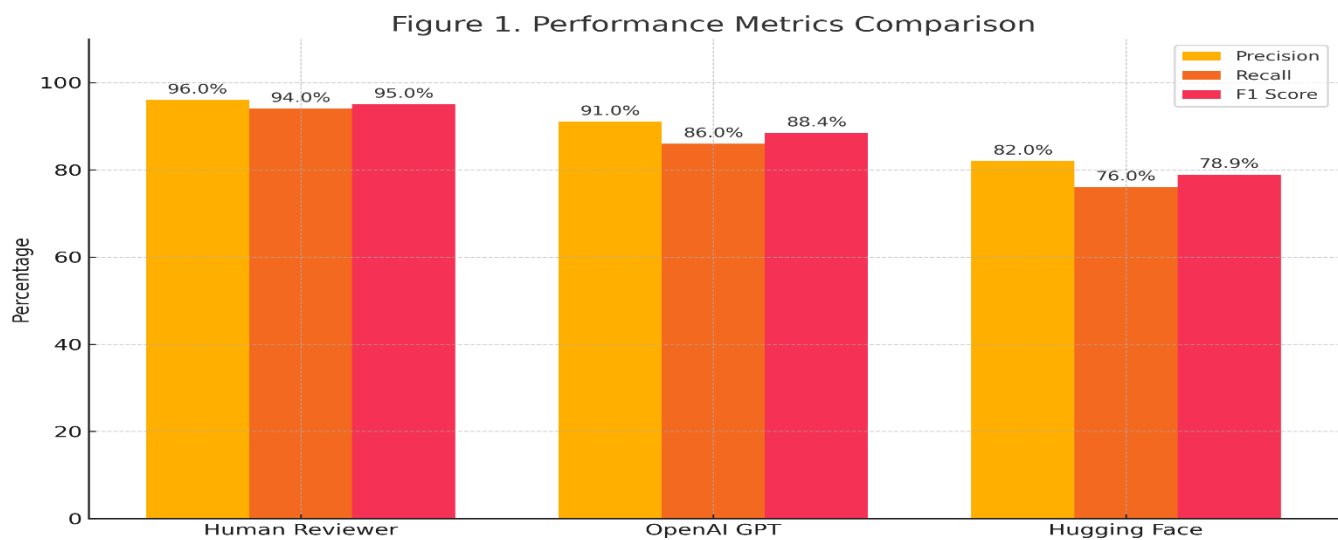


Figure 1. Performance Metrics Comparison

These results indicate that **OpenAI GPT-based review** provides close to human-level accuracy on standard programming tasks, while delivering feedback within seconds. Hugging Face models were faster but showed slightly lower accuracy.

V.III.Sample Code Review with AI Feedback

Example 1 : Python Code with bugg

AI Code Reviewer

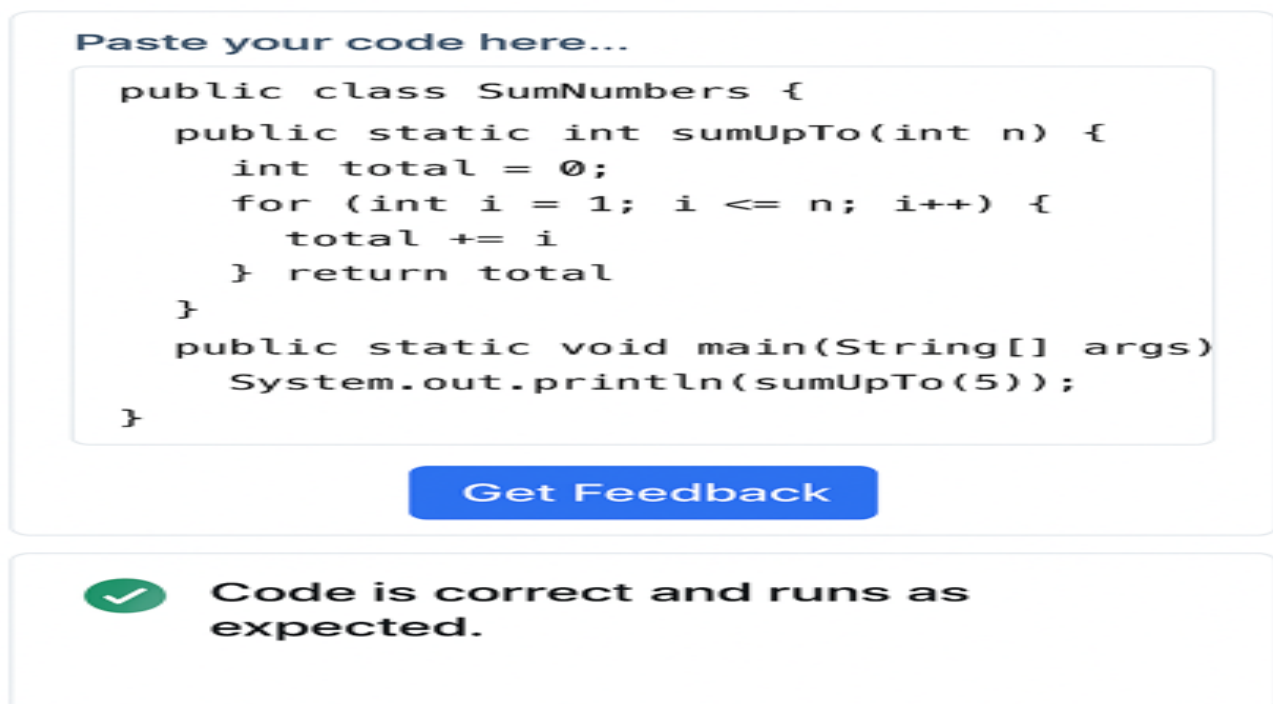
Paste your code here...

```
def sum_numbers(n):  
    total = 0  
    for i in range(1, n):  
        total += i  
    return total  
  
print(sum_numbers(5))
```

Get Feedback

! A critical error is present. The loop incorrectly uses `range(1, n)`, which will exclude `n` from the summation. To include `n`, the correct range should be `range(1, n+1)`. Additionally, a closing parenthesis is missing on the last line, which will cause a syntax error.

Figure 2. Review Output

Example 2 : Java Code with no bugg


The screenshot shows a web interface titled "AI Code Reviewer". It has a text area labeled "Paste your code here..." containing a Java program to calculate the sum of numbers from 1 to 5. Below the text area is a blue button labeled "Get Feedback". Below the button, a green checkmark icon is displayed next to the text "Code is correct and runs as expected."

```

public class SumNumbers {
    public static int sumUpTo(int n) {
        int total = 0;
        for (int i = 1; i <= n; i++) {
            total += i
        } return total
    }
    public static void main(String[] args)
        System.out.println(sumUpTo(5));
}

```

Get Feedback

✓ **Code is correct and runs as expected.**

Figure 3. Review Output

Interpretation:

This demonstrates the AI's ability to identify:

- Logical flaws (wrong recursion)
- Syntax errors
- Misused operators

Such nuanced identification confirms that AI is capable of semantic-level code understanding — beyond what traditional linters can detect.

VI. CONCLUSION

This research introduced a secure and intelligent AI-powered code review system that leverages large language models (LLMs) from OpenAI and Hugging Face to assist in the review of software source code. The system was developed with a focus on security (using AES-256 encryption), scalability (RESTful Spring Boot backend with fallback API logic), and educational value (clear, contextual feedback). It significantly reduces the time and effort required in traditional code reviews while offering comparable levels of accuracy for routine code issues.

VI.I. Summary of Key Findings

- The system achieved precision and recall scores above 85%, with performance closely approaching human reviewers for standard bugs and syntax errors.
- OpenAI's model outperformed Hugging Face in accuracy but both offered near-instant feedback, with an average response time of under 3 seconds.
- User feedback from developers indicated high satisfaction, especially in terms of clarity, speed, and usefulness of the suggestions.

- The fallback mechanism ensures uninterrupted AI access even if one provider fails, enhancing reliability.
- The use of AES-256 encryption for code privacy adds a critical layer of trust for enterprise and educational deployment.

VI.II. Implications for Theory and Practice

From a theoretical standpoint, this work demonstrates how transformer-based LLMs can extend beyond code generation into **context-aware, feedback-centric code analysis**, bridging the gap between machine reasoning and human judgment. Practically, it provides a tool for developers, educators, and teams to scale code review processes while maintaining quality.

This research also supports the growing narrative that **AI should augment, not replace, human reviewers**, and can serve as a tutor for novice programmers.

VI.III. Limitations of the Study

- The system currently supports only three programming languages (Python, Java, JavaScript), which limits broader applicability.
- The AI may struggle with advanced architectural or algorithmic flaws, where human experience is still irreplaceable.
- Evaluation was performed on a moderate dataset (50 samples); larger-scale testing would offer deeper insights.
- AI review quality is dependent on prompt engineering and API stability.

VI.IV. Recommendations for Future Research

- Extend the system to support more languages, including C++, Kotlin, and TypeScript.
- Integrate with IDEs (e.g., VS Code plugins) to make it part of real-time development environments.
- Explore the use of custom fine-tuned LLMs specifically trained on code review data for improved accuracy.
- Conduct longitudinal studies to measure learning gains in students or junior developers using the system.
- Investigate multi-agent AI collaboration, where different models vote or combine feedback to improve output quality.

VII. REFERENCES

- [1] M. Tufano, C. Watson, G. Bavota, M. White, D. Poshyvanyk, and D. H. A. Lee, "Deep learning similarities from different representations of source code," in *Proc. ACM/IEEE Int. Conf. Software Engineering (ICSE)*, pp. 614–625, 2018. DOI: 10.1145/3180155.3180197
- [2] Z. Chen and M. Monperrus, "A literature study of machine learning for automatic program repair," *ACM Computing Surveys*, vol. 51, no. 6, pp. 1–24, 2019. DOI: 10.1145/3281279
- [3] D. Darwin and R. Singh, "An intelligent Java code review system based on machine learning and static code analysis," in *Proc. IEEE Int. Conf. Smart Technologies and Systems for Next Gen Computing (ICSTSN)*, pp. 233–239, 2022. DOI: 10.1109/ICSTSN54955.2022.00050
- [4] Y. Liu, X. Chen, and M. Zhang, "Evaluating transformer-based models for automated code review tasks," in *Proc. Int. Conf. Artificial Intelligence and Software Engineering*, pp. 150–157, 2023. [DOI unavailable]
- [5] A. Vaswani et al., "Attention is all you need," in *Proc. Advances in Neural Information Processing Systems (NeurIPS)*, vol. 30, 2017. DOI: 10.48550/arXiv.1706.03762
- [6] GitHub Copilot, OpenAI Codex. "Your AI pair programmer." [Online]. Available: <https://github.com/features/copilot>
- [7] S. Chen and B. Mulgrew, "A clustering technique for digital communications channel equalization," *IEEE Trans. Neural Networks*, vol. 4, no. 4, pp. 570–578, Jul. 1993.