AGENTCODE INSPECTOR: AN AGENTIC AI-BASED AUTONOMOUS CODE REVIEW SYSTEM

Mrs. Hema Prabha G, Madhusree M, Neethish S, Sanjanaa S
Department of Computer Science and Engineering, Bachelor of Engineering,
Sri Shakthi Institute of Engineering and Technology, Coimbatore – 641062

ABSTRACT

This project focuses on developing an intelligent and autonomous system capable of accurately reviewing source code and identifying bugs, code smells, security vulnerabilities, and performance inefficiencies. By leveraging advanced technologies such as **Agentic AI, Large Language Models (LLMs), and rule-based analysis frameworks**, the system ensures high accuracy and consistent performance in detecting code-level issues. The **AgentCode Inspector** allows developers to submit code through a web interface, and the AI agent autonomously performs multi-step code analysis and delivers human-like review comments and optimized code fixes. Key features include **syntax checking**, **complexity analysis**, **security scanning**, and **intelligent suggestion generation** for seamless software quality improvement. The system is supported by a robust agent framework that enables continuous reasoning and iterative evaluation, ensuring reliable and efficient performance in automating the software code review process.

Keywords—Agentic AI, Code Review Automation, LangChain, CrewAI, AutoGen, Static Code Analysis, Security Vulnerability Detection, Code Smell Detection, Multi-Step Agent, Flask, FastAPI, Software Quality, Bug Detection

INTRODUCTION

- The **AgentCode Inspector** uses Agentic AI to autonomously analyze source code and generate detailed review comments, helping bridge the gap between manual code review limitations and modern software complexity.
- It applies **multi-agent frameworks and large language models** to evaluate code based on logic, structure, security, and performance — going far beyond traditional rule-based linters.
- The system uses **LangChain / CrewAI / AutoGen agent pipelines** to plan and execute multi-step code analysis, achieving consistent and high-quality reviews without human intervention.
- By automating the entire review process, the system makes software development faster, more reliable, and more secure through intelligent feedback and code fix suggestions.

Objective:

- The objective of the AgentCode Inspector is to develop an intelligent agentic system that can autonomously review source code and provide actionable feedback with optimized fix suggestions in real time.
- Automate the code review process to reduce manual effort and minimize human errors that occur during developer-driven evaluations.
- Detect bugs, code smells, security vulnerabilities, and performance issues using structured multi-step agent reasoning pipelines.
- Use Agentic AI to perform autonomous, iterative code analysis where agents plan, execute, and refine review steps independently.
- Provide intelligent suggestions and optimized code fixes that help developers resolve issues quickly and efficiently.

- Design the system to be scalable and user-friendly, supporting code submission through a web interface for both individual files and project modules.
- Integrate rule-based analysis alongside AI-driven reasoning to improve the accuracy and coverage of detected issues.
- Improve overall software quality and development efficiency by delivering consistent, experience-independent review feedback.

LITERATURE SURVEY

1. AI and LLM-Based Code Review Approaches:

Early research in automated code review focused on static analysis tools and rule-based linters that flag predefined patterns. Recent studies have shifted toward using Large Language Models (LLMs) to perform context-aware code review. Sharma & Rattan (2025) conducted a review on code quality generated by AI tools, evaluating the effectiveness of various AI-assisted review techniques and their impact on software reliability and maintainability.

2. Agentic AI in Software Engineering Workflows:

Agentic AI systems operate through autonomous agents that plan and execute multi-step tasks without human intervention. Viswanathan (2024) explored AI agentic scriptless automation in software testing, demonstrating that agent-based systems can autonomously orchestrate complex testing workflows. This foundation supports the application of similar agentic approaches to code review automation.

3. AI Tools for Contextual Code Feedback:

Providing contextual and actionable feedback during code review is a key challenge. Jangam & Karri (2025) investigated AI tools for automating code reviews and providing contextual feedback, concluding that AI-driven systems significantly improve the review process efficiency and feedback quality compared to traditional manual approaches.

4. Agent Frameworks for Multi-Step Reasoning:

Frameworks such as LangGraph, AutoGen, and CrewAI enable AI agents to reason across multiple steps, delegate tasks, and coordinate to achieve complex goals. The official LangGraph documentation highlights how graph-based agent workflows allow for precise control over multi-step reasoning, while the AutoGen framework by Microsoft enables multi-agent collaboration for complex task automation, both of which are foundational to the proposed system.

METHODOLOGY

Manual Code Review Process:

Traditionally, reviewing source code requires experienced developers or senior engineers who manually inspect logic, style, and security issues in submitted code. This process is time-consuming, subjective, and inconsistent across reviewers, making it difficult to maintain quality at scale.

Limitations of Traditional Techniques:

Earlier automated systems for code analysis relied on static analysis tools such as linters and rule checkers that flag only predefined patterns. While useful for syntax errors and style violations, these approaches fail to capture deep logical flaws, context-dependent security vulnerabilities, or optimization opportunities, resulting in incomplete and inconsistent feedback.

Challenges with Rule-Based Systems:

Some existing systems rely entirely on fixed rule sets to evaluate code, but these are often rigid, difficult to update, and unable to understand the semantic intent of the code. They also lack the ability to suggest contextually appropriate fixes, reducing their usefulness for developers working on complex software systems.

Advancement with Agentic AI Models:

Modern approaches employ **multi-agent AI frameworks** and **Large Language Models (LLMs)** to autonomously plan and execute multi-step code evaluations. Agentic AI systems overcome the limitations of manual and rule-based methods by providing intelligent, context-aware feedback across syntax, security, complexity, and performance dimensions. These systems ensure **high consistency, autonomous operation, and scalability** for effective and reliable software code review automation.

EXISTING SYSTEM

1. Traditional Manual Code Review Systems

How It Works:

Traditional code review systems primarily rely on human developers or senior engineers who manually inspect source code during pull request reviews or scheduled audit sessions. Some teams also use basic linting tools to enforce style guidelines alongside manual reviews. These systems depend on the reviewer's personal expertise and experience to identify issues.

Drawbacks:

- **Inconsistency:** Feedback quality varies significantly depending on the reviewer's knowledge and experience, leading to uneven code quality across projects.
- **Time-Consuming:** Manual reviews require significant developer time, especially for large codebases, slowing down the development pipeline.
- **Limited Coverage:** Human reviewers may overlook subtle bugs, security vulnerabilities, or performance bottlenecks due to fatigue or expertise gaps.

2. Rule-Based Static Analysis Tools

How It Works:

Static analysis tools such as ESLint, SonarQube, and Pylint scan code against predefined rule sets to flag violations. These tools check for syntax errors, style issues, and known anti-patterns using deterministic rules and pattern matching without executing the code.

Drawbacks:

- **Rule Rigidity:** Static tools can only detect issues that match their predefined rule sets and cannot reason about context or code intent.
- **No Fix Suggestions:** Most tools only identify problems without providing intelligent, context-aware suggestions for resolution.
- **False Positives:** Rule-based systems frequently generate false positives that require manual filtering, reducing developer trust and efficiency.

- **Inability to Handle Complex Logic:** These tools are unable to detect deep logical errors or security flaws that require semantic understanding of the codebase.

PROPOSED SYSTEM

Agentic AI-Based Analysis:

The proposed system uses Agentic AI frameworks (**LangChain / CrewAI / AutoGen**) to autonomously perform multi-step code review. Agents are orchestrated to independently plan, execute, and refine evaluation tasks including syntax checking, complexity analysis, security scanning, and optimization recommendation.

Autonomous Multi-Step Code Evaluation:

The system reviews code through a structured pipeline where each agent handles a specific review dimension. It automatically extracts issues from the submitted code and classifies them by severity, type, and recommended fix — helping developers resolve problems efficiently in real time.

Intelligent Fix Suggestion Engine:

The system not only identifies issues but also generates context-aware, optimized code fixes using LLM reasoning. This provides developers with actionable improvements rather than just problem descriptions.

Confidence and Severity Scoring:

Each detected issue includes a severity rating and confidence score, indicating how critical the problem is and how reliably it has been identified, ensuring developers can prioritize their fixes effectively.

User-Friendly Web Interface:

A simple and intuitive web interface built with Flask / FastAPI allows developers to paste or upload their code and instantly receive a comprehensive AI-generated code review report with suggestions and fixes.

SYSTEM REQUIREMENTS

HARDWARE SPECIFICATIONS

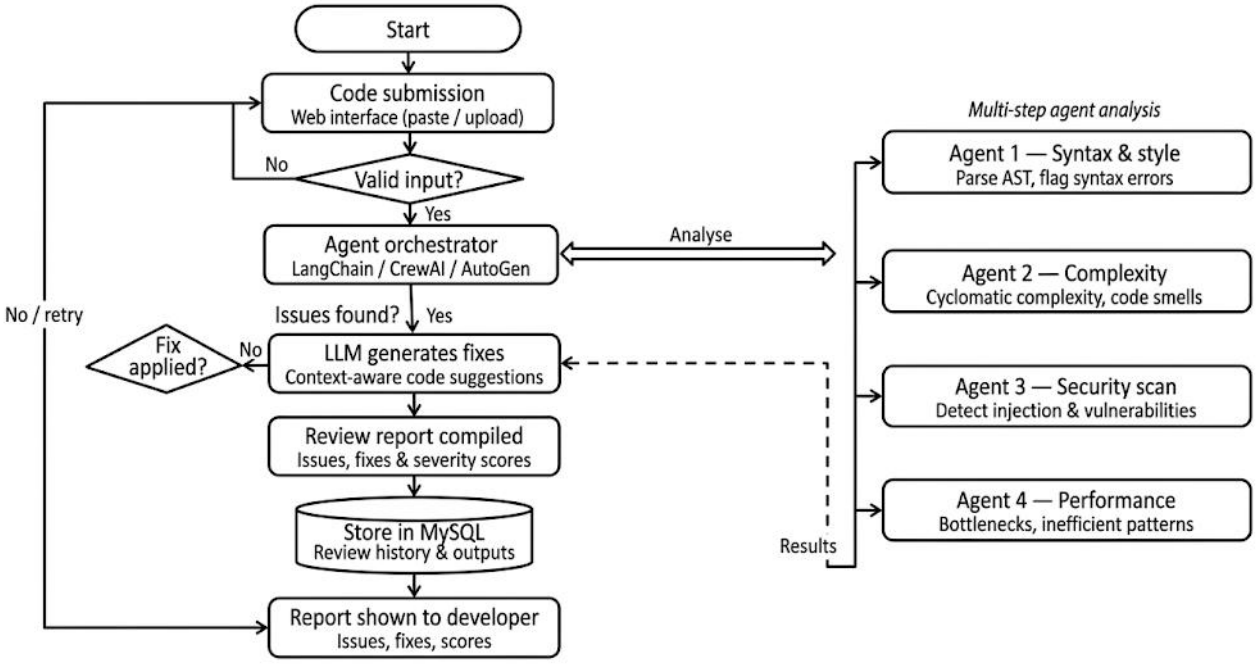
- RAM: 8–16 GB or higher for smooth model inference and agent pipeline execution.
- Processor: Intel i5/i7 or AMD Ryzen series for fast multi-step agent computation.
- Storage: 256 GB or more for storing project files, model configurations, and logs.
- Additional: Stable internet connection for accessing LLM APIs and a standard keyboard and display for interface interaction.

SOFTWARE SPECIFICATIONS

- Operating System: Windows 11.
- Frontend: HTML, CSS, JavaScript, Bootstrap for a responsive and user-friendly interface.
- Backend Framework: Python (Flask / FastAPI) for API routing and request handling.

- AI Technologies: Agent Frameworks — LangChain / CrewAI / AutoGen for multi-step agentic reasoning and code evaluation pipelines.
- Database: MySQL for storing submitted code, review history, and agent outputs.

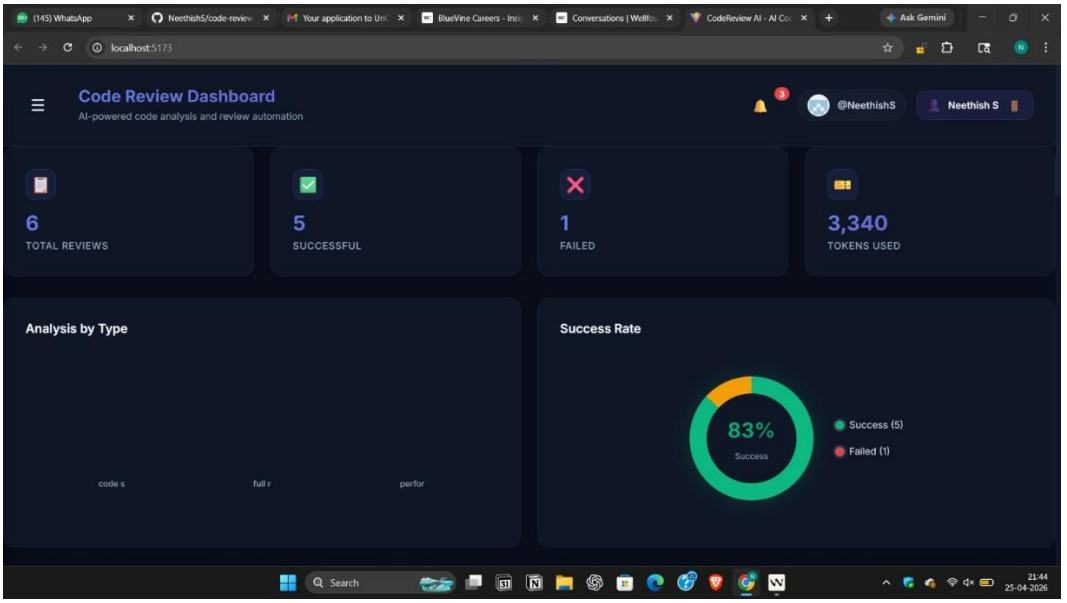
PROCESSING



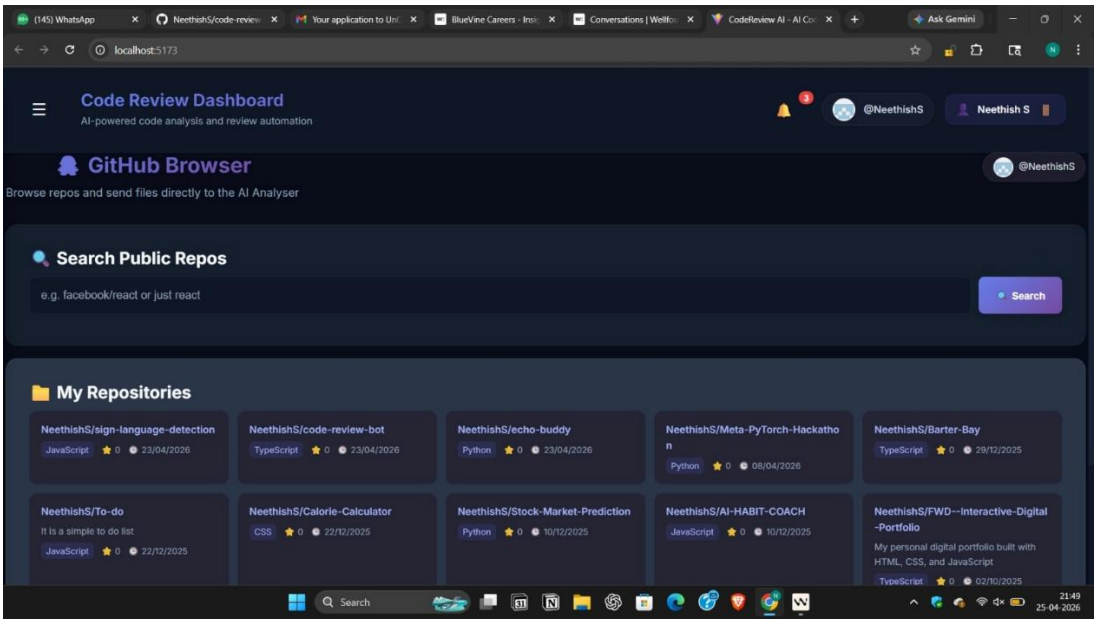
MODE OF DESCRIPTION

- Code Submission Interface
- Agent Pipeline Execution
- Syntax and Complexity Analysis
- Security Vulnerability Scanning
- Optimization and Fix Suggestion
- Review Report Generation

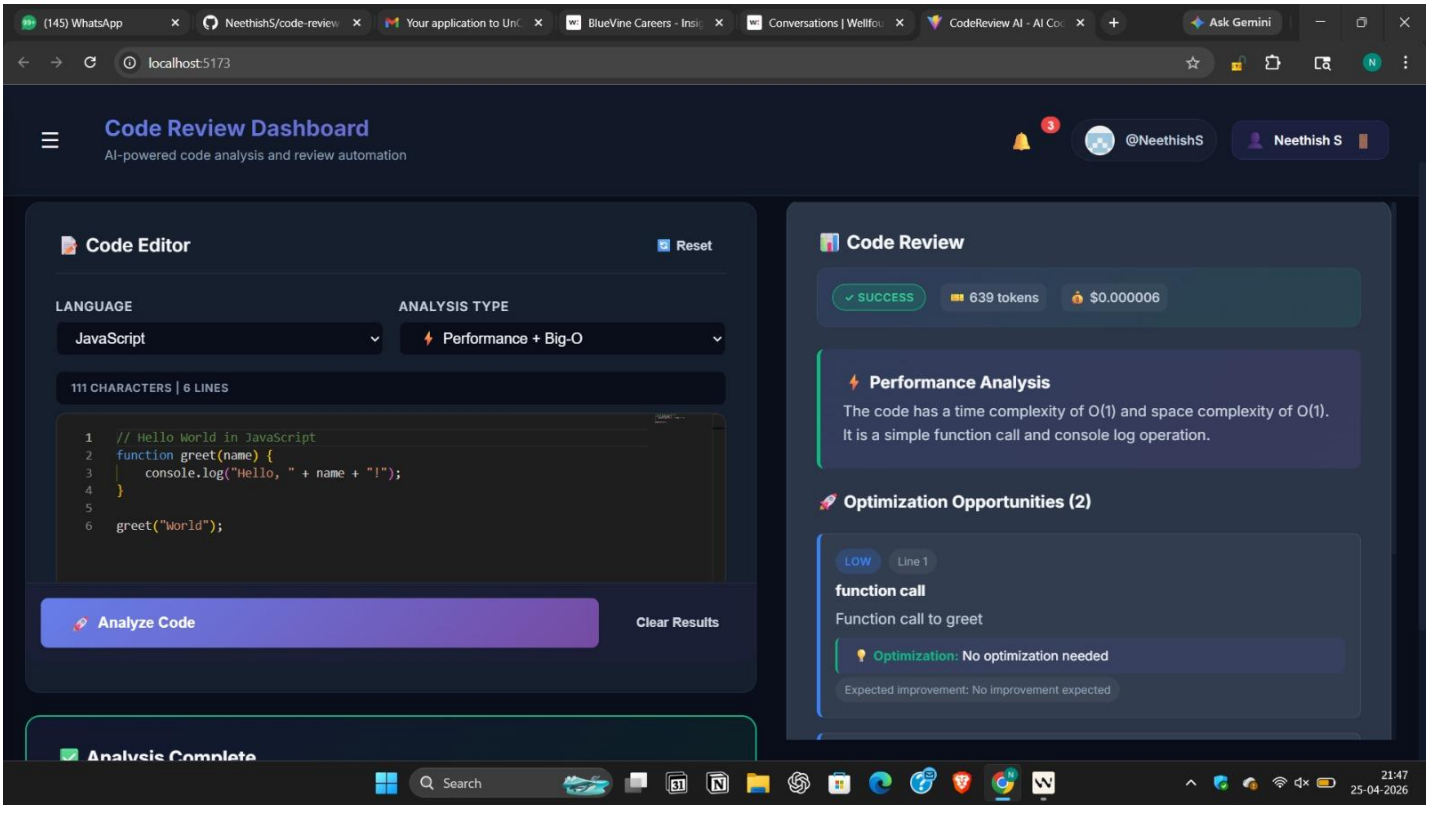
CODE REVIEW INTERFACE:



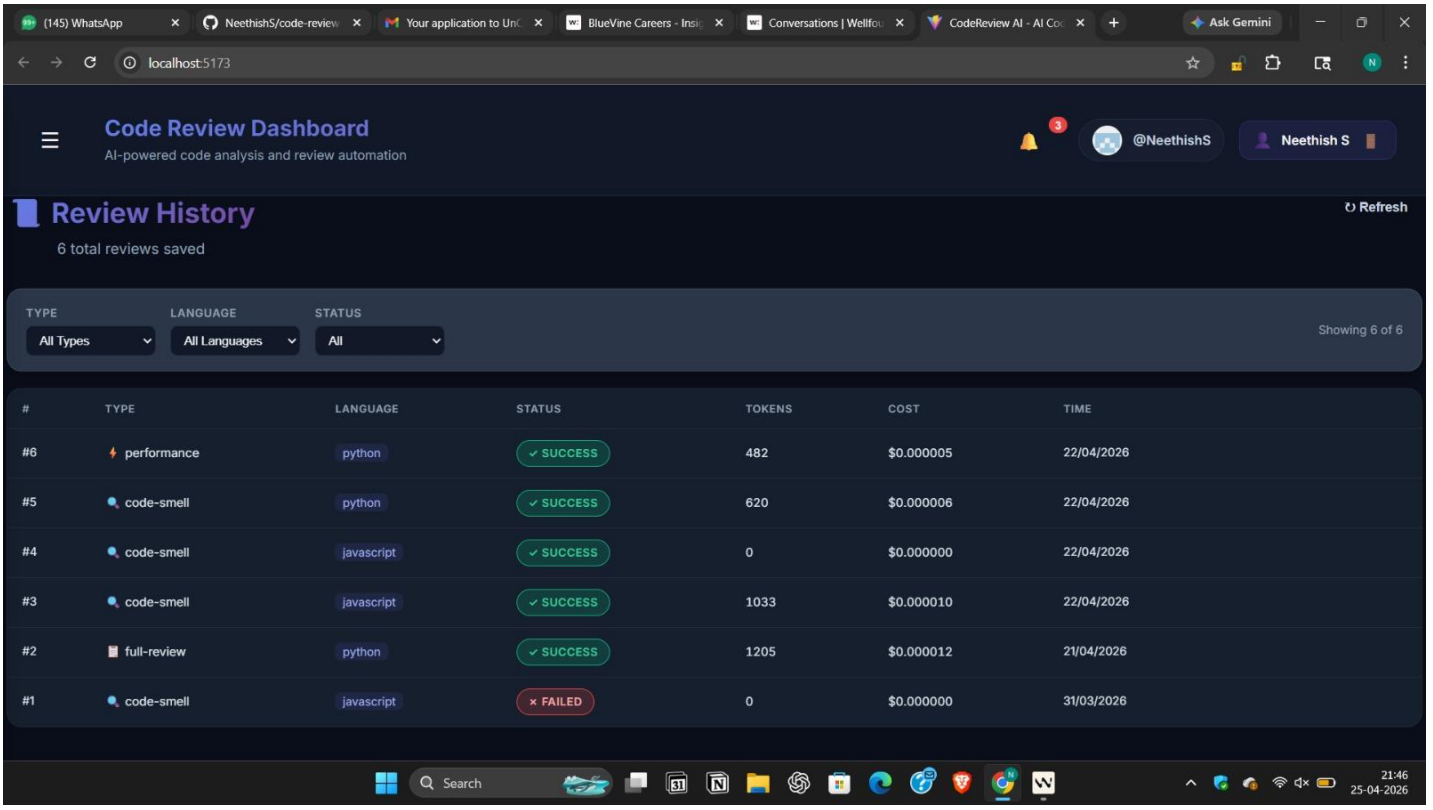
GITHUB BROWSER:



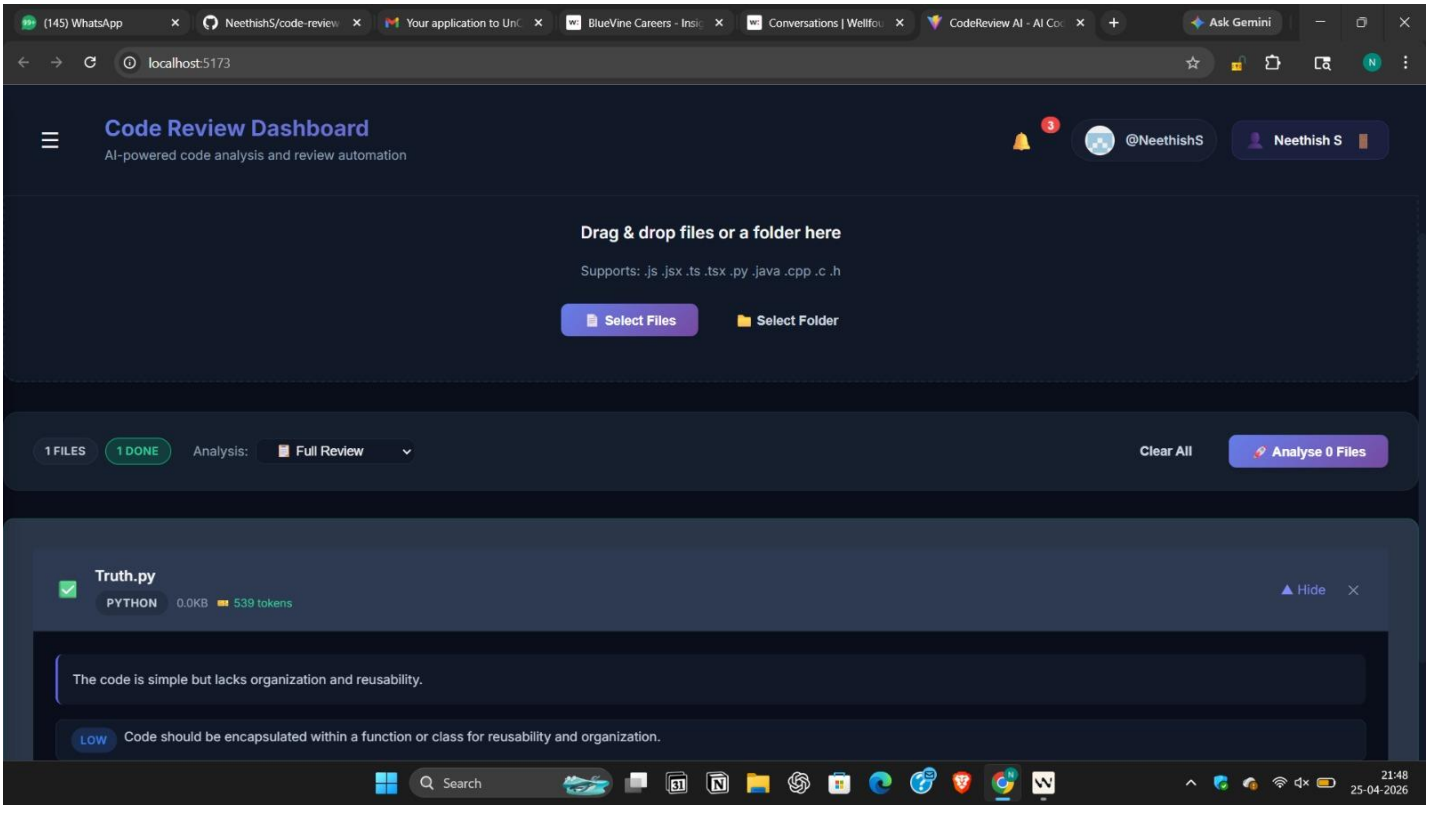
SYNTAX AND COMPLEXITY ANALYSIS:



REVIEW HISTORY:



REVIEW REPORT GENERATION:



CONCLUSION

The **AgentCode Inspector** effectively provides a reliable and autonomous solution for reviewing source code and delivering intelligent, actionable feedback. By leveraging **Agentic AI frameworks such as LangChain, CrewAI, and AutoGen** along with advanced LLM-based reasoning, the system achieves high consistency and coverage in identifying bugs, security vulnerabilities, code smells, and performance issues across diverse codebases. Overall, this project contributes to the field of AI-assisted software engineering by offering an accessible, efficient, and scalable platform that reduces manual review effort and improves code quality. It also lays the foundation for future improvements, such as supporting additional programming languages, integrating with CI/CD pipelines, and extending agent capabilities for real-time collaborative code review.

REFERENCES

I. Sharma & D. Rattan, "Code Quality Generated by AI Tools: A Review," IOSR Journal of Computer Engineering (IOSR-JCE), vol. 27, no. 3, pp. 55–68, 2025.

G. Viswanathan, "AI Agentic Scriptless Automation in Software Testing," International Journal of Computer Trends and Technology (IJCTT), vol. 72, no. 9, pp. 120–125, 2024.

S. K. Jangam & N. Karri, "AI Tools for Automating Code Reviews, Providing Contextual Feedback, and Improving the Efficiency of the Review Process," Advances in Interdisciplinary Journal of Computer Science and Technology (AIJCST), vol. 1, no. 1, pp. 36–42, 2025.

Official LangGraph Documentation — https://langchain-ai.github.io/langgraph/

Official AutoGen Documentation — https://microsoft.github.io/autogen/

FastAPI Documentation — https://fastapi.tiangolo.com/