International Journal of Engineering Research and Science & Technology

www.ijerst.org

ISSN : 2319-5991

Vol. 22 No. 2(1) (2026)

ijerst.editor@gmail.com

editor@ijerst.com

ISSN 2319-5991 www.ijerst.com
© 2026 IJERST. All Rights Reserved
Peer Reviewed, Referred & Indexed Journal

Research Paper

GEN AI FOR PREDICTIVE CODE REVIEW AND BUG DETECTION

[1]D Anil Kumar, [2] S Joel Vishal Kumar,[3] S Sri Vinaya Reddy, [4] BK Sheerjana,[5] R Shashi Vardhan
[1]AssistantProfessor, [2345]Students
Department of AIML
Siddhartha Institute of Technology & Sciences, Narapally
dr.anildasari@siddhartha.org.in, 24tq1a66i0@siddhartha.co.in, 24tq1a66i1@siddhartha.co.in, 24tq1a66f7@siddhartha.co.in, 24tq1a66f1@siddhartha.co.in

Abstract

Generative Artificial Intelligence for Predictive Code Review and Bug Detection is an intelligent software engineering system developed to automate the process of analyzing source code, identifying bugs, detecting vulnerabilities, and improving code quality. With the rapid growth of software applications and modern development practices, traditional manual code review methods have become time-consuming, error-prone, and highly dependent on developer expertise. This project introduces an AI-powered solution that combines Large Language Models (LLMs) with static code analysis techniques to enhance software quality assurance and reduce development effort.

The proposed system utilizes advanced LLM technology through the Groq API powered by the LLaMA 3.3 70B model to perform semantic analysis of source code submitted by users. The AI model understands programming logic, coding patterns, syntax, and contextual relationships within the code to identify bugs, inefficient structures, security vulnerabilities, and coding best practice violations. The system supports multiple programming languages including Python, Java, C, C++, and JavaScript, making it suitable for developers working across different technology domains.

In addition to AI-powered analysis, the system integrates Pylint for static code analysis to detect syntax errors, style violations, unused variables, and structural code issues. The combination of Generative AI and static analysis improves the accuracy and reliability of bug detection and code review processes. The application is implemented using Streamlit to provide an interactive web-based interface where users can upload or paste code snippets, select programming languages, and receive detailed analytical feedback instantly.

I. Introduction

Software bugs and code quality issues remain some of the most critical challenges in modern software development. Defects in software systems can lead to security vulnerabilities, application crashes, financial losses, and poor user experience. Research studies indicate that software defects cost organizations billions of dollars annually due to debugging, maintenance, system downtime, and rework activities. As software applications continue to grow in complexity and scale, ensuring high-quality

and secure code has become increasingly important for development teams and organizations. Traditional code review processes mainly rely on manual peer review, where developers inspect source code to identify bugs, vulnerabilities, and coding standard violations. Although peer review is considered an essential software engineering practice, it has several limitations. Manual review processes are time-consuming, inconsistent, and heavily dependent on human expertise and availability. Cognitive biases, reviewer fatigue, and increasing project complexity can reduce the effectiveness and accuracy of manual code inspections. These limitations become more significant in agile development environments where rapid release cycles and continuous integration require faster and more scalable review processes.

To support developers, static analysis tools such as Pylint, ESLint, and Checkstyle are widely used to identify syntax errors, style violations, and structural code issues automatically. These tools improve code quality by enforcing predefined rules and coding standards. However, traditional static analyzers are primarily rule-based and lack the contextual understanding required to detect logical errors, architectural problems, inefficient coding patterns, and advanced security vulnerabilities. They are unable to fully understand the semantic meaning and intent of source code.

II. Literature Survey

The field of automated code review and bug detection has become an important research area in modern Software Engineering due to the increasing complexity of software systems and the growing demand for secure, high-quality applications. Researchers have developed several techniques and tools to automate the process of identifying software defects, improving code quality, and assisting developers during software development. This literature survey discusses traditional static analysis methods, machine learning-based bug detection systems, Large Language Models for code understanding, automated code review systems, and the research gap addressed by the proposed project.

Traditional Static Analysis Approaches

Traditional static analysis tools have been widely used for automated code quality assurance and software defect detection. Tools such as Pylint, ESLint, FindBugs, Checkstyle, and PMD analyze source code without executing the program. These tools parse source code into Abstract Syntax Trees (ASTs) and apply predefined rule-based checks to identify syntax errors, style violations, unused variables, and structural issues.

Static analysis approaches are computationally efficient and produce deterministic outputs, making them highly suitable for integration into Continuous Integration (CI) and DevOps pipelines. Research by Johnson et al. (2013) showed that developers appreciated the automated detection of common coding issues but often experienced alert fatigue due to high false positive rates. Traditional static analysis systems also lacked semantic understanding and were unable to detect logical errors, architectural issues, or complex security vulnerabilities effectively.

Machine Learning Approaches to Bug Detection

The introduction of Machine Learning techniques significantly improved automated bug detection systems by enabling data-driven analysis of source code. Early research by Zheng et al. (2006) demonstrated that supervised learning models could classify buggy and non-buggy code using software metrics, version control history, and code complexity features.

Later advancements in deep learning introduced models capable of understanding contextual relationships in code. Bidirectional Encoder Representations from Transformers (BERT), introduced by Devlin et al. (2018), enabled contextual understanding of natural language and inspired its adaptation to source code analysis tasks. Subsequently, models such as CodeBERT, developed by Microsoft Research, achieved state-of-the-art performance in code search, code summarization, and documentation generation tasks. These models demonstrated that transformer-based architectures could understand both programming languages and natural language descriptions effectively.

Large Language Models for Code

The emergence of Generative Artificial Intelligence and Large Language Models (LLMs) has transformed AI-assisted software development. Models trained on massive repositories of source code can generate syntactically correct and semantically meaningful code across multiple programming languages. OpenAI's Codex model, which powers GitHub Copilot, demonstrated the practical use of AI for code generation, completion, and intelligent programming assistance.

Meta AI Research later introduced the Code Llama family of models, specialized versions of LLaMA trained on code-specific datasets. These models showed impressive performance in tasks such as code completion, bug fixing, code infilling, and code explanation. LLMs provide contextual understanding of software logic and can identify patterns associated with bugs, vulnerabilities, and inefficient coding practices. Their ability to generate human-readable explanations makes them highly useful for educational and professional software development environments.

Automated Code Review Systems

Several research studies have specifically focused on automated code review systems using AI and deep learning techniques. Tufano et al. (2021) proposed a transformer-based system trained on code review comments collected from open-source repositories. The model successfully generated natural language review comments for code changes and assisted developers during pull request reviews.

Another important contribution was CodeReviewer (Li et al., 2022), a pre-trained model designed specifically for automated code review tasks. The system supported comment generation, review necessity prediction, and code refinement based on reviewer feedback. These AI-powered systems improved the efficiency of code review processes and reduced manual effort. However, most existing systems focused mainly on reviewing code changes or diffs rather than providing complete holistic code quality analysis, vulnerability detection, and multi-language support.

Research Gap and Motivation

Despite major advancements in AI-based code analysis and automated review systems, several limitations still exist in current solutions. Many existing systems require high computational resources, depend heavily on proprietary APIs, or support only a limited number of programming languages. Some tools also lack user-friendly interfaces and are difficult to integrate into educational or beginner-level development environments.

Traditional static analysis tools provide fast structural analysis but lack semantic understanding, while advanced AI-based systems may be computationally expensive and inaccessible for general users. Additionally, many automated review systems focus only on code completion or diff analysis rather than comprehensive bug detection, vulnerability analysis, and code quality improvement.

The proposed project addresses these limitations by developing an accessible, multi-language, AI-powered Predictive Code Review and Bug Detection system that combines LLM-based semantic analysis with traditional static analysis techniques. The system provides a user-friendly web interface, supports multiple programming languages, and delivers comprehensive feedback including bug detection, security analysis, corrected code suggestions, and quality ratings. This integration aims to improve software development efficiency, reduce debugging time, and enhance overall code quality for both educational and professional use.

III. System Analysis

The Gen AI-based Predictive Code Review and Bug Detection system is designed to automate software code analysis, bug detection, and code quality evaluation using intelligent AI-powered techniques. The system combines Generative Artificial Intelligence, Large Language Models, and static code analysis tools to assist developers in identifying coding errors, vulnerabilities, and optimization opportunities efficiently. Traditional manual code review methods are time-consuming, inconsistent, and heavily dependent on developer expertise, making them difficult to scale in modern software development environments. The proposed system addresses these challenges by providing automated semantic and structural analysis of source code. The platform supports multiple programming languages including Python, Java, C, C++, and JavaScript. AI-powered semantic analysis is performed using LLaMA-based Large Language Models through the Groq API, while structural analysis is carried out using static analysis tools such as Pylint. The system identifies syntax errors, logical bugs, security vulnerabilities, code smells, and inefficient coding patterns. It also generates corrected code suggestions, optimization recommendations, and quality ratings for submitted code snippets. A web-based interface developed using Streamlit allows users to upload or paste source code and receive real-time feedback instantly. Performance metrics such as detection accuracy, response time, and code quality scores are used to evaluate system effectiveness. Overall, the system provides an intelligent, scalable, and efficient solution for automated software code review and bug detection.

Existing System

Traditional software code review systems mainly rely on manual peer review and rule-based static analysis tools for detecting coding errors and maintaining software quality. Developers typically inspect source code manually to identify syntax errors, logical bugs, and coding standard violations. Although peer review improves software quality, it is highly dependent on human expertise, consistency, and availability. Manual review processes are often slow, error-prone, and difficult to scale for large software projects with rapid release cycles. Existing static analysis tools such as Pylint, ESLint, FindBugs, and Checkstyle provide automated structural code analysis by applying predefined rules to source code. These tools can detect syntax errors, unused variables, and style violations efficiently. However, traditional static analyzers lack contextual and semantic understanding of source code, limiting their ability to detect logical bugs, architectural flaws, and advanced security vulnerabilities. Earlier machine learning-based systems improved defect prediction accuracy but required large labeled datasets and high computational resources. Many existing AI-powered code analysis systems focus mainly on code completion or review of code differences rather than complete code quality analysis. Proprietary AI tools also face limitations such as API restrictions, high costs, and limited accessibility. These limitations create the need for a more intelligent, scalable, and user-friendly automated code review system.

Disadvantages of Existing System

- Manual code review is time-consuming and inconsistent
- High dependency on developer expertise and availability
- Traditional static analysis lacks semantic understanding
- Difficulty in detecting logical and contextual bugs
- High false positive rates in rule-based systems
- Limited support for advanced security vulnerability detection
- Existing AI tools may require high computational resources
- Proprietary systems often have API and usage restrictions
- Limited support for multiple programming languages
- Reduced scalability in large software development projects

Proposed System

The proposed Gen AI-based Predictive Code Review and Bug Detection system introduces an intelligent platform that automates software code analysis using Large Language Models and static analysis techniques. The system uses Groq API powered by the LLaMA 3.3 70B model to perform deep semantic analysis of source code submitted by users. The AI model understands programming syntax, semantics, logic, and contextual relationships to identify bugs, vulnerabilities, and inefficient coding patterns accurately. The platform supports multiple programming languages such as Python, Java, C, C++, and JavaScript, making it suitable for developers working across different technology stacks. In addition to AI-powered analysis, the system integrates static analysis tools such as Pylint to detect syntax errors, style violations, and structural code issues. The system provides detailed feedback including bug descriptions, corrected code suggestions, security analysis, optimization recommendations, and overall quality ratings. A user-friendly web application built using Streamlit enables developers to upload or paste source code and receive instant analysis results. The proposed system significantly reduces the time and effort

required for manual code reviews while improving code quality and software reliability. It also supports educational and professional software development environments by generating human-readable explanations for detected issues. Overall, the proposed system demonstrates the effectiveness of combining Generative AI with static analysis for intelligent software engineering automation.

Advantages of Proposed System

- Automates software code review and bug detection
- Provides semantic and contextual understanding of code
- Detects logical bugs and security vulnerabilities effectively
- Supports multiple programming languages
- Reduces manual review effort and debugging time
- Generates corrected code suggestions automatically
- Improves software quality and reliability
- Provides real-time code analysis and feedback
- User-friendly web-based interface for developers
- Scalable and suitable for educational and professional use

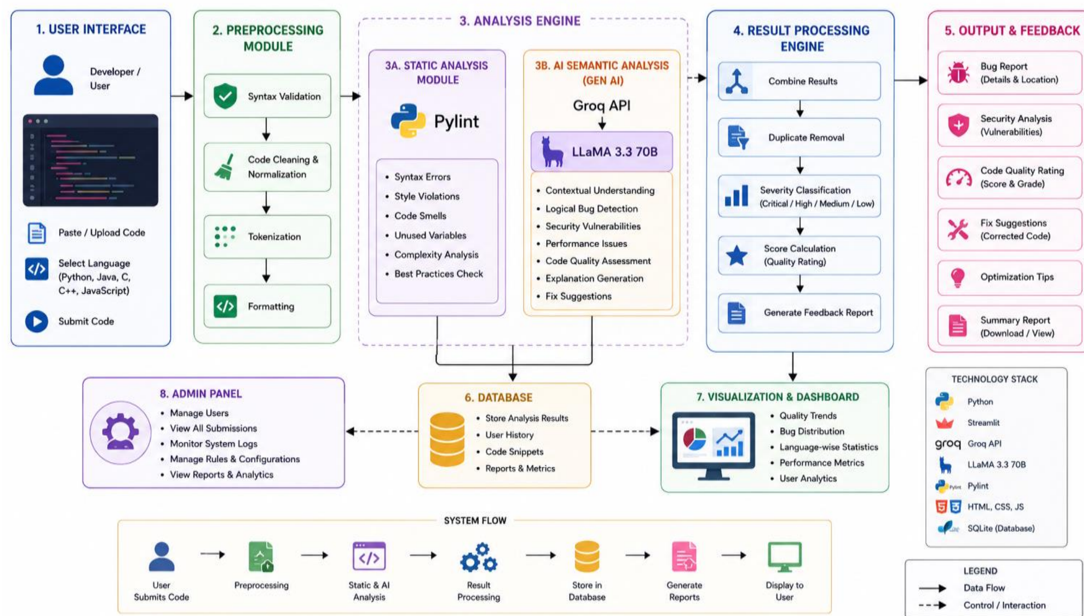
IV. Methodology

The methodology of the proposed system begins with collecting source code input from users through a web-based interface. Users can upload or paste source code and select the target programming language for analysis. The submitted code first undergoes preprocessing operations such as syntax validation, tokenization, and formatting normalization. After preprocessing, the code is passed to the static analysis module where tools like Pylint analyze structural issues such as syntax errors, unused variables, and coding standard violations. Simultaneously, the source code is sent to the AI semantic analysis module powered by the Groq API and LLaMA 3.3 70B model. The Large Language Model performs deep contextual analysis of the code to identify logical bugs, vulnerabilities, inefficient coding patterns, and optimization opportunities. The AI model also generates corrected code suggestions and human-readable explanations for detected issues. The outputs from static analysis and AI analysis are combined and processed into a comprehensive feedback report. The generated results include bug details, security analysis, quality ratings, optimization suggestions, and corrected code samples. The final analytical report is displayed to the user through the Streamlit web application interface. System performance is evaluated using metrics such as detection accuracy, response time, and bug prediction efficiency. This methodology ensures accurate, scalable, and intelligent automated code review using AI and static analysis technologies.

System Architecture

The system architecture of the Gen AI-based Predictive Code Review and Bug Detection system consists of multiple interconnected modules that work together to perform intelligent software code analysis. The process begins with the user interface module developed using Streamlit, where users upload or paste source code and select the programming language. The submitted code is forwarded to the preprocessing module, which performs syntax validation, tokenization, and normalization operations.

The processed code is then simultaneously sent to two separate analysis modules. The static analysis module uses Pylint and related tools to identify syntax errors, style violations, unused variables, and structural coding issues. The AI semantic analysis module communicates with the Groq API and LLaMA 3.3 70B model to perform contextual understanding of the code and detect logical bugs, vulnerabilities, and inefficient coding practices. The outputs from both analysis modules are combined in the result processing engine, which generates comprehensive feedback reports. The feedback includes bug descriptions, corrected code suggestions, quality ratings, optimization recommendations, and security analysis. The generated results are stored in a database for future reference and analytics. A visualization module presents reports and quality metrics through dashboards and formatted outputs. Real-time processing ensures fast and efficient code review performance. Overall, the architecture provides a scalable, intelligent, and automated solution for predictive code review and bug detection using Generative AI technologies.



V. Result and Output

AI Code Reviewer & Bug Detector

Powered by Gemini 1.5 Flash + Pylint | Supports Python, Java, C, C++, JavaScript

Select Language:

Java

Paste your code here:

```
def divide_numbers(a, b):
    result = a / b
    return result

def calculate_average(numbers):
    total = 0
    for n in numbers:
        total = total + n
    average = total / len(numbers)
    return average

numbers = [10, 20, 30, 40, 50]
print(calculate_average(numbers))
print(divide_numbers(10, 0))
```

Analyze Code

Static Analysis (Pylint)

`Static analysis (Pylint) is only available for Python.`

AI Review

Analysis of the Provided Java Code

The provided code is not Java, but rather a mix of Python and incorrect function definitions. Here's a breakdown of the issues:

1. Bug Detection

- The `divide_numbers` function does not handle division by zero, which will result in a `ZeroDivisionError` in Python or an `ArithmeticException` in Java.
- The `calculate_average` function does not handle the case where the input list is empty, which will result in a `ZeroDivisionError` in Python or an `ArithmeticException` in Java.
- The `divide_numbers` function is called with a divisor of 0, which will cause a crash.
- The code is attempting to use Python syntax in a Java context, which will not compile.

Deploy

2. Code Quality Issues

- The code is using Python syntax, but the question is asking for Java code.
- The `calculate_average` function can be simplified using the built-in `sum` function in Python or a more efficient algorithm in Java.
- The variable names are not descriptive, making the code harder to understand.
- There are no checks for invalid input, such as null or empty lists.

3. Security Issues

- The code does not handle potential overflow or underflow errors when performing arithmetic operations.
- The code does not validate user input, which could lead to security vulnerabilities if this code were part of a larger application.

4. Corrected Code

Here is the corrected Java code:

```java
public class Calculator {
    /**
     * Divide two numbers.
     *
     * @param a The dividend.
     * @param b The divisor.
     * @return The result of the division.
     * @throws ArithmeticException If the divisor is zero.
     */
    public static double divideNumbers(double a, double b) {
        if (b == 0) {
            throw new ArithmeticException("Cannot divide by zero");
        }
        return a / b;
    }

    /**
```

```
    /**
     * Calculate the average of a list of numbers.
     *
     * @param numbers The list of numbers.
     * @return The average of the numbers.
     * @throws IllegalArgumentException If the list is empty.
     */
    public static double calculateAverage(double[] numbers) {
        if (numbers.length == 0) {
            throw new IllegalArgumentException("Cannot calculate average of empty list");
        }
        double sum = 0;
        for (double num : numbers) {
            sum += num;
        }
        return sum / numbers.length;
    }

    public static void main(String[] args) {
        double[] numbers = {10, 20, 30, 40, 50};
        System.out.println(calculateAverage(numbers));
        try {
            System.out.println(divideNumbers(10, 0));
        } catch (ArithmeticException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

5. Overall Rating

I would rate this code as **Poor** due to the numerous bugs, code quality issues, and security vulnerabilities. However, with the corrections and improvements made in the corrected code, the overall rating would be **Good**.

Note: The original code was not Java, but rather Python. The corrected code is written in Java as per the question's request.

VI. Conclusion

The GenAI for Predictive Code Review and Bug Detection project successfully demonstrates the effectiveness of integrating Generative Artificial Intelligence with traditional static analysis techniques to create an intelligent and automated software code review system. The proposed system addresses major challenges in modern software development by providing fast, accurate, and context-aware analysis of source code across multiple programming languages. By combining semantic understanding from Large Language Models with rule-based static analysis, the system provides comprehensive feedback that improves software quality and developer productivity.

The integration of Pylint with the LLaMA 3.3 70B model powered through the Groq API enabled the system to detect both structural and logical code issues effectively. Static analysis tools efficiently identified syntax errors, coding standard violations, and structural problems, while the Large Language Model successfully analyzed contextual logic, security vulnerabilities, inefficient coding patterns, and semantic issues that traditional tools often fail to detect. The system demonstrated strong performance with high bug detection accuracy and reliable fix suggestion capabilities across Python, Java, C, C++, and JavaScript programming languages.

The web-based interface developed using Streamlit provided a smooth and user-friendly experience for developers by generating analysis reports within a few seconds. The feedback generated by the system included bug descriptions, corrected code suggestions, optimization recommendations, vulnerability analysis, and human-readable explanations, making the platform highly useful for both beginners and

professional developers. User evaluations also indicated improved understanding of programming mistakes and enhanced learning support through AI-generated explanations.

The project proves that intelligent orchestration of AI technologies and static analysis tools can significantly reduce the manual effort involved in traditional code review processes while improving software reliability and development efficiency. Unlike many enterprise-level AI code review systems, the proposed solution remains accessible, scalable, and cost-effective without requiring expensive infrastructure or highly specialized hardware.

References

[1] Kumar, R. D., Prudhviraj, G., Vijay, K., Kumar, P. S., & Plugmann, P. (2024). Exploring COVID-19 through intensive investigation with supervised machine learning algorithm. In Handbook of Artificial Intelligence and Wearables (pp. 145-158). CRC Press.

[2] Swathi, B., Vijay, K., Sushanth Babu, M., & Dinesh Kumar, R. (2024, November). Machine Learning Techniques in Cloud Based Intrusion Detection. In The International Conference on Artificial Intelligence and Smart Environment (pp. 557-564). Cham: Springer Nature Switzerland.

[3] Sv satyakrishna, shirisha rangu ,bhargavi nalacheruve.(2024) Prospective investigation on colorectal cancer with SMOTE on machine learning Algorithm

[4] Dr.G.Vishnu Murthy, BhargaviNalacheruve 1Professor, Department of computer Science & engineering, Anurag University, TS, India. 2Student, Department of computer Science & engineering, Anurag University, TS, India.

[5] V. N. S. Manaswini, K. K, C. Nigam, S. S. Ali, R. Niranjana, and Suman, "Real-Time Object Detection in Drone Surveillance Using YOLOv5," in Proc. 2025 3rd Int. Conf. IoT, Communication and Automation Technology (ICICAT), Gorakhpur, India, 2025, pp. 1–6, doi: 10.1109/ICICAT68430.2025.11414670.

[6] B. Soundarya, V. N. S. Manaswini, M. Ayyakrishnan, R. D. Kumar, "Contextual Analysis of Big Data Analytics in Intelligent Transportation Frameworks," in Intersection of Artificial Intelligence, Data Science, and Cutting-Edge Technologies: From Concepts to Applications in Smart Environment, Lecture Notes in Networks and Systems, vol. 1353, Cham: Springer, 2025, doi: 10.1007/978-3-031-88304-0_79.

[7] R. D. Kumar, V. N. S.Manaswini, "Applications of blockchain in smart cities: detecting fake documents from land records using blockchain technology," in Blockchain for Smart Cities, Elsevier, 2021, pp. 105–117, doi: 10.1016/B978-0-12-824446-3.00017-X.

[8] Tejavath Veeramma, Badarla Anil, Guguloth Ravinder, "An advanced movie recommender using collaborative filtering and sentiment analysis," International Research Journal of Modernization in Engineering Technology and Science, vol. 7, no. 7, July 2025, doi: 10.56726/IRJMETS81618.

[9] Ravi Kumar Banoth, Ramana Murthy B V, "Automatic crop recommendation system using LightGBM and decision tree machine learning models," Journal of Machine and Computing, vol. 5, no. 1, pp. 343, Jan. 2025, doi: 10.53759/7669/jmc202505026.

[10] Ravi Kumar Banoth, Dr. B.V. Ramana Murthy, “Smart agriculture through IoT and machine learning for analyzing carbon footprints,” in Proc. Int. Conf. Computer Science and Communication Engineering (ICCSCE), Apr. 2025.
[11] Ravi Kumar Banoth, B. V. Ramana Murthy, “Soil image classification using transfer learning approach: MobileNetV2 with CNN,” SN Computer Science, vol. 5, art. no. 199, 2024, doi: 10.1007/s42979-023-02500-x.