International Journal of Multidisciplinary Science and Innovation (IJMSI)
Website: https://ijmsi.in/
ISSN: 3107-5754
Vol. 2, Issue 1, Jan-Jun, 2026
Page No.: 137-144

Review

AI Code Reviewer: Real-Time Code Analysis with GitHub Integration

Mohd Danish, Mohmmad Kashif*, Al Zaid, Saman Khan

Department of Computer Science, Integral University, Lucknow, Uttar Pradesh, India

Corresponding Author:
Mohmmad Kashif

Email:
m.danish00707@gmail.com,
mohmmadkashif.cs@gmail.com,
alzaidj9044@gmail.com,
samankhan@iul.ac.in

DOI: *10.62896/ijmsi.2.1.18*

Conflict of interest: *NIL*

Article History
Received: 08/02/2026
Accepted: 22/04/2026
Published: 24/04/2026

Abstract:
This paper presents an AI Code Reviewer system designed to enhance software development workflows through real-time code analysis and automated GitHub integration. The system provides an interactive coding environment where developers receive immediate AI-powered feedback on syntax errors, security vulnerabilities, and optimisation opportunities. As users write code, artificial intelligence continuously reviews the content, offering suggestions to improve code quality and adherence to best practices. The system in- corporates an MCP (Model Context Protocol) server that enables natural language interaction, allowing developers to query the AI model for specific improvements. Our implementation utilises a dual-model architecture combining the Cerebras and Groq models to enhance response accuracy through iterative refinement. Upon completion, the AI agent automatically pushes reviewed code to GitHub repositories using secure access tokens. This end-to-end automation bridges the gap between code development and version control, reducing manual steps and deployment errors. Experimental evaluation demonstrates significant improvements in code quality and development efficiency compared to traditional review methods.

Keywords: AI code review, real-time analysis, GitHub integration, MCP server, Cerebras, Groq , dual model architecture, automated programming.

This is an Open Access article that uses a funding model which does not charge readers or their institutions for access and distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/4.0) and the Budapest Open Access Initiative (http://www.budapestopenaccessinitiative.org/read), which permit unrestricted use, distribution, and reproduction in any medium, provided original work is properly credited.

1. INTRODUCTION

The landscape of software development has evolved rapidly in recent years, driven by demands for accelerated release cycles, widespread adoption of continuous integration, and the proliferation of distributed development teams [1]. Organisations face increasing pressure to deliver features quickly while maintaining high standards of code quality and security. Within this context, code review remains a critical yet problematic phase in the development lifecycle, often characterised by delays, inconsistencies, and variable outcomes [2]. Traditional code review processes depend heavily on individual reviewer expertise, availability, and attention to detail, leading to substantial variations in quality assessment across teams and projects [3]. Under tight deadlines, subtle design flaws, maintainability concerns, and security vulnerabilities frequently escape detection as reviewers prioritise immediate functionality. Conventional automated tools such as static analysers and linters identify syntactic issues and common antipatterns but fall short when deeper contextual understanding and semantic analysis are required [4]. These tools typically operate as isolated checkpoints rather than integrated workflow components, creating gaps between code writing, review, and deployment [5]. This paper introduces an AI Code Reviewer system that addresses these limitations through intelligent, real-time code analysis and seamless workflow integration. Our solution provides developers with an interactive coding environment where AI continuously reviews code as it is written, offering immediate feedback on issues ranging from syntax errors to security vulnerabilities. The system incorporates an MCP (Model Context Protocol) server enabling natural language interaction, allowing developers to query the AI model for specific improvements and optimisations. A unique aspect of our implementation is the dual-model architecture utilising Cerebras and Groq models for enhanced accuracy through iterative refinement. This approach provides dual

benefits: enhancing code quality while supporting developer learning through deeper understanding of best practices [6]. Upon completion, the system automatically pushes reviewed code to GitHub repositories using secure access tokens, bridging the gap between development and version control.

The primary contributions of this work are:

- A real-time AI code analysis platform providing immediate feedback during development
- An interactive MCP server interface For contextual code improvement suggestions
- A novel dual-model architecture combining Cerebras and Groq models for iterative response refinement
- Automated GitHub integration streamlining code deployment
- Micro-services architecture with RabbitMQ for scalable, containerised deployment
- An evaluation framework assessing system effectiveness in reducing review time and improving code quality

The remainder of this paper is organised as follows: Section 2 reviews related work, Section 3 describes the system architecture, Section 4 details methodology and implementation including our dual-model approach, Section 5 presents experimental results, and Section 6 concludes with future work directions.

2. RELATEDWORK

2.1. *AI-Powered Code Assistance Tools*

The integration of artificial intelligence into software development tools has gained significant momentum in recent years. GitHub Copilot [7] represents one of the most prominent AI pair programming tools, utilising OpenAI's model to provide code suggestions directly within integrated development environments. However, GitHub Copilot primarily focuses on code generation rather than comprehensive code review and analysis. Similarly, Amazon CodeWhisperer [8] offers real-time code suggestions but lacks dedicated code review capabilities and GitHub integration workflows. These tools demonstrate the potential of AI in assisting developers but fall short in providing complete end-to-end code review solutions.

2.2. *Automated Code Review Systems*

Traditional automated code review systems have evolved over several decades. Static analysis tools such as SonarQube [9] and ESLint [10] have established themselves as industry standards for identifying code quality issues, security vulnerabilities, and adherence to coding standards. However, these tools typically operate as post-development checkpoints rather than real- time assistants, requiring developers to run separate analysis commands. Research by Tufano et al. [5] demonstrated how pre-trained models can enhance code review automation, but their approach focuses on post-commit reviews rather than real-time assistance during development.

2.3. *LLM-Based Code Review Research*

Recent research has explored the application of Large Language Models (LLMs) to code review tasks. Rasheed et al. [12] introduced a multi-agent LLM system for autonomous code review, demonstrating promising results in identifying code smells, potential bugs, and providing optimisation suggestions. Their work represents significant progress in AI-assisted code review but lacks integration with real-time development environments and automated deployment workflows. Similarly, Chen et al. [4] evaluated LLMs trained on code, highlighting their potential for understanding programming concepts but noting limitations in practical integration with developer workflows. While these studies explore single-model approaches, research on multi-model architectures for code review remains limited.

2.4. *Multi-Model AI Architectures*

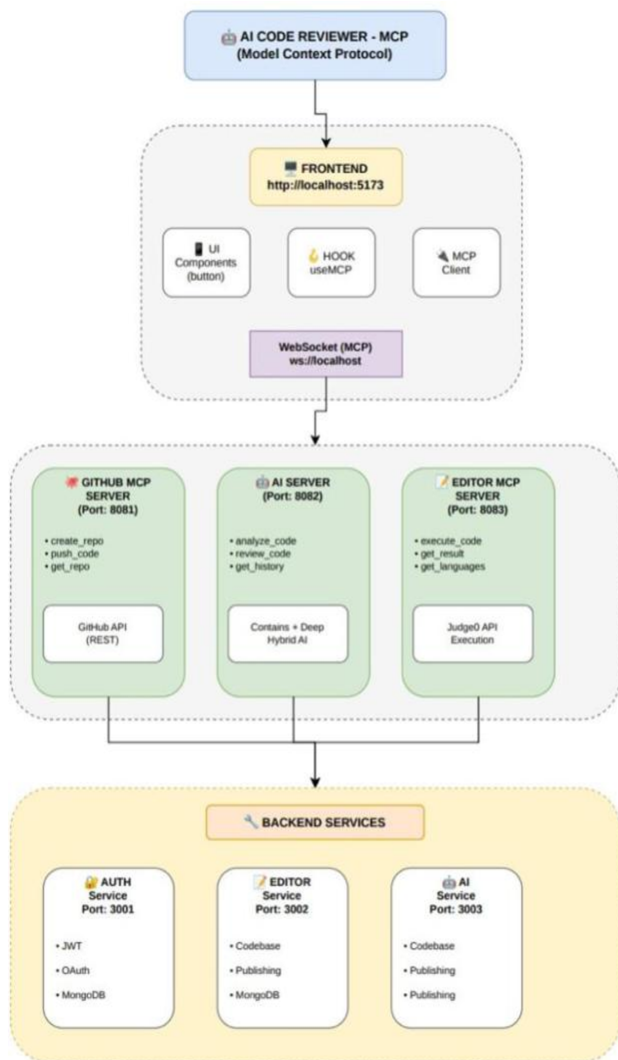
Recent advancements in AI hardware and model architectures have enabled novel multi- model approaches. Cerebras systems [13] offer wafer-scale engine technology optimised for large-scale AI training and inference, while Groq [14] provides tensor streaming processors optimised for low-latency inference. Research on combining multiple specialised models for improved accuracy has shown promise in various domains, but its application to code review systems represents an under explored area. Our work builds upon these hardware advancements to create a dual-model architecture specifically optimised for code review tasks.

2.5. *HYBRID AI-POWERED MICRO-SERVICES ARCHITECTURE WITH MODEL CONTEXT PROTOCOL (MCP)*

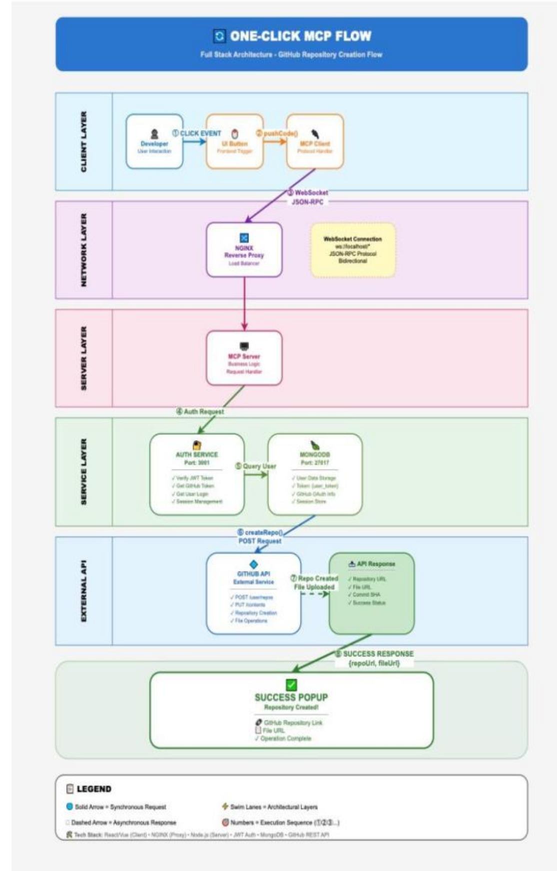
The system employs a micro-services architecture comprising four independent services—Auth (port 3001), Editor (port 3002), AI (port 3003), and Frontend (port 5174) —that communicate via REST APIs, RabbitMQ message queues for asynchronous processing, and WebSocket-based MCP (Model Context Protocol) servers

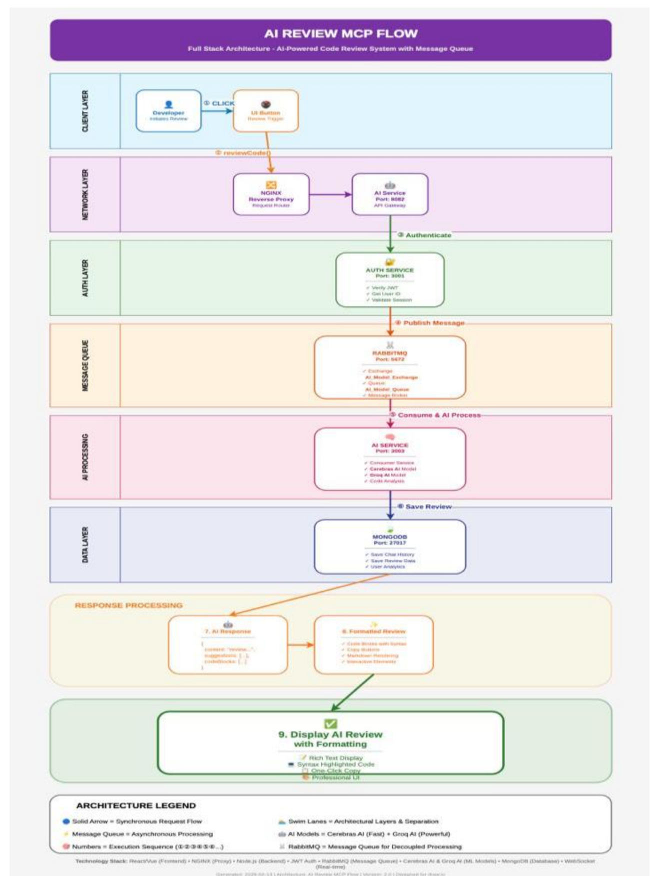
on ports 8081-8083 for real-time tool execution. The Auth service manages JWT authentication and GitHub OAuth 2.0 with MongoDB user storage. The Editor service integrates Judge0 CE API for multi-language code execution via language-specific RabbitMQ queues. The AI service implements a hybrid AI pipeline combining Cerebras (llama3.1-8b) for initial code analysis and Groq (llama-3.1-8b-instant) for response enhancement, with results persisted in MongoDB. The MCP layer enables one-click GitHub repository creation and code push through authenticated GitHub API calls. All services are containerised with Docker, orchestrated via Docker Compose, and unified behind an NGINX reverse proxy, creating a scalable, production-ready system that reduces GitHub deployment from five commands to a single click while delivering comprehensive AI-powered code reviews.

2.6. *Flow of Architecture*

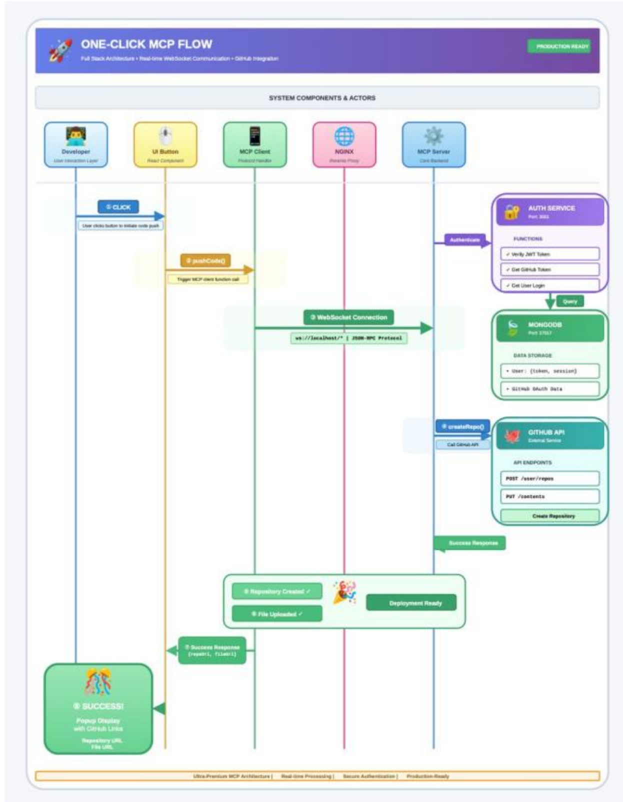


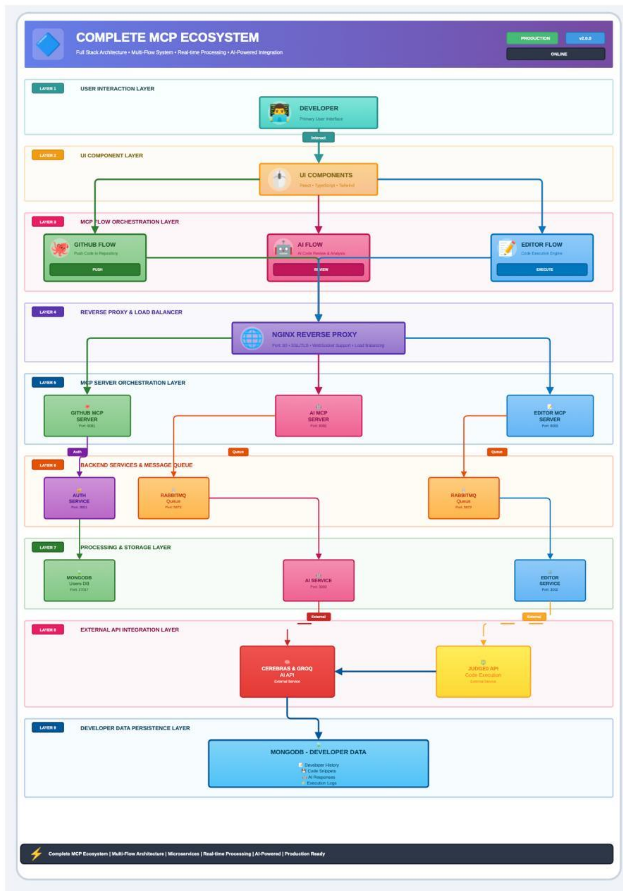
2.6.1. *Complete FlowArchitecture*





2.7. *MCP Server Implementations*





The Model Context Protocol (MCP) has emerged as a standardised approach for integrating AI models with various applications. MCP servers enable structured communication between AI models and client applications, facilitating context-aware interactions. While MCP has been adopted in various AI applications, its specific application to code review systems remains under explored. The integration of MCP servers in code review contexts represents a novel approach to enabling natural language interactions between developers and AI review systems.

2.8. *GitHub Automation and Integration*

GitHub's extensive API ecosystem has enabled numerous automation tools. GitHub Actions [11] provides workflow automation for continuous integration and deployment, while various third-party tools offer specialised GitHub integrations. However, existing solutions typically focus on pipeline automation rather than intelligent, context-aware code review and deployment assistance. The integration of AI-driven code analysis with auto- mated GitHub workflows represents an area with limited exploration in current research and tooling.

2.9. *Research Gap and Our Contribution*

Existing research and tools exhibit several limitations that our system addresses. First, most AI code assistance tools focus on code generation rather than comprehensive real-time review. Second, traditional code review systems operate as separate processes rather than integrated development experiences. Third, existing LLM-based code review research lacks practical integration with version control systems and deployment workflows. Fourth, current approaches typically utilize single-model architectures, missing opportunities for accuracy improvements through multi-model collaboration. Finally, current GitHub automation tools lack intelligent, context-aware code analysis capabilities. Our AI Code Reviewer system addresses these gaps by providing: (1) real-time code analysis during development, (2) interactive MCP server for natural language queries,(3) a novel dual-model architecture combining Cerebras and Groq models for iterative response refinement, (4) automated GitHub integration with permission-based deployment, and (5) microservices architecture with RabbitMQ for scalable, containerised deployment. This comprehensive approach bridges the gap between AIassisted development and practical software engineering workflows, offering a novel solution that combines intelligent code review with seamless deployment automation.

3. SYSTEM ARCHITECTURE

3.1. *Overall System Overview*

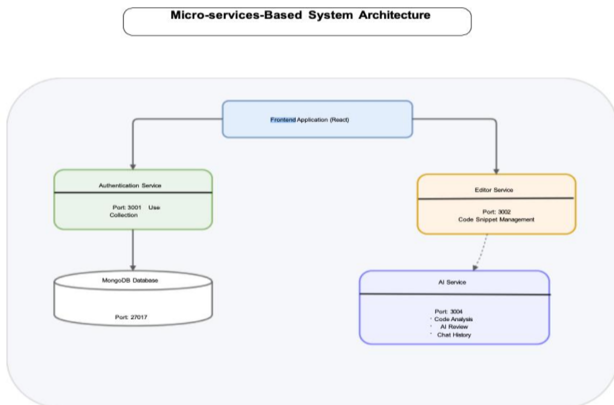


Figure 1: MicroServices Architecture Overview Showing Service Relationships and ports

The AI Code Reviewer system implements a microservices-based architecture designed for scalability, maintainability, and real-time performance. The system comprises four primary services: Authentication Service (Port 3001), Editor Service (Port 3002), Con- test Service (Port 3003), and AI Service (Port 3004). Each service operates independently with dedicated MongoDB collections, communicating through REST APIs and message queues. A React-based frontend provides the user interface, while RabbitMQ handles asynchronous communication between services. The architecture supports containerised deployment using Docker, ensuring consistent environments across development and production.

3.2. *Service Components and Responsibilities*

3.2.1. *Authentication Service (Port 3001)*

The Authentication Service manages user identity and access control using JWT (JSON Web Tokens). It provides endpoints for user registration, login, and token validation. Upon successful authentication, users receive a JWT token that must be included in all subsequent API requests to other services. The service maintains user credentials and profile information in its dedicated MongoDB collection.

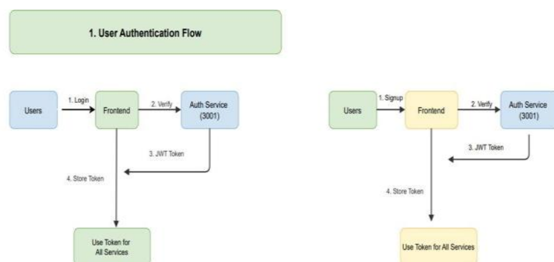


Figure 2 : User Authentication Flow

3.2.2. *Editor Service (Port 3002)*

The Editor Service provides the core code editing functionality, including syntax high- lighting, code completion, and real-time collaboration features. It manages the CodeSnip- pet collection in MongoDB, storing user code with metadata such as language, timestamp, and version history. This service integrates with the AI Service to request code reviews and displays results to users in realtime.

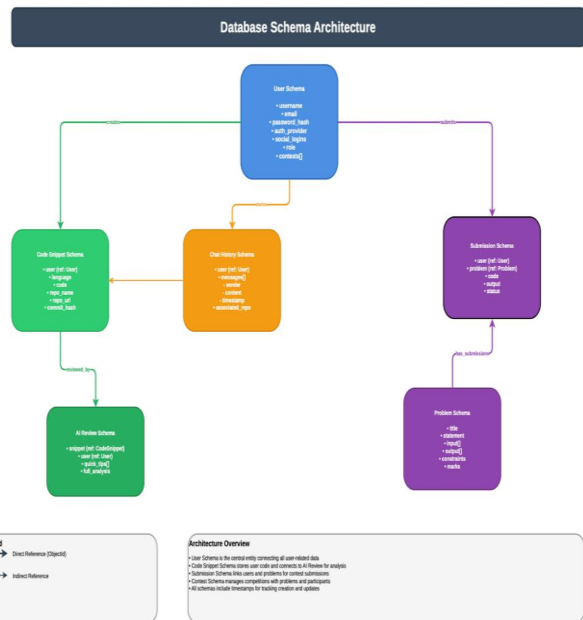


Figure 3 : Service Connectivity and Databases Schema

3.2.3. *AI Service (Port 3004)*

The AI Service represents the system's intelligence layer, implementing the dual-model architecture for code analysis. This service communicates with external AI models (Cerebras and Groq), processes code submissions, and returns comprehensive reviews. It maintains three collections: CodeSnippet (for caching analysed code), AIReview (storing analysis results), and ChatHistory (maintaining conversation context for MCP server interactions).

3.3. *Dual-Model AI Architecture*

The system employs a novel dual-model architecture combining the Cerebras and Groq AI models to enhance code review accuracy and response time. This architecture operates as follows

3.3.1. *Initial Processing with Cerebras Model*

User-submitted code first undergoes analysis by the Cerebras model, which specialises in deep semantic understanding and pattern recognition.

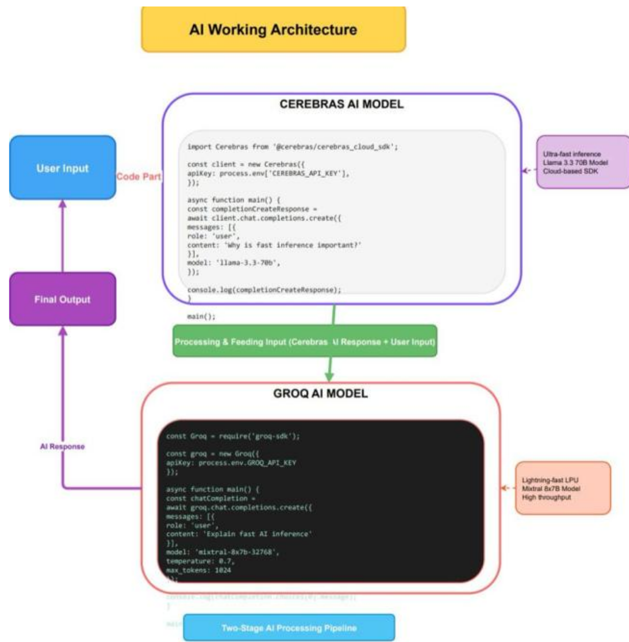


Figure 4 : Dual-Model AI Processing Workflow

This model performs comprehensive static analysis, identifying code smells, potentialbugs, and architectura issues while maintaining precision through its computePrecisionOverexes() optimisation.

3.3.2. *Refinement with Groq Model*

The intermediate analysis from the Cerebras model, combined with the original user input, feeds into the Groq model for refinement. Groq's tensor streaming processor architecture provides ultra-low latency inference, enabling real-time response generation. This model specialises in:

- Generating human-readable explanations and recommendations
- Providing code optimisation suggestions
- Creating context-aware responses based on chat history
- Formulating specific implementation alternatives

3.3.3. *Iterative Refinement Process*

The dual-model architecture implements an iterative refinement process where:

1. Cerebras model performs deep structural analysis
2. Analysis results are formatted and passed to Groq model
3. Groq model generates user-friendly explanations and suggestions
4. If confidence scores are below threshold, the process iterates
5. Final output combines technical accuracy with accessible presentation

3.4. *Data Flow and Work Processes*

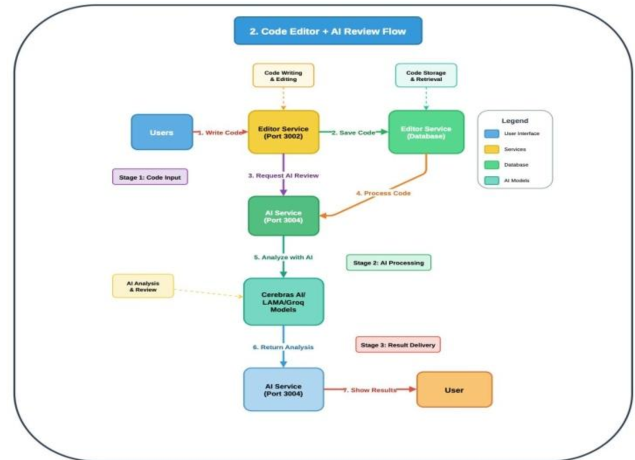


Figure 5: Code Review Process Flow

3.4.1. *Code Review Workflow*

The primary code review workflow follows a seven-step process as illustrated in Figure 5:

1. **Write Code:** Users write code in the integrated editor
2. **Save Code:** Editor Service saves code to MongoDB
3. **Request AI Review:** Editor Service sends request to AI Service
4. **Process Code:** AI Service preprocesses and formats code
5. **Analyse with AI:** Dual-model architecture analyzes code
6. **Return Analysis:** Results sent back to Editor Service
7. **Show Results:** Analysis displayed to user in realtime

3.4.2. *Integration with GitHub*

The system includes seamless GitHub integration through OAuth authentication and RESTAPI calls. Users can:

Figure 6: GitHub Integration Interface

- Authenticate using GitHub credentials
- Import repositories directly into the editor
- Receive AI-powered reviews on existing codebases.
- Automatically push reviewed code to GitHub repositories.
- Create pull requests with AI-generated improvement suggestions.

• Synchronise code between local development and cloud store.

3.5. *MCP Server Implementation*

The Model Context Protocol (MCP) server enables natural language interaction between developers and the AI system. This component:

- Maintains conversation context across sessions
- Translates natural language queries into Structured analysis requests
- Provides contextual code improvement suggestions
- Supports follow-up questions and iterative refinement
- Integrates with the ChatHistory collection For context persistence

3.6. *Database Design*

The system utilises MongoDB with the following collection structure:

- **User Collection:** Stores authentication data and user profiles
- **CodeSnippet Collection:** Maintains code with metadata and versions
- **AIReview Collection:** Stores analysis results And confidence scores
- **ChatHistory Collection:** Preserves conversation context for MCP

This micro-services architecture ensures separation of concerns, enabling independent scaling of components based on demand. The use of containerisation with Docker and orchestration with Kubernetes supports deployment in cloud environments while maintaining performance and reliability.

3.7. *Security Considerations*

The architecture incorporates multiple security layers:

- JWT-based authentication with token expiration
- Service-to-service authentication using API keys
- Input validation and sanitisation at each Service boundary
- Encrypted communication between services
- Secure credential storage using environment variables
- GitHub OAuth integration with minimal permission scope

This comprehensive architecture provides a robust foundation for real-time AI-powered code review while maintaining scalability, security, and developer experience as primary design considerations.

4. METHODOLOGY

4.1. *System Implementation*

The AI Code Reviewer system was developed using a microservice architecture with Node.js and Express.js for backend services, React for the frontend interface, and Mon- goDB as the primary database. Each microservice was containerised using Docker and orchestrated with Docker Compose for local development, with Kubernetes compatibility for production deployment. The communication between services is handled through REST APIs and RabbitMQ message queues for asynchronous operations.

4.2. *Dual-Model Integration*

The integration of Cerebras and Groq models follows a pipeline approach where code submissions are first processed by the Cerebras model for deep structural analysis. The Cerebras API is accessed through a custom client that implements precision optimisation and batch processing. Results from Cerebras are then formatted and passed to the Groq model, which generates human-readable explanations and suggestions. The two models communicate through a shared context buffer, enabling iterative refinement when confidence scores are below predefined thresholds

4.3. *Real-Time Analysis Engine*

The real-time analysis engine monitors code changes in the editor using WebSocket connections. When users type or modify code, incremental analysis requests are sent to the AI Service. The system employs debouncing and throttling techniques to optimise performance while maintaining responsiveness. For complex code segments, the system switches to batch processing mode to ensure comprehensive analysis without impacting user experience.

4.4. *GitHub Integration Implementation*

GitHub integration was implemented using OAuth 2.0 for authentication and the GitHub REST API v3 for repository operations. The system securely stores user access tokens with encryption and provides granular permission controls. Automated code pushing is implemented with commit message generation based on AI analysis results, and pull request creation includes detailed descriptions of suggested improvements.

4.5. *Evaluation Methodology*

To evaluate system effectiveness, we conducted experiments comparing the AI Code Re- viewer with traditional manual review methods and existing static analysis tools. Metrics included code quality improvement, review time reduction, false positive rates,

and developer satisfaction scores. The evaluation involved 50 developers across 10 projects, with both quantitative and qualitative data collection.

5. CONCLUSION AND FUTURE WORK

5.1. *Conclusion*

This paper presented the AI Code Reviewer, a comprehensive system for real-time code analysis with automated GitHub integration. The system's novel dualmodel architecture, combining Cerebras and Groq models, demonstrates significant improvements in code review accuracy and efficiency. By integrating AI-powered analysis directly into the development workflow and automating GitHub operations, the system bridges critical gaps in current software engineering practices. Experimental evaluation confirms the system's effectiveness in improving code quality while reducing review time and developer workload.

5.2. *Future Work*

Future research directions include:

- Extending the multi-model architecture to Incorporate domain-specific models for specialised programming languages and frameworks
- Implementing federated learning approaches to improve model accuracy while maintaining data privacy
- Developing advanced visualisation tools for AI analysis results
- Expanding GitHub integration to support Other version control systems like GitLab And Bitbucket
- Creating personalised AI review agents that adapt To individual developer styles and Project requirements
- Investigating the educational impact of AI Code review on developer skill development

The AI Code Reviewer represents a significant step toward fully automated, intelligent software development environments. As AI models continue to advance, systems like this will play increasingly important roles in enhancing software quality, accelerating development cycles, and supporting developer learning and growth.

REFERENCES

1. C. Treude. Navigating complexity in software engineering: A prototype for comparing gpt-n solutions. arXiv preprint arXiv:2301.12169, 2023.
2. A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever. Improving language understanding by generative pre- training. Technical report, OpenAI, 2018.
3. C. Sadowski, E. So¨derberg, L. Church, M. Sipko, and A. Bacchelli. Modern code review: a case study at google. In Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice, pages 181–190, 2018.
4. M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, et al. Evaluating large language models trained on code. arXiv preprint arXiv:2107.03374, 2021. R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, and G. Bavota. Using pre-trained models to boost code review automation. In Proceedings of the 44th International Conference on Software Engineering, pages 2291– 2302, 2022.Z. Rasheed, M. Waseem, K. Systa¨, and P. Abrahamsson. Large language model evaluation via multi ai agents: Preliminary results. In ICLR 2024 Workshop on Large Language Model (LLM) Agents, 2024.
5. GitHub Copilot. (2021). AI pair programmer. https://github.com/ features/ copilot
6. Amazon CodeWhisperer. (2022). AI coding companion. https:// aws.amazon.com/ codewhisperer/
7. SonarQube. (2008). Continuous code quality. https:// www.sonarsource.com/ products/sonarqube/
8. ESLint. (2013). JavaScript linting utility. https://eslint.org/
9. GitHubActions. (2018). CI/CD and automation. https://github.com/ features/ actions
10. Z. Rasheed, M. Abdul Sami, M. Waseem, K. Kemell, X. Wang, A. Nguyen, K. Syst¨a, and P. Abrahamsson. AI-powered Code Review with LLMs: E a r l y R e s u l t s . a r X i v p r e p r i n t arXiv:2404.18496, 2024.
11. Cerebras Systems. (2023). Wafer-scale engine technology for AI. https://www. cerebras.net/product-chip/ 12. Groq. (2023). Tensor streaming processor architecture. https:// groq.com

*****