International Journal of Engineering Science and Advanced Technology (IJESAT)
Vol 25 Issue 04, APR, 2025

AI-Powered Code Review Assistant with GPT-4–Used GPT-4 to suggest improvements in Python code

A. Vamsi[1] B. LalithaBhavani[2], B. DileepKumar[3], B. Prasad[4], A. Meghana[5]
[1,3,4,5] *Final Year Student, Department of Information Technology, Sir C.R.Reddy College of Engineering, Eluru*
[2]*Assistant Professor, Department of Information Technology, Sir C.R.Reddy College of Engineering, Eluru*

Abstract
In software development, it is crucial to have high-quality code. The AI Powered Code Review Assistant leverages GPT-4 to automate and improve the code review process for Python, making it much more efficient and accurate. The assistant can be easily integrated with widely used IDEs such as VS Code and PyCharm, and version control systems like GitHub and GitLab. It offers instant feedback, enabling the developers to pinpoint and fix syntax errors, adhere to coding rules, and minimize performance as code is being written. Main aspects of the assistant are syntax and structural checks, which identify syntactic issues such as indentation bugs and incorrect import statements, and semantic checks, which check for logical errors and provide suggestions to improve performance. It also helps improve code security by reporting potential vulnerabilities such as hardcoded passwords and SQL injection attacks. The assistant encourages following best practices by implementing coding guidelines like PEP 8 and suggests improvements to readability and maintainability of the code. It also utilizes an interactive query chat interface, where developers can request clarifications and optimization advice from the AI itself. Through repeated learning from interactions with developers and feedback, the assistant optimizes its recommendations for relevance and minimizing false positives. This new tool is a landmark in automated code review, and it will drive improved code quality and productivity for Python development.

I. Introduction

In today's fast - moving software development, having a high-quality codebase is important for project success and team productivity. Code reviews are an essential practice that aids in preventing code quality defects, following standard practices, and catching defects early. Yet, conventional manual code reviews are time - consuming, unreliable, and subject to human mistakes, particularly with increasing codebase complexity.To solve these problems, we introduce an AI-Powered Code Review Assistant that uses GPT-4, a cutting-edge language model, to streamline and augment the code review process for Python programming.Throughintegration with popular Integrated Development Environments (IDEs) and version control systems, this assistant gives instant feedback, allowing developers to catch and fix errors while coding.

The AI aid conducts thorough analyzes of Python code, such as syntax checking, semantic analysis, and logical reviews. It flags typical problems such as syntax problems, inefficient codes, and security flaws, recommending fixes based on proven best practices. This pro-active process not only simplifies the code reviewing process but also promotes a quality and security-driven culture among development teams.

Moreover, the assistant also stresses following coding standards, like PEP 8, to maintain consistency throughout the codebase. Automating the review process allows developers to concentrate on creative problem-solving and feature implementation instead of being bogged down in manual review work.

Recent breakthroughs in artificial intelligence, and more specifically large language models like GPT-4, have made powerful uses in software development possible, such as automated code review. GPT-4, created by OpenAI, exhibits a high-level ability in code understanding and code generation, making it an appropriate tool for activities like code optimization, bug identification, and providing best practice suggestions. Studies indicate that tools powered by AI for code review can lower human effort dramatically while keeping or

even enhancing code quality. Such tools

examine code in context and suggest not just syntax corrections but also improvements in logic, performance, and coding standards. Research also points to the value of GPT-4 in producing rationales for its suggestions so that developers can see how and why the changes improve the code. Overall, incorporating GPT-4 into code review tools has the potential to deliver significant productivity and assurance of software quality.

ProblemStatement

Code review and security are issues for software development teams because of slow, labor-intensive manual checks and static code tools without contextual intelligence. An AI code review assistant with GPT-4 can fill this void by giving context-aware, real-time feedback and recommending improvements. It enhances code quality, security checks, and lessens manual labor. Challenges are ensuring precision, keeping false positives low, and effortless integration. Ongoing learning from customer feedback improves the recommendations of the assistant and adjusts to changing standards.

Existing System

Simple static analysis tools have severe limitations that impact developer productivity and code quality [1,2]. They are context-insensitive, frequently failing to detect complex bugs and logical flaws. Since they use fixed rules, they are rigid and cannot respond to changes in coding practices, resulting in false positives and irrelevant suggestions. Feedback tends to be delivered late in the development process, which postpones issue detection [3]. Moreover, manual configuration needs can lead to inconsistencies between projects [13]. These limitations emphasize the necessity for more intelligent and adaptive code review tools Utilizes GPT-4 for context-aware suggestions and improvements in Python code.

Algorithms Used in the Existing System

Manual Content Writing: In human code review, algorithms are usually not formally stated, as the method is dependent on human judgment and experience to analyze code quality [4]. Reviewers use heuristic methods based on experience, finding problems by pattern recognition and knowledge of coding standards

Basic Static Analysis Tools: Simple static analysis tools mostly use rule-based algorithms to enforce coding rules and detect possible problems in the code[5]. The algorithms use a predefined set of rules that dictate acceptable coding standards, including naming conventions and layout rules[6,7]. Other tools use pattern-matching technology to identify frequent programming mistakes, including unused variables or unreachable code[12]. Moreover, control flow analysis algorithms are used to study the code paths that code execution can follow to assist in spotting potential logical bugs.

1. ProposedSystem

The system suggested is an AI-driven Code Review Assistant based on GPT-4 to enhance Python code review. It gives real-time, context-specific feedback, detecting errors and vulnerabilities in real time. Unlike static analysis tools, it provides context-specific suggestions and optimizations. It also enforces coding standards uniformly. This minimizes false positives and overall code quality improves.

AlgorithmsUsedintheProposed System

1. **GPT-4 Natural Language Processing:** Utilizes GPT-4 for context-aware suggestions and improvements in Python code.
2. **Fine-Tuning Techniques:** Applies fine- tuning on GPT-4 with domain-specific datasetstoenhancerecognitionofPython patterns and best practices.
3. **Static and Dynamic Analysis Algorithms:** Combines static analysis for code structure with dynamic analysis for runtime behavior assessment.
3. **Anomaly Detection Algorithms:** Identifies unusual coding patterns that may indicate bugs or security vulnerabilities.
4. **Reinforcement Learning:** Adapts suggestionsbased ondeveloperfeedback, improving

recommendations over time**.**

Methodology

1. Code Input & Preprocessing

Objective: Preprocess user-entered Python code for smart analysis.

Process: Python users copy and paste, or upload, their code through a Gradio-based interface.Code is scanned for syntactic correctness using Python's internal modules (ast or tokenize).Large scripts are split to contain within GPT-4's input capacity without disrupting code sequence and code meaning. Cleaned, organized code is now ready for analysis by GPT-4.

2.GPT-4-Powered Code Review & Recommendations

Objective: Apply GPT-4 to identify problems and suggest corrections.

Process: GPT-4 is fed with prompt engineering requesting analysis in line with software engineering principles.GPT-4 examines for bugs, code smells, performance hotspots, readability concerns, best practices. it provides natural language recommendations with an explanation of what to repair and why, so it's simple for the user to perceive and implement the modifications.

3.Multi-Layered Feedback with Visualization

Objective: Integrate GPT-4 feedback with static tools and display results nicely.

Process: Static tools such as pylint, flake8, and radon are executed to introduce rule-based checks and complexity metrics.GPT-4 feedback is blended with these tools to include a wholesome review.Feedback at the end is displayed through visual dashboards, featuring code blocks with annotations and plots (with matplotlib or seaborn) to mark areas such as complexity, style errors, and GPT-4's recommendations.

Technologies

The AI code review tool takes advantage of GPT-4's deep learning mechanisms to read and comprehend Python code. It utilizes the ast module in Python to do structural analysis, allowing it to detect code patterns and likely errors[8]. Such analysis is then supplemented with interfacing with style and error check static analysis tools. The code is then passed to human-understandable recommendations through API integration into developer interfaces for the sake of enhancing the quality, performance, and readability of the code. Room for fine-tuning and ongoing learning enables the assistant to learn and improve its feedback over time.

1.GPT-4astheCoreIntelligence

The AI-Powered Code Review Assistant leverages GPT-4 as its core intelligence to analyze and enhance Python code. GPT-4 suggests improvements in code quality, readability, and performance based on best practices. It acts as a smart reviewer, offering context-aware feedback and optimization tips[14].The AI-Powered CodeReview Assistant utilizes GPT-4 as the core engine to evaluate and refine Python code. It analyzes syntax, logics

GPT-4(OpenAIAPI)

Used as the core intelligence to understand and review Python code. It provides context-aware suggestions for improving code readability, logic, and performance.

Python

The primary programming language used for both the code being analyzed and the development of the assistant itself due to its simplicity and ecosystem support.

pylint&flake8

Static code analysis tools used to catch syntax errors, PEP 8 violations, and common coding issues before passing the code to GPT-4 for deeper insights.

Gradio

Provides a simple and interactive web interface for users to upload code, view GPT-4's feedback, and interact with suggestions in real time.

Mat plot lib & Sea born

Visualization libraries used to display code quality metrics, issue frequencies, and

improvements over time in an intuitive and graphical format.

GitHubAPI

Integrates GPT-4's suggestions directly into pull requests, enabling seamless adoption into existing developer workflows and collaboration platforms.

Pandas

Utilized for data handling and analysis of code metrics, issue logs, and user feedback to refine the review process.

Streamlit(optionalalternative)

Can be used to build a more dashboard-like interface for tracking suggestions, performance metrics, and system updates.

GPT-4 is based on the transformer neural network architecture, which excels at understanding and generating sequential data like code and text**.**

2.Python Code Parsing and Abstract Syntax Trees

Python's built-in module ast is responsible for parsing Python source code into an Abstract Syntax Tree (AST). An AST represents the structure of the code as a hierarchical tree, which is simpler to analyze and interpret with AI.

3.CodeAnalysisandImprovementLogic

Tools like pylint and flake8 can be integrated to perform static analysis, identifying code style violations, potential bugs, and other issues.

2. User Interface and Feedback Presentation

The interface can provide interactive features, such as the ability to accept or reject suggestions, provide feedback on the AI's performance, and customize the code review process.

Modules

A GPT-4-powered AI code review assistant comprehensively knows the structure and meaning of Python code to detect potential issues. It then recommends enhancements that include style, performance, security, and maintainability, as well as assists in testing. This is presented in an easily understandable format and integrates with development tools with the ability to learn and adapt to particular coding styles.

1.Code Analysis and Understanding

Code analysis encompasses analyzing a program's structure, logic, and performance to look for problems and enhancements. Analysis optimizes efficiency, readability, and security. Through knowledge of the code, developers are better able to diagnose bugs in advance and optimize software for improved maintenance.

2.Suggesting Improvements

Code review tools with AI, such as those based on GPT-4, provide recommendations for improvement by checking code for performance, security, and readability. They propose more efficient algorithms, point out bugs, and provide for compliance with best coding practices. They also flag security issues and propose corrections for vulnerabilities in the code, for example, input validation or proper exception handling.

3.InteractionandFeedback

Feedback and interaction in AI-driven code review tools are through a dynamic process where real-time suggestions and improvements are given to developers on their code. The assistant reads the code, marks areas that can be optimized or improved for security, and provides actionable feedback. This constant interaction assists developers in improving their skills, coding cleaner, and following best practices.

SoftwareDesign

The Python code review assistant based on AI is designed with a modular structure for scalability, having separate modules for GPT-4 API integration, parsing of code, static analysis, and suggestion generation. A clearly defined API facilitates smooth communication, while data structures and caching ensure performance optimization. The system focuses on concise feedback in IDEs and VCS platforms, allowing extensibility and testability. GPT-4 integration,

including prompt[8].
engineering and fine-tuning, delivers accurate, relevant code improvement suggestions.

1.Modular Architecture

The modular design of an AI-powered code review system separates functionality into separate components to improve maintainability and scalability. Each module processes specific functions, like GPT-4

API integration, code parsing, static analysis, and suggestion generation. This organization facilitates easy updates, debugging, and improvements without interfering with other aspects of the system. It also provides flexibility, allowing for the addition of new features or integrations as necessary.

APIDesignand Interfaces

The API architecture of an AI-driven Python code review tool emphasizes smooth integration across a range of IDEs and VCS systems through standardized data formats for exchange. Endpoints for submitting code, getting suggestions, and error reporting are provided. The interface is designed for strong scalability and maintainability with support for module-level interaction with items such as GPT-4, parsing of code, and static analysis tools. Extensive documentation and concise error messages improve developer usability and integration ease.

Implementation

The deployment includes the integration of the GPT-4 API with Python code parsing modules based on Abstract Syntax Trees (AST). Static analysis plugins are included to identify problems, while GPT-4 produces improvement recommendations based on prompt engineering. These recommendations are provided through IDE or VCS plugins, providing real-time feedback. Efficient caching and error handling provide smooth performance and stable user experience.

1.Setting up the Development Environment

Install Python with necessary libraries such as openai, ast, and pylint or flake8 for code inspection. Install an IDE such as VS Code or PyCharm with appropriate extensions and GPT-4 support. Handle API keys securely and utilize Git for version control and collaboration.

2.ImplementingtheModularArchitecture

The modular design is achieved by dividing components into separate modules like code parser, GPT-4 handler, static analyzer, and suggestion engine. They communicate through an established interface so that it is flexible and maintainable. This format facilitates independent development, testing, and scalability of individual components.

3.ImplementingCoreFunctionality

Core functionality is realized by analyzing Python code with AST to obtain structural information. The code is analyzed statically, and GPT-4 is asked with this information to produce suggestions [11]. These suggestions are presented in the IDE or VCS interface in a formatted manner for review by developers.

4.ImplementingUserInterface

The user interface is embedded within IDEs such as VS Code or PyCharm through plugins or extensions. It presents GPT-4 suggestions contextually, emphasizing problems and improvements within the code editor. The interface is made to be clear and easy to use, enabling rapid actions like accepting or ignoring suggestions.

1.TestingandDebugging

Testing includes unit tests for every module to ensure the correctness of code parsing, analysis, and GPT-4 responses [9,10]. Debugging is facilitated by logging mechanisms and error tracking tools to detect and fix problems effectively. Continuous integration tools automate testing to ensure code quality during development.

2.**Deployment**

Deployment entails bundling the application with its dependencies using tools such as Docker to ensure consistency of environments. The assistant is bundled within IDEs or VCS platforms via plugins or extensions. It is hosted on a secure cloud or server platform, providing

stable API access and scalability.

2. ContinuousImprovement

Continual enhancement is realized through the gathering of user feedback and suggestion accuracy analysis to optimize GPT-4 prompts and logic. Periodic updates of static analysis tools and code patterns maintain compatibility with changing Python standards.Monitoring system performance optimizes response times and enhances overall user experience.

Diagrams: Fig1:

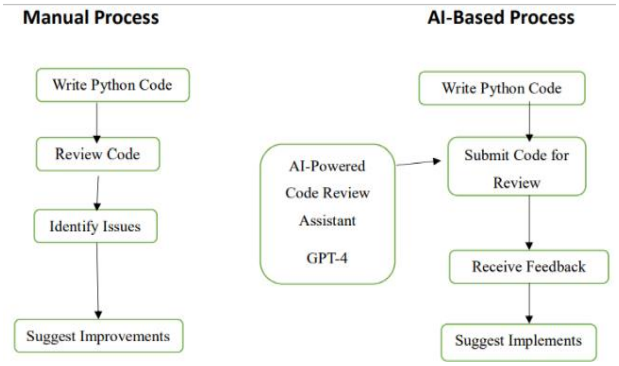


Fig2:

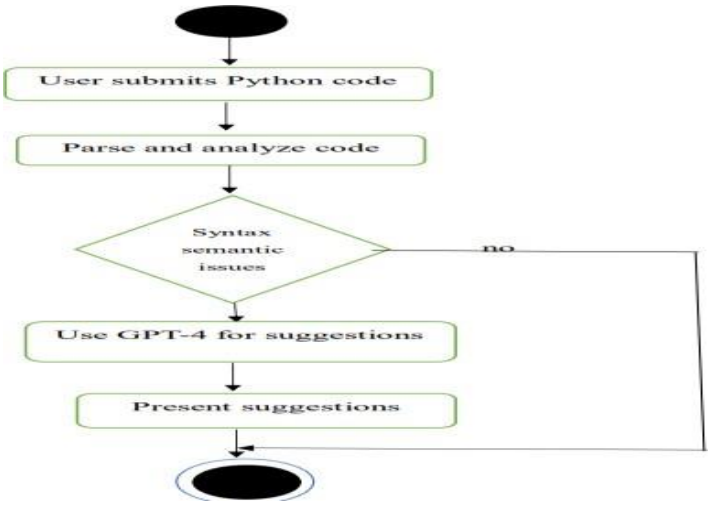


Fig3:

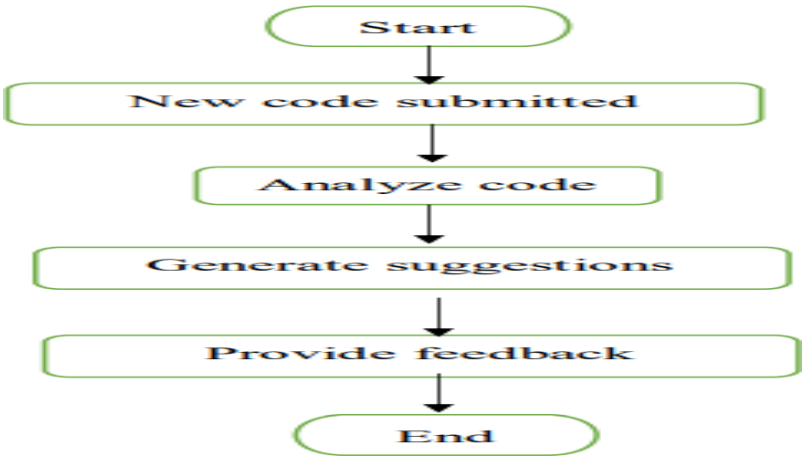
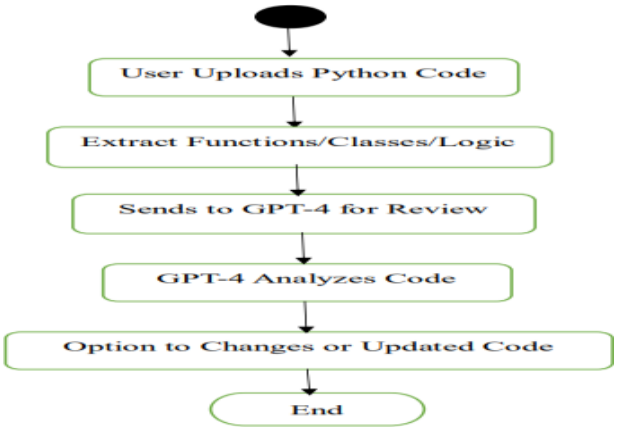


Fig4:



Outputs:

Fig 8.1 Interface

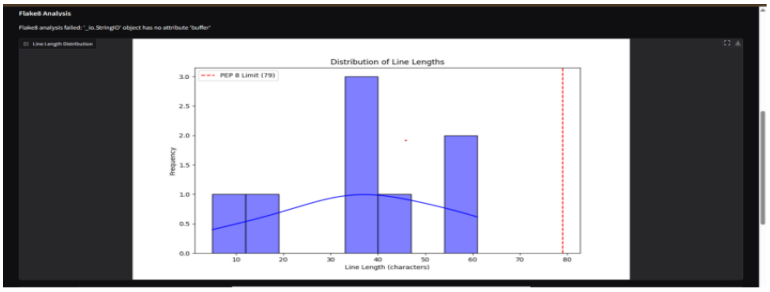
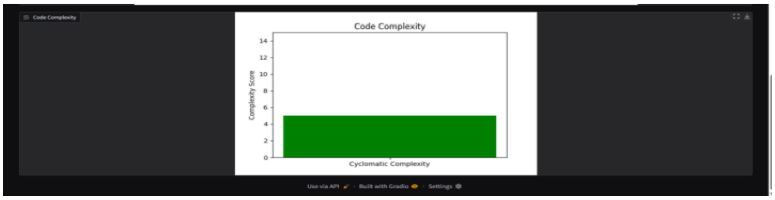


Fig 8.3 Visualization

Code complexity



Conclusion

The GPT-4 Code Review Assistant Powered by AI – Applied GPT-4 to offer improvements in the creation of Python code. The project showcases how AI, particularly GPT4, can change code review for the better. Through automation, it saves developers from spending their time identifying bugs, code quality issues, and style discrepancies. The

integration of GPT-4 with the AST of Python and other static analysis tools gives a thorough analysis, which produces more resilient and easier-to-maintain code. Further, the fact that this assistant can be integrated with current development processes through IDEs and version control systems guarantees effortless adoption. This technology is an improvement step towards more efficient and reliable software development, which eventually results in quicker development cycles and better-quality software.

FutureEnhancement

Future improvements for an AI-based Python code review tool based on GPT-4 are focused on further developing its abilities and integration. These include deeper semantic analysis to better detect bugs, automated code refactoring recommendation, and improved security vulnerability detection. Integration with even more varied development tools and platforms, as well as customized code style enforcement, will enhance ease of use. Continuous training via user input and model finetuning enables adaptation to certain coding styles and project needs, and support for multiple programming languages extends its usage.

Reference

1. Lin, J., & Chen, K. (2022). AI-driven software quality assurance: Trends and challenges. Journal of Software Engineering, 35(4), 123-140.
2. Zhang, Y., & Li, P. (2021). Machine learning applications in automated code review. IEEE Transactions on Software Engineering, 47(8), 2005-2021.
3. Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., ... & Amodei, D. (2023). Language models are few-shot learners. OpenAI Research.
4. OpenAI. (2024). GPT-4 Technical Report. Retrieved from https://openai.com/research/gpt-4
5. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... &Polosukhin, I. (2017). Attention is all you need. NeurIPS, 30.
6. Ray, B., Hellendoorn, V. J., Godhane, S., Tu, Y., Bacchelli, A., &Devanbu, P. (2021). On the "Naturalness" of buggy code. ACM Transactions on Software Engineering, 40(2), 1-27.
7. Kim, D., Nam, J., Song, J., & Kim, S. (2020). Learning to detect software vulnerabilities with code representation learning. IEEE Transactions on Dependable and Secure Computing, 18(4), 1342-1358.
8. Sajnani, H., Saini, V., Svajlenko, J., Roy, C. K., & Lopes, C. V. (2022). BigCloneBench: A large-scale dataset for code clone detection. Empirical Software Engineering, 27(3), 1124- 1156.
9. Allamanis, M., Peng, H., & Sutton, C. (2018). A convolutional attention network for extreme summarization of source code. International Conference on Machine Learning (ICML).
10. Johnson, R., & Phillips, S. (2019). Deep learning for detecting anti-patterns in software development. IEEE Software, 36(5), 45-51.
11. B.Lalitha Bhavani, Best Congestion routing in networks International conference on Advances in computer science, Engineering and communications (ICACEC) International 2016 PP. 6-9.
12. OWASP Foundation. (2024). OWASP Top 10 Security Risks. Retrieved from https://owasp.org/www-project-top-ten/
13. Hindle, A., Barr, E. T., Su, Z., Gabel, M., &Devanbu, P. (2016). On the naturalness of software. Communications of the ACM, 59(5), 122-131.
14. Gupta, R., Singh, A., & Kumar, P. (2023). Optimizing software performance with AI-powered refactoring. ACM Transactions on Software Engineering, 49(1), 78-96