International Journal of Scientific Research and Engineering Development-– Volume 9 Issue 4, Jul – Aug 2026
Available at www.ijsred.com

RESEARCH ARTICLE OPEN ACCESS

Intelligent Multi-Agent AI System with Predictive Analysis

Suresh
Dept of ISE
Don Bosco Institute of Technology
Bangalore, India
246suri@gmail.com

Yashas R Gowda
Dept of ISE
Don Bosco Institute of Technology
Bangalore, India
yashasgowda1901@gmail.com

Tejas A
Dept of ISE
Don Bosco Institute of Technology
Bangalore, India
tejasannappa0@gmail.com

Yashwanth L
Dept of ISE
Don Bosco Institute of Technology
Bangalore, India
yashwanthyash9483@gmail.com

Bhavya B V
Assistant Professor
Dept of ISE
Don Bosco Institute of Technology
Bangalore, India
bhavya.bv@dbit.co.in

----------**********----------

Abstract:

Efficient code review procedures are essential in modern software development in order to guarantee software quality, security, and maintainability. Traditional approaches to code review are usually time-consuming, rely on the expertise of the reviewers, and are unable to give thorough feedback with respect to a range of quality aspects. While static analysis tools can detect pre-defined coding problems, they generally do not have a good understanding of the context and do not offer specific recommendations. In order to overcome these shortcomings, this paper presents ReviewSphere AI, a multi-agent automated code review framework driven by Large Language Models (LLMs). The system makes use of dedicated AI agents which separately examine the source code for security vulnerabilities, code quality problems, and performance inefficiencies. These agents carry out their tasks at the same time and produce structured findings that are combined into a single review report including severity levels, explanations, suggestions for improvement, and overall quality scores. The framework is built using FastAPI for the backend, React for the interactive user interface, and Ollama for local LLM inference, thus allowing for a secure and cost-efficient deployment without having to depend on cloud-based AI services. The progress of the review is made available in real time via Server-Sent Events (SSE), which improves transparency and the user experience throughout the analysis process. Because of its modular design, the framework can also incorporate further review agents, which makes it scalable and capable of adapting to changing software engineering needs. By combining parallel multi-agent analysis with intelligent language models, ReviewSphere AI offers a comprehensive, context-aware, and extensible approach to automated code review, helping developers to create more secure, efficient, and maintainable software while reducing the amount of manual review required.

----------**********----------

I. INTRODUCTION

With the rapid growth of large-scale applications, distributed systems, and the use of continuous integration, software development has become more and more complex. As software projects move forward, it has become a major challenge for development teams to keep up the required standards in terms of code quality, security, and performance. One of the most effective practices in software engineering is code review since it helps to detect defects, improves maintainability, ensures that coding standards are followed, and reduces security vulnerabilities before deployment. Yet traditional manual code review is generally time-consuming, demands a lot of resources, and relies heavily on the knowledge and availability of the people carrying out the review, which makes it hard to scale in today's agile development environments.

In order to overcome these limitations, developers usually use automated static analysis

tools which are able to identify syntax errors, code smells, and violations of predefined programming rules. While such tools do improve development efficiency, they mainly depend on fixed rules and pattern matching and thus lack the capacity to understand the functional context, the intended purpose of the code, and the logical relationships present in the source code. As a result, developers often get a number of isolated warnings without any clear explanations or practical advice on how to improve the code.

Recent progress in Large Language Models (LLMs) has greatly improved artificial intelligence's ability to understand programming languages, software architecture, and the intent of developers. Unlike traditional static analyzers, LLMs are able to interpret code in a semantic way, detect complex programming problems, and produce explanations that are readable to humans along with improvement suggestions which are aware of the context. Because of these capabilities, new possibilities have arisen for creating intelligent code review systems that give more thorough and developer-friendly feedback.

The paper introduces ReviewSphere AI, a multi-agent system for automated code review which uses specialized AI agents to examine source code from a variety of aspects of software quality. The framework makes use of separate Security, Quality, and Performance agents that simultaneously assess the submitted code by means of a Large Language Model that is locally deployed. Each of these agents produces structured results consisting of the issues it has detected, their severity levels, explanations and suggested solutions, all of which are combined into a single review report. The local model was run using Ollama, and Server-Sent Events were used to provide users with updates regarding the review progress in real-time. This approach allowed to design a modular system that is easy to support and scale while keeping the code under review and the model running locally to avoid relying on external services.

The purpose of this research was to design and implement an automated code review system that would be able to analyze the source code from different viewpoints and give helpful suggestions to the developers. This system would help to increase the quality of software products and reduce the time and effort cost associated with traditional code reviews. The proposed framework achieves thorough code analysis by combining the reasoning abilities of Large Language Models with a parallel multi-agent architecture, thus helping developers to detect security vulnerabilities, improve maintainability.

II. RELATEDWORK

Recent advances in artificial intelligence have had a major impact on automated code review by combining traditional software analysismethods with Large Language Models (LLMs). Tools based on conventional static analysis—such as SonarQube, PMD, ESLint, and Pylint—are commonly used for detecting syntax errors, code smells, security vulnerabilities, and breaches of coding standards. Although these tools enhance software quality by picking up on predefined problems, they mainly depend on rule-based analysis and do not have the capability to understand the semantics of the code, the intent of the developer, or the contextual relationships among the various parts of the source code. Because of this, developers typically get a number of isolated warnings without enough explanations or any practical recommendations.

A number of recent studies have looked into using Large Language Models, especially those based on GPT, for the purpose of automating code reviews. These methods show that LLMs are capable of understanding programming logic, explaining the problems they have detected in natural language, suggesting improvements to the code, and aiding educational programming environments by giving individualized feedback. Unlike traditional static analyzers, LLM-based systems provide a better understanding of context and make recommendations that are more acceptable to developers. The problem is that most of the current solutions rely on a single AI model to carry out all the review tasks, which limits their ability to carry out specialized analysis covering a range of software quality aspects.

Research into multi-agent AI systems has led to a collaborative method in which a number of specialized agents work together to deal with complex software engineering problems. Rather than giving all the code review tasks to one single model, separate agents independently assess various aspe. cts of the source code, including security, maintainability, performance, and adherence to coding standards. This kind of architecture increases modularity, permits parallel execution, and allows each agent to concentrate on analysis that is specific to its area of expertise. However, many of the existing frameworks are still mainly conceptual or experimental and typically depend on cloud-hosted language models, which gives rise to concerns about response latency, scalability, privacy, and deployment cost.

The literature surveyed also stresses the need for structured review reports, explainable AI feedback, and smooth integration with current development workflows. Even though recent systems have achieved a great deal in the area of intelligent code analysis, only a small number manage to combine local LLM deployment, real-time review monitoring, parallel multi-agent execution, and thorough structured reporting in one framework.

Inspired by these observations, ReviewSphere AI has developed a modular multi-agent code review framework which incorporates dedicated Security, Quality, and Performance agents that are driven by a large language model locally deployed via Ollama. The agents are run at the same time, their findings are combined into a single review report that includes severity levels and suggestions for improvements, and real-time updates on progress are provided using Server-Sent Events. Since the approach combines the advantages that have been identified in previous research while overcoming their practical limitations, ReviewSphere AI seeks to offer a scalable, context-aware, and privacy-preserving automated code review solution appropriate for use in modern software development environments.

III. PROPOSEDSYSTEM

The proposed system, **ReviewSphere AI**, is a multi-agent intelligent code review framework designed to automate software quality assessment using Large Language Models (LLMs)

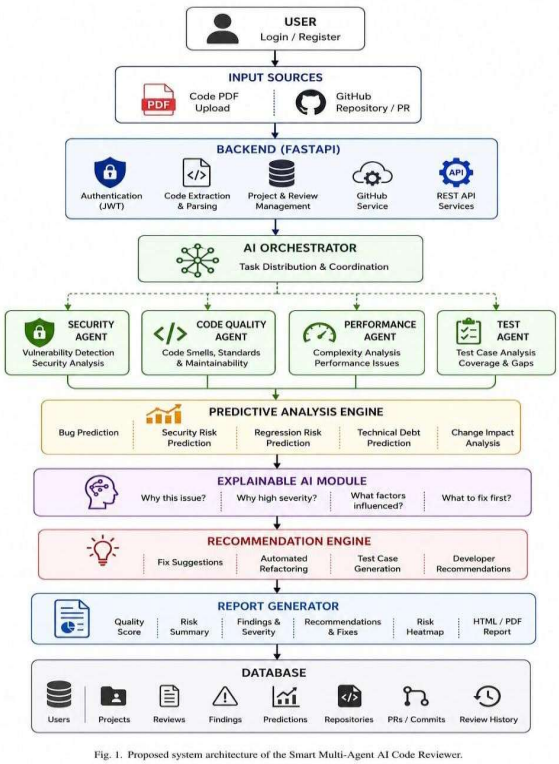


Fig. 1. Proposed system architecture of the Smart Multi-Agent AI Code Reviewer.

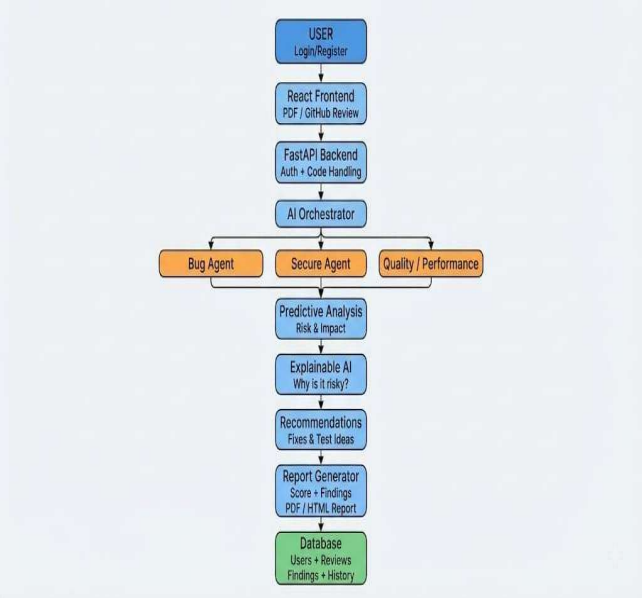
Rather than using traditional static analysis tools which depend entirely on predefined rules, the proposed framework carries out context-aware analysis by making use of a number of specialised AI agents, each of which independently assesses the source code from different viewpoints regarding software quality. The main aim of the system is to help developers by giving them thorough, explainable and actionable feedback on their code while at the same time reducing the amount of time and effort needed for manual inspection.

ReviewSphere AI is implemented as a web-based application using React to implement the frontend and FastAPI to manage backend operations. The language model is run locally using Ollama, while the multi-agent review engine is responsible for analyzing the source code. When developers interact with the code editor, their inputs are sent to the backend, which routes the request to the relevant agents to review the code.

The first agent, Security Agent, analyses the source code looking for potential vulnerabilities, insecure practices, and possible threats. The second agent, Quality Agent, focuses on code quality, including coding standards, readability, and maintainability. The third agent, Performance Agent,evaluate performance-related issues such as inefficient algorithms, extra computations, or anything else that could add to the run-time overhead of the code. All the agents come together to provide an in-depth code review, which is subsequently compiled into a report for the user.

However, the survey results in the literature emphasize the need for structured report generation, explainable feedback, and the ability to fit into existing software development processes. While recent research has produced compelling methodologies for code review, very few of these solutions embody all the critical elements – local LLM execution, live feedback, multi-agent architecture, and structured report generation. In addition, many of the implementations are still conceptual or require cloud-based language models, potentially introducing latency, scalability, privacy, and cost considerations. ReviewSphere AI offers a scalable, extensible, and privacy-preserving approach to intelligent automated code review by combining parallel multi-agent analysis, structured reporting, and context-aware reasoning.

IV. FLOWDIAGRAM



System Flow:The ReviewSphere AI system uses a multi-stage workflow when carrying out automated code analysis. The developer provides the source code via the frontend, which is based on React and includes the Monaco editor. The code review request is then forwarded to the FastAPI backend, where the review process is managed and the dedicated AIagents are coordinated. The Security Agent checks the code for possible security vulnerabilities, the Quality Agent looks at the code's structure, readability, and maintainability, and the Performance Agent spots any possible problems relating to performance and efficiency. The various agents work on their own and can carry out their tasks in parallel in order to shorten the total review time. The backend gathers and combines the responses from the agents and produces a structured review result that includes scores, the detected issues, severity levels, explanations, and suggested fixes. During the review process, real-time processing events can also be sent to the frontend. Lastly, the results generated are displayed in the developer dashboard, enabling the developer to see the problems that have been identified and to improve the code that has been submitted.

V.METHODOLOGY

The proposed ReviewSphere AI system uses a multi-agent architecture to automate source code review by dividing the analysis process into specialized tasks. The methodology consists of source code submission, preprocessing, parallel analysis by specialized agents, result aggregation, structured report generation, and presentation of the final feedback to the developer. The system is designed to combine the capabilities of multiple AI agents instead of relying on a single analysis process, allowing different aspects of code quality to be examined independently.

A. Code Submission and Processing

The review process begins when the developer submits source code through the React-based frontend. The Monaco Editor provides an interactive environment for entering or modifying the code. The request for a review is sent to the FastAPI backend via an API interface; the backend then verifies the request and starts the review process. By separating

the frontend from the backend, the analysis engine is able to function independently of the user interface.

B. Multi-Agent Code Analysis

Having received the code, the backend divides the review process between three agents. First, a Security Agent scans the code for possible vulnerabilities and unsafe practices. Second, a Quality Agent takes care of the code structure and overall quality. Finally, a Performance Agent analyzes the code for possible performance issues and computational inefficiencies.

C. Parallel Agent Processing

The processing of the code by agents is parallel and non-serial. This means that once the backend triggers the processing, the three agents can analyze one and the same code snippet at the same time. As opposed to the serial processing, such a method reduces the overhead of waiting for the results of another agent and allows delivering the review results faster. The Agent Manager manages the three agents and collects their results.

D. AI-Based Analysis and Response Generation

Each agent is provided with the input source code along with the description of the analysis objective according to the agent's role. It allows fulfilling the separation of concerns when analyzing the source code. In particular, the Security, Quality, and Performance agents focus on potential security issues, code quality and architecture, and performance-related patterns or issues, respectively.Each agent returns the response with the identified issues, their severity level, and possible solutions once the analysis is completed. Since each agent operates independently and the analysis objectives are isolated from each other, the analysis can be done in parallel to save time and computational resources.

E. Aggregation of Results and Structured Report Generation

When all agents finish the analysis, the backend module merges the results into one review result that follows the unified structure. During this step, the results are structured to form the final review document containing category-specificscores, issues, severity levels, explanations, and solutions.In such a way, the review report is prepared to be further visualized in the user interface as a complete structured document that provides an end-user with a clear understanding of the review findings..The final report includes scores by category, the detected issues, the severity levels, the explanations and the suggested fixes. This structured form also enables the frontend to show the review results in a clear and organized way.

F. Real-Time Review Events

The system makes use of Server-Sent Events (SSE) in order to give real-time information concerning the progress of the review process; during the analysis, events such as the start and completion by an agent can be sent from the backend to the frontend. This means that the developer can watch the progress of the various agents rather than having to wait until the full review response is received before getting any feedback.

G. A review of the visualization and developer feedback

Once the analysis has been carried out, the structured review report is sent back to the React frontend. The developer dashboard displays the results through category-wise scores, details about the issues, severity indicators, explanations and suggested fixes. This way, developers are able to spot significant problems rapidly and can use the feedback produced to improve their source code. The overall methodology thus brings together specialized AI analysis, parallel execution, structured reporting and interactive visualization in a single automated code review process.

VI. FUTURE WORK

The proposed ReviewSphere AI system can be expanded in various ways in the future to make it a more complete and powerful software engineering ecosystem. First of all, the system can be expanded to allow for multi-agent verification of any code issues identified by a single agent to address potential false positives and increase the level of confidence in each identified problem and recommended code changes. The system can also be extended to provide confidence scores,

explainability, and risk assessment details for each review agent's findings, thus giving developers a more comprehensive overview of each potential issue.

In the future, the system can also be extended to allow for automated code fix suggestion and implementation, wherein the agent that identifies code problems can also suggest code changes or automatically implement them with the developer's approval. Another interesting extension would be to implement a code-review history tracker that would let developers track their code quality improvements across multiple code reviews, thus visualizing the improvements in security, quality, performance, and other aspects.

The system's functionality can also be expanded to include a resource-efficiency and cost estimation agent that would be able to identify wasteful code patterns that lead to unnecessary CPU, memory, or other computer resource usage as well as extra database operations, API requests, and other expensive operations. The system can be connected to a cloud platform to estimate how much a given piece of code would cost in terms of infrastructure costs, thus acting as a static code analyzer and security code scanner with an extended set of capabilities. More advanced explainable AI features can also be added, which would go into detail for each finding, thus outlining what evidence points towards the identified problem and what code changes can be implemented to resolve the issue.

beyond the capabilities of a traditional code review assistant. The system can also be extended to analyze larger codebases and to operate alongside current static code analysis tools and automated code-review systems. In the future, it would be interesting to test the system on larger and more diverse codebases while comparing the results of ReviewSphere AI to the findings of traditional code auditing methods and see how the findings match up in terms of accuracy, false-positive rates, speed, quality of recommendations, and quality of code improvements. All of these additions and improvements would make ReviewSphere AI a comprehensive multi-agent code analysis system that goes far

VII. RESULTS AND DISSCUSSIONS

The AI prototype known as ReviewSphere shows that it is possible to use a multi-agent architecture for automated source code review. The system manages to combine the frontend, the backend, and the various specialised AI agents into a single review workflow. Source code submitted via the web interface is passed on to the FastAPI backend, where it is dealt with by the Security, Quality, and Performance agents. The fact that these agents operate independently enables different aspects of code quality to be examined within the same review process The feedback produced by each individual agent is transformed into a structured format which includes the review categories,

the issues identified, the severity levels, the explanations, the scores, and the recommended fixes. This structured form of the feedback is easier to process and present than unstructured textual responses. Additionally, the system offers an evaluation broken down by category, enabling developers to tell the difference between security problems and those relating to quality and performance.

The use of parallel agent execution gives the proposed system a significant benefit. The three analysis tasks being mostly independent means that they can be carried out at the same time instead of one after another. This approach avoids unnecessary delays between the various analyses and forms a basis for enhancing the total response time of the review system. Moreover, the incorporation of Server-Sent Events further enhances the interaction by enabling the frontend to receive in real time information about the progress of the review, such as when an agent is started and when it is completed.

The frontend includes an interactive, developer-focused interface based on the Monaco code editor and review dashboard. The results generated are shown in the form of scores, details about issues, severity markings, and recommendations for corrective action. This way, developers are able to

spot significant problems without having to manually look through the entire source code review output. Because the approach combines specialized analysis with structured visualization, it is more systematic than simply using a general-purpose code review response.

The system has certain limitations as well. The quality of the feedback produced depends on the capabilities of the underlying language model and on the instructions given to each agent. Any recommendations generated by AI will need to be verified by a developer before they are used in production code. Moreover, in order to quantitatively measure detection accuracy, false-positive rates, response latency, and scalability a thorough evaluation using a larger set of source code samples and existing code-review benchmarks is necessary.

CONCLUSION

The current study introduced ReviewSphere AI, a multi-agent artificial intelligence-based system for code review, which focuses on addressing various security, quality, and performance-related issues related to the development and use of source code. This system utilizes a set of specialized agents for the improved detection, identification, and resolution of issues automatically detected during an analysis of the specific lines of code and corresponding files. The paper demonstrated the architectural design of the code review system, which employs React and FastAPI-based interfaces for managing communication between multiple agents and providing appropriate feedback to the developers in the form of issues detected, their levels of severity, scores, descriptions, and suggested resolutions. The current study's implementation demonstrated how multi-agent AI could be used in practice to offer improved automated code review and developer assistance by dividing the required tasks into several agent-specific instructions, making the process more transparent and structured.

The future research directions should build on this foundation by expanding the scope and variety of the agents involved in the system, increasing the number of programming languages for which code review is available, and making the system suggestions more reliable and trustworthy. Other improvements would include the addition of version control system features, such as reviewing code changes and commits, supporting continuous integration and delivery (CI/CD) platforms, analyzing the code at a repository level, and implementing automatic verification of suggested modifications The future study may also involve a larger-scale experiment, which would utilize benchmark datasets, to evaluate the performance of the multi-agent code review system. The key performance indicators to be measured include the accuracy of issue detection, false-positive rates, response time, and scalability to demonstrate the effectiveness, efficiency, and practical applicability of the solution compared to the state-of-the-art code review approaches.

REFERENCES

[] T. Sharanarthi and S. Polineni, "Real-Time Adaptive Code Analysis with a Self-Learning Multi-Agent Framework: A Retrieval-Augmented Reinforcement Learning Approach," in 2025 International Conference on Artificial Intelligence and Digital Ethics (ICAIDE), 2025.

[] P. Thongtanunam, C. Pornprasit, and C. Tantithamthavorn, "Auto-Transform: Automated Code Transformation to Support Modern Code Review Process," in Proc. of ICSE, 2022, pp. 237-248.

[] H. Y. Lin, P. Thongtanunam, C. Treude, and W. Charoenwet, "Improving Automated Code Reviews: Learning from Experience," in 2024 IEEE/ACM 21st International Conference on Mining Software Repositories (MSR), 2024.

[] X. Zhou, K. Kim, B. Xu, D. Han, J. He, and D. Lo, "Generation-based code review automation: How far are we?" in 2023 IEEE/ACM 31st International Conference on Program Comprehension (ICPC), 2023, pp. 215-226.

[] T. Kakimoto, H. Uwano, H. Inayoshi, and A. Monden, "Evaluating the Source Code Review Performance of LLM-based AI Chatbots," in 2025 IEEE/ACM 47th International Conference on

Software Engineering: Companion Proceedings (ICSE-Companion), 2025.

[] J. K. Siow, C. Gao, L. Fan, S. Chen, and Y. Liu, "Core: Automating review recommendation for codechanges," in 2020 IEEE 27thInternational Conference on Software Analysis, Evolution and Reengineering (SANER), 2020, pp. 284-295.

[] J. He, C. Treude, and D. Lo, "Llm-based multi-agent systems for software engineering: Literature review, vision and the road ahead," 2024.

[] S. Ramesh, J. Bose, H. Singh, A. Raghavan, S. R. Chowdhury, G. Sridhara, N. Saini, and R. Britto, "Automated Code Review Using Large Language Models at Ericsson: An Experience Report," in 2025 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2025.

[] R. Tufano, L. Pascarella, M. Tufano, D. Poshyvanyk, and G. Bavota, "Code review automation by leveraging pre-trained representations," in "Towards automating code review activities," 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), 2021, pp. 163-174.

[] R. Tufano, S. Masiero, A. Mastropaolo, L. Pascarella, D. Poshyvanyk, and G. Bavota, "Using pre-trained models to boost code review automation," in Proc. 44th ICSE, 2022, pp. 2291-2302.

[] U. Cihan, V. Haratian, A. Içöz, M. K. Gül, Ö. Devran, E. F. Bayendur, B.M. Uçar, and E. Tüzün, "Automated Code Review In Practice," in 2025 IEEE/ACM 47th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), 2025, pp. 425-436.

[] D.-K. Lee and I. Joe, "A GPT-Based Code Review SystemWith Accurate Feedback for Programming Education," IEEE Access, vol. 13, pp. 105724-105737, 2025.

[] C. Adapa, A. A. R. K., S. S. Avulamanda, and A. Victor, "AI-Powered Code Review Assistant for Streamlining Pull Request Merging," in 2024 IEEE International Conference for Women

[] J. Venkatarao, M. Harshitha, J. Manvitha, K. Kumari, and G. Deekshitha, "Automated Code Review and Quality Assurance System using Static Analysis and Complexity Evaluation in Flask-based Web Application," in 2026 7th International Conference on Mobile Computing and Sustainable Informatics (ICMCSI), 2026.

[] Z. Li, S. Lu, D. Guo, N. Duan, S. Jannu, G. Jenks, D. Majumder, J. Green, A. Svyatkovskiy, S. Fu, and N. Sundaresan, "Automating code review activities by large-scale pre-training," in Proc. of ESEC/FSE ,2022, pp. 1035-1047.

