International Journal of Emerging Research in Engineering and Technology
Pearl Blue Research Group | Volume 7 Issue 2 PP 135-143, 2026
ISSN: 3050-922X | https://doi.org/10.63282/3050-922X.IJERET-V7I2P117

Original Article

A Large Language Model-Powered Agentic Framework for Interactive Coding Assistance and Automated Error Fixing

Dr. Nilesh Jain
Associate Professor, Department of Computer Sciences and Applications, Mandsaur University, Mandsaur.

Received On: 16/03/2026 Revised On: 15/04/2026 Accepted On: 23/04/2026 Published On: 29/04/2026

Abstract - *The inefficiencies in software development are still prevailing in modern software development, as developers are taking time in order to find syntax errors, warnings, logic errors, and security bugs in various programming languages. Conventional support systems are unable to offer much feedback, adaptive performance, and the balance between detection and automated correction. To address these issues, this paper proposes an agentic framework based on Large Language Models (LLMs) to provide interactive code assistance and auto error correction, with the implementation based on the MERN stack and multi-model optimization (e.g., Groq, LLaMA) to achieve better accuracy, response time, and token efficiency. The system allows any code written in Python, JavaScript, Java, C#, C++ to be submitted in a secure way, with real-time analysis, highlighting of issues, and proposed context-aware fixes, and making automatic corrections to the code performed by the LLM. Performance evaluation is excellent at detection with different success rates of correction and highest success rates in Python (91.6% success rate) and lower success rates in more complex languages like C++ (80%). Such measures as cyclomatic complexity, fix success ratio, token consumption, and variability of fix time (400ms-7688ms) are saved in MongoDB and displayed on React dashboards. Findings show that the framework has a high level of consistency in error detection and interactive feedback but more work is required to stabilize the correction efficiency and lower the variability of fix times. The significance of this system is in its flexibility in cross-language, real-time debugging, and built-in performance analysis, and has the potential to significantly shorten software development cycles, which is evident as a clear advantage over the old-fashioned tools of the traditional form of the static analysis.*

Keywords - *MERN Stack, Large Language Models (LLM), Groq AI, Interactive Coding Assistance, Automated Error Fixing, Software Metrics, Intelligent Debugging, Code Analysis.*

1. Introduction

The impact of Artificial Intelligence (AI) does not belong to the sphere of software engineering [1], but it spreads to many industries and transforms the classical processes [2]. The AI can be applied to assist in the diagnostics, drug discovery and personal therapy in the sphere of healthcare[3][4]. Financial institutions apply AI to such spheres as algorithmic trading, risk assessment, and fraud detection [5][6][7][8]. Autonomous cars apply AI to find their way through difficult environments, whereas virtual assistants apply natural language processing to enhance human-computer interaction[9][10][11]. Nonetheless, AI has been significant in radicalizing the rapidly paced industry of software engineering [12]. AI and software engineering have established a symbiotic relationship in which smart systems improve and accelerate the traditional development process, ushering in an era of unprecedented innovation [13].

Language has become an important part of human communication, self-expression, and interaction with technology. The growing need to have machines perform complex language operations, including translation, summarization, information search, conversational interactions, etc. is what prompts the necessity of generalized models [14][15]. Recent progress in language models has been extremely impressive due to transformers, better processing power, and the accessibility of large volumes of training data [16][17]. These improvements have made it possible to build LLMs that can be able to approximate human performance on numerous tasks leading to a radical shift[18]. LLMs are the most recent advanced AI to understand and generate text with sensible conversation and generalization to numerous tasks [19][20][21].

AI has experienced a significant paradigm shift in its development since to agentic frameworks [22][23] that transform the way AI systems interact with complex environments and make independent decisions. According to recent rigorous studies, these frameworks generate a beautiful 52.3% improvement of job completion through a range of industrial uses, particularly in situations where real-time adjustment is necessary [24][25]. This evolution is a significant advancement over traditional rule-based systems, which have traditionally been difficult in dynamic environments and have efficiency plateaus of approximately 58% in complex decision-making conditions [26].

The main benefit of bug tracking is that it provides accurate essential data concerning the development request (bugs and improvements, including often cloudy areas) and its status. When creating a product or even simply the "next release," the master list of backlogs often referred to as the backlog provides helpful information. Error systems can be

used to provide product reports in a commercial context. Bugs are fixed in production by programmers. However, because the parameters may have varying weights and difficulty, this might occasionally result in inaccurate findings. The complexity of rectifying the problem won't be immediately impacted by its severity [27]. In this study, a Large Language Model (LLMs) powered agentic framework for interactive coding support and automated error resolution[28]. Today's software development teams are struggling with debugging, error resolution, and code mentorship, and traditional development tools take a more static, limited approach to providing support. This contribution extends past simply providing passive code suggestions, they promise structured, interactive, and scalable knowledge that can potentially enhance developer productivity and accelerate the time of error repair and overall software development capabilities.

The major contributions of this research are summarized:

- MERN Implementation with LLM: Developed a functional app using MERN stack, which provided sufficient communication between the React frontend and the Express.js backend. This project has implemented large language models (LLMs) to do real-time automatic code analysis and error correction.
- Interactive Code Automated Assistance: The ability of allowed users to post code snippets in an interactive way and provide context-sensitive analysis, debug and automated fix suggestions, based on the LLM. The platform can be used to denote syntax, logic, security, warning, and memory risks.
- Automated Error Detection and Correction: Implemented a system that tracks the number of errors identified and the number of errors corrected with average time to correct an error and number of tokens used. This allows an analysis of performance, comparing across languages and trends over time with the efficiency with which the current system can handle errors.
- Support for Multi-Model Optimization: Developed the system to support multiple LLM models (e.g., Groq, LLaMA) to handle queries, optimize the speed, accuracy, and the count of tokens consumed by code evaluation, and precondition the adaptive selection of models based on task complexity or language.

1.1. Study Layout and Sequence

The layout of the paper is as follows: Section II introduces the related literature, Section III discusses the methodology and the technology stack used, Section IV examines the final results and implementations, and Section V covers the conclusion and future work of the research.

2. Literature Review

The literature review illustrates progress in LLM-based code assistance, agentic debugging and automated review tools, which provide better performance, ethical assessment and educational assistance, while the study identifies three main challenges: system expansion, practical testing and system reliability.

Xiang et al. (2026) To reduce this issue, developed the Agent-centric Debugging Interface (ADI) system, which serves as a new debugging tool that enables cost-effective, complete autonomous system debugging. The SWE-bench benchmark test results provide proof of ADI's operational performance in both effectiveness and efficiency. By simply equipping a basic agent with ADI, it successfully resolves 63.8\% of the tasks on the SWE-bench Verified set, even slightly outperforming the highly optimized and high-investment Claude-Tools agent, at an average cost of USD 1.28 per task with Claude-Sonnet-3.7. Furthermore, demonstrate ADI's generality by integrating it as a plug-and-play component into existing SOTA agents, delivering consistent gains ranging from 6.2\% to 18.5\% on the resolved tasks[29].

Existing work on LLM-powered coding assistance, as focused by Mohammed et al. (2025), uses Large Language Models (LLMs) towards addressing both these concerns. The show how to harness LLM capabilities to do complex code reasoning as well as rewriting of large codebases. also present a novel framework for whole-program transformations that leverages lightweight static analysis to break the transformation into smaller steps that can be carried out effectively by an LLM. The implement ideas in a tool called MSA that targets the CheckedC dialect. The study tests MSA performance through both micro-benchmarks and actual code testing which extends to 20K lines of code. The results show better performance than both the basic LLM model and the current top non-LLM symbolic method[30].

Acharya, Kuppan and Divya's (2025) paper presents the ethical questions posed by Agentic AI and provides remedies to the scarcity of resources, environmental adaptation, and goal alignment. They offer a guideline on how to incorporate Agentic AI into society safely and effectively, noting that more ethical research is needed to ensure that social impacts of AI are beneficial. This survey gives an in-depth overview of the Agentic AI and could help academicians, developers, and lawmakers to use its transformational potential in an ethical and creative way[31].

Sheng Xuan and Lee (2024) presents a Code Review and Bug Detection Tool, which is an automated system, and is used to simplify the detection and correction of code errors. The tool has a graphical user interface (GUI) that is user-friendly to both an advanced and a novice programmer. It uses powerful machine learning models to categorize errors and propose code improvements, which guarantees effective and efficient error detection. The development cycle entailed combining machine learning algorithms in error classification in a simplified GUI. The feedback received by 21 participants evaluated the usability and effectiveness of the tool in its ability to make programming more accessible and streamline the review process. Its successful implementation shows that the tool has the potential to offer a user-friendly solution, which contributes to more efficient programming practices and makes code readable to a wider audience[32].

Pornphol and Chittayasothorn (2024) Learning database programming such as SQL programming is a challenging task

when the queries become more complex. SQL is a declarative language based on relational calculus which describes the definition of the query results instead of describing the procedure or steps used to obtain the query result. Tutorial sessions using tutorial assistance are generally required to support the learning of the advanced part of the language. Recently, generative AI systems demonstrated question answering capabilities, including programming code generation. This paper verifies the SQL code-generating capabilities of four generative AI systems: Bing, Bard, ChatGPT, and Copilot and their suitability as SQL programming assistants [33].

Jayasuriya and Thawalampola (2023) research study encapsulates a modern programming assistance tool developed to solve the challenges that students and instructors experience during fundamental programming lessons. Students frequently experience numerous kinds of coding issues during time-constrained lab environments, with limited opportunities for direct communication with instructors. CodeCoach makes use of the features of GPT-3.5 and provides straightforward explanations of errors identified, provides observations on students' programming approaches, and provides practical suggestions for code development. One of the unique features of CodeCoach is its extensive assistance for numerous programming languages, which ensures its flexibility across various learning environments [34].

Malysheva and Kelleher (2022) examine how TAs' capacity to identify, repair, and handle errors in student code is affected by the availability of corrected code. They discovered that TAs can debug code 29% faster and provide more thorough and accurate explanations of the faults for students (30% more likely to accurately address a specific bug) when they saw a fixed version of the student code. The researchers also discovered that TAs do not typically struggle with conceptual grasp of the underlying content. Rather, their challenges appear to be tied to deficits with working memory, attention, and general high cognitive load [35].

Below is a summary of studies involving an Autonomous Agentic model for AI-enabled error detection and code fixing, which is illustrated in Table I.

Table 1: Large Language Model–Based Code Guidance and Error Resolution

Reference	Methodology	Results	Key contributions	Limitation	Future work
Xiang et al. (2026)	Agent-based debugging with structured execution tracing	Improved task resolution and cost efficiency	Enhances autonomous debugging through structured interaction	Limited real-world validation; benchmark-focused	Apply to real-world systems; improve scalability and robustness
Mohammed et al. (2025)	Developed MSA tool using LLMs with lightweight static analysis for program transformation; evaluated on micro-benchmarks and 20K LOC real-world code	Outperformed vanilla LLM baselines and symbolic methods in code transformation	Demonstrated LLMs' capability for large-scale code rewriting and reasoning	Limited to CheckedC dialect; scalability to other languages untested	Extend framework to broader programming languages and larger industrial codebases
Acharya, Kuppan & Divya (2025)	Conceptual framework addressing ethical issues of Agentic AI (goal alignment, adaptability, resource use)	Provides structured guidelines for safe and ethical AI integration	Raised awareness of ethical considerations for agentic AI deployment	Lacks empirical validation; theoretical in nature	Conduct practical case studies, develop measurable evaluation frameworks
Sheng Xuan & Lee (2024)	Designed ML-based automated code review & bug detection tool with GUI; tested usability with 21 participants	Tool successfully identified & classified errors; user-friendly interface enhanced accessibility	Improved code review process for novices; GUI simplified debugging	Limited sample size; evaluation not on large industrial projects	Test tool on large-scale projects; expand error-classification models
Pornphol & Chittayasothorn, (2024)	Evaluated SQL generation accuracy of Bing, Bard, ChatGPT, and Copilot for SQL learning	Showed varying degrees of accuracy across tools; some useful for SQL tutoring.	Validated generative AI potential as SQL learning assistants	Accuracy inconsistency; reliability depends on tool	Enhance SQL-focused LLM models; longitudinal studies in

					classroom environment
Jayasuriya & Thawalampola (2023)	Developed CodeCoach using GPT-3.5 for error explanation & multi-language programming support in educational labs	Provided accurate error explanations & practical coding suggestions	Enhanced student learning by offering scalable, flexible programming assistance	Focused on beginners; advanced programming not covered	Extend CodeCoach to advanced programming and larger student groups
Malysheva & Kelleher (2022)	Experimental study with TAs using corrected vs. uncorrected student code	TAs debugged 29% faster and gave 30% more accurate explanations with corrected code	The corrected code improves debugging speed and quality	Limited to TA-student interaction; not AI-driven	Apply findings to AI-driven teaching assistants; test scalability across disciplines

3. Methodology

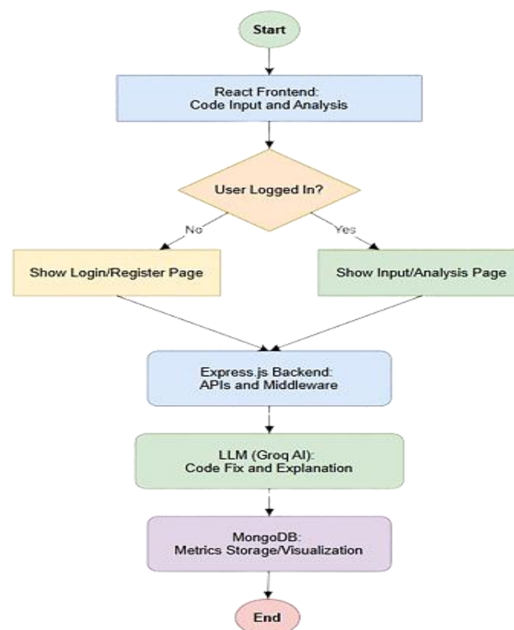

Fig 1: Data Flow Chart of LLM-Powered Code Assistance System

The proposed LLM-mediated agentic coding assistance system utilizing the MERN stack displayed in Figure 1. The process begins at the React front end where users can login or register with the provided secure authentication module. Once the user is authenticated, they can access the code input and assessment interface where they can enter programming problems or input code snippets. The user-provided input is then forwarded to the Express.js backend, where all API calls and middlewares are handled and directly sent to the integrated LLCM engine. The model breaks the block of code and identifies any errors in the code, and gives recommendations on how the code can be fixed and reasons why the code should be implemented in real time. Meanwhile several metrics are gathered and stored in MongoDB to track, visualize, and analyze them. The flow enables the user to interact with the AI-based structure with ease whilst providing both the low-latency debug support and substantive usability through data-driven insights regarding the code quality and software performance analyses.

3.1. Tools and Technologies Used

The system Architecture and Technologies that were used to develop LLM powered code assistance system is shown below:

- MERN Stack: The system is developed with the MERN stack which provides a complete JavaScript environment, or full-stack JavaScript. React.js powers the frontend for an interactive code submission and metrics visualization, Express.js with Node.js handles the back end and API requests, while MongoDB serves to store and manage user submissions, error logs and historical performance metrics [36].
- Authentication & Security: JSON Web Tokens (JWT) are utilized for secure user login and session management. This guarantees that only authorized users are able to submit code and retrieve stored results. JWT provides a trust mechanism for: authentication, data integrity and secure communication between the front- and back-end components [37][38].

- Large Language Model (LLM): The system distributes LLM API code analysis across various languages and identifies syntax errors, logic errors, severe errors for security risks, and detects possible optimizations to fix. The LLM generates LLM APIs to flag the errors and fix as necessary. Performance is tracked through errors detected/fixed, tokens used, and average time to fix [39].
- Groq AI Integration: The framework is optimized for performance and supports Groq AI, which is a high-throughput method to get low-latency inference for LLM queries. This means faster error detection and faster error correction to the point where the system is responsive enough to provide real-time interactive coding assistance [40].

3.2. System Design and Workflow

The automated error-fixing framework developed by means of the MERN stack. After using a secure JSON Web Token-based login to authenticate, users can submit code snippets in Python, JavaScript, Java, C#, or C++ programming languages. An integrated Groq A.I. LLM to learn about it. This allows for rapid assessment and automated error corrections. The LLM can detect syntax errors, logic errors, and security issues. After applying corrections, the system stores metrics on each code snippet to MongoDB including; lines of code, comment density, cyclomatic complexity, errors detected versus errors fixed, tokens spent, and fix time.

Fig 2: Login Form

This is a standard user login form. The form is shown in Figure 2 with a clean and modern interface featuring a center container. The form has two input fields, one for Email and one for Password, each one having placeholder text to guide the user through the process. Below the input fields is a prominent and brightly colored blue Login button. Below the button is a link to a registration page for those users who do not already possess an account. The overall design utilizes minimal design features and emphasizes user interaction in a simple manner.

Fig 3: Register Form

Here is a user registration form. Figure 3 shows a new user registration form with a clean, simple layout that is easy for the new user to create an account. The user must complete three pieces of information consisting of a Username, Email and a Password. Each input field contains a placeholder for the required information. There is a visible green Register button underneath the fields. At the bottom there is a link for existing users who already have an account to log in.

Fig 4: Code Assistance

This is the primary interface of the LLM Coding Assistant. Figure 4 presents a clean, user-friendly layout of the "Code Analyzer" tool. The interface features a text area for the user to paste good code, which already has a sample JavaScript code snippet populated for viewing purposes. Below the code editor, there is a drop-down menu to select the programming language. There is also a blue button to "Analyze Code," which is used to begin the analysis.

Fig 5: Analysis Results

This page illustrates the "Fixed Code" view from the LLM Coding Assistant. In Figure 5, can see the outcome of the

automatic assessment and fix of the input code. The top portion of the view shows the fixed code in JavaScript, which simply improved the for-loop code with a better performance and memory-utilized Array.prototype.map() method. The bottom half, provides the detailed "explanation", which is in markdown, for why the code was inefficient and how the new code allowed for improved performance and memory management.

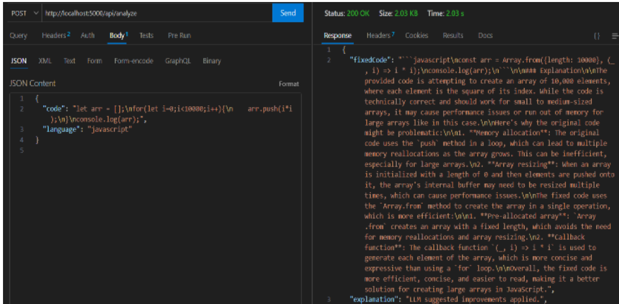

Fig 6: Thunder Client API Testing

This is the Thunder Client interface, a popular extension made to run API tests. In Figure 6, the user, shown on the left side of the screen, makes a POST request to an API endpoint (/api/analyze) to analyze a piece of code, with the request (JSON) body containing the language and the code to analyze. The Response panel is displayed on the right side of the screen, showing a server response that contains the fixed code and a detailed explanation of the fix made. This is a test environment to make sure the functionality of the API that performs code analysis works as expected.

4. Results and Discussion

This section summarizes the main metrics and outputs of the LLM-powered automated coding assistance ecosystem. It was developed on an HP Laptop, with an AMD Ryzen 9 processor, 32GB DDR5 RAM, 1TB NVMe SSD, Windows 11 and a fast internet connection. The framework was developed on the MERN stack, featuring client-side (React), server-side (Node.js, Express, and MongoDB) using user registration and login, comfort to enter and send multiple programming language code snippets through the Groq AI LLM which is capable of taking the submitted code and performing multi hold error detection and automated fixing on the code snippet. The metrics are logged in a MongoDB database. All of these metrics were utilized and visualized in React using Chart.js. This visualization helps to see and compare per-language performance, system performance, and overall efficiency and code improvement.

4.1. Results and Visualization

The key conclusions and interpretations drawn from the system gathered are summarized below:

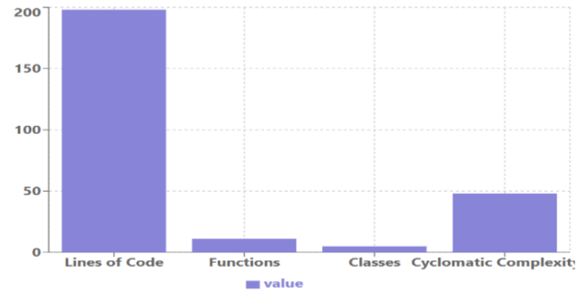

Fig 7: Code Structure Bar Char

This bar chart gives an overview of code structure metrics. In Figure 7 "Overview of Code Structure Metrics" can see that the Lines of Code metric is the single highest value, close to 200. The counts for Functions and Classes are very low, both below 10. The Cyclomatic Complexity is moderate with an almost 50. This implies that although these codebases may contain many lines, they are not complex, with only a very small number of functions or classes.

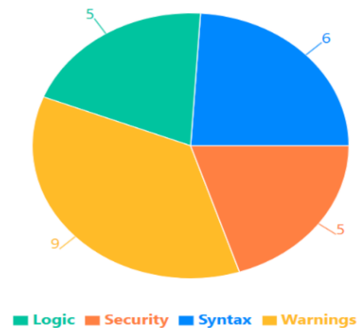

Fig 8: Distribution Pie Chart

This pie chart gives the details of the breakdown of the various kinds of code issues found. Figure 8, which is called Duration of Code Issues, demonstrates that the most common issue present is warnings, which make up the majority of the total, having a count of 9. The common problems are also syntax and security issues, to which the counts of 6 and 5, respectively, are attributed. Logic issues are the least of the problem areas, with a count of 5. This pie chart gives a sector of what type of errors are being incurred with the code that has been analyzed.

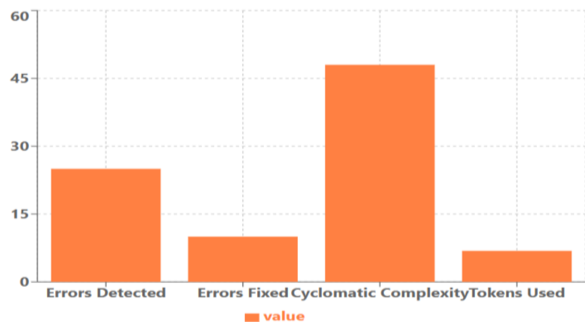

Fig 9: Performance Bar Chart

The bar chart is a summary of some of the key performance indicators of the LLM-powered coding assistant. Figure 9, Overall System Performance Metrics, illustrates that

the system detected 20+ errors, but corrected fewer than 10. The Cyclomatic Complexity of the code is relatively high, i.e. 48. The TokensUsed is about 8, which is low. The coding assistant demonstrates that it is capable of identifying problems successfully, but the success of corrections identified by the coding assistant varies in how successful they are at getting automatically fixed.

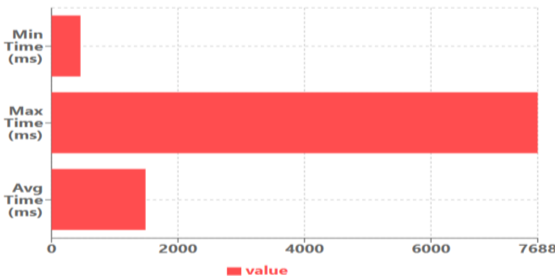

Fig 10: Code Fix Time Statistics

This bar chart shows minimum, maximum, and average fix times in milliseconds. Figure 10, called "Code Fix Time Statistics," indicates that there is considerable variability in performance. The Minimum Time is fairly low, just over 400ms, and the Maximum Time is quite high, about 7688ms. The Average Time is about 1300ms, which indicates while there are fixes that take short amounts of time, a very small number of fixes that take much longer time to process have a significant impact on overall performance.

The chart includes the performance of the system over time. In the graph shown in Figure 11 titled "Performance Over Time" both errors detected and errors fixed are consistently low and constant throughout the time the system is tested (the x-axis). Fix time plays out very differently from errors detected and fixed. The line represents fix time is very high; then drops before becoming much more variable. Since it's clear that detection and fixing are well behaved, but the time it takes to fix each error, even within the same code snippet, can vary greatly from one to the next, and this is what is causing the variability in fix time.

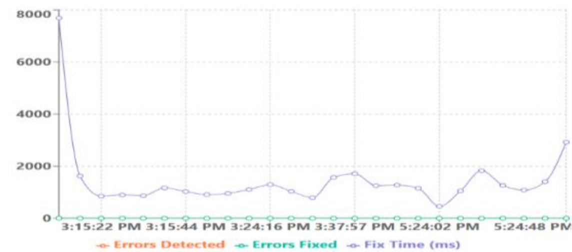

Fig 11: Performance Over Time

Table 2: System Performance Metrics Across Programming Languages

Language	Avg. LOC per Submission	Errors Detected	Errors Fixed	Success Rate (%)	Avg. Response Time (s)	Cyclomatic Complexity (avg)
Python	85	12	11	91.6	1.8	6.2
JavaScript	92	15	13	86.7	2.1	7.5
Java	110	18	15	83.3	2.4	8.1
C#	95	14	12	85.7	2.0	7.2
C++	130	20	16	80.0	2.7	9.4

Table II presents the performance of the proposed Groq AI–powered MERN framework across five programming languages. The highest fix success rate (91.6) and low average response time (1.8s) showed that Python submissions are the most efficient in dealing with dynamic scripting code. On the other hand, C++ had a higher cyclomatic complexity and a higher response time (2.7s) which led to a relatively lower success rate (80%). Java and Csh were associated with moderate complexity, fixed performance, and moderate results. These results show how the framework can identify and fix mistakes on a regular schedule and also show how the complexity of the language affects system performance.

4.2. Discussion and Evaluation

The framework suggested is an effective approach to automated code support and error detection via a MERN stack that has been composed with Groq AI. The users are identified by JWT-secured authentication and post code snippets in various languages, which are analyzed in syntax, logic and security concerns and then automated fixes are implemented. Complex metrics e.g. lines of code, comment density, cyclomatic complexity, errors found vs fixed, number of tokens and fix time are made accessible in MongoDB. The React frontend displays these metrics in interactive charts, which make it possible to monitor metrics in real-time. This design is scalable, provides correct performance tracking, and meaningful analytics to the researchers and developers.

4.3. Limitations and Future Work

The existing structure has shortcomings in the form of a lack of consistency in fix time - between a few hundred milliseconds and a few seconds - and successful correction of complex fixes, especially with complex logic bugs and high-complexity languages, like C++. Moreover, the system is confined to a few programming languages and relies on a set of pretrained LLM models, which can limit flexibility and scalability. The new direction of work will be to increase accuracy and stability with more sophisticated methods like RLHF, prompt engineering, and hybrid reasoning and also provide language support and integrate tools like Docker, Kubernetes, CI/CD pipelines, IDE extensions, and cloud-based collaborative debugging to make the work more scalable and closer to practice.

5. Conclusion

The suggested agentic architecture based on LLM illustrates how far-off interactive coding assistance, and error

correction can be integrated into a modular MERN stack framework. The system supports the use of Groq and LLaMA to optimize multi-models, which significantly lowers the inference latency to generate syntax, security, logic and warnings across multiple programming languages in real-time. The framework, with secure authentication, structured storage of submissions in MongoDB, and visualization dashboards built with React, not only includes direct support in debugging but also rich analytics of the code structure, cyclomatic complexity, and token usage, fix success rates, and response times. Experimental assessments verify that it has a high level of error detection, with Python having the best success rate (91.6%), and C++ having lower rectifying success (80%), which demonstrates the difficulty of a language and its influence on automated correction. Detection was also steady, although fix times were highly varied, between 400ms and more than 7s, indicating the difficulty of constant real-time correction. In general, the framework not only boosts productivity by decreasing the effort required in debugging but also prepares the basis of future developments in adaptive LLM-based coding support, leading to more reliable, efficient, and intelligent software development environments.

References

[1] G. Modalavalasa, “The Role of DevOps in Streamlining Software Delivery : Key Practices for Seamless CI / CD,” *Int. J. Adv. Res. Sci. Commun. Technol.*, vol. 1, no. 2, 2021, doi: 10.48175/IJARSCT-8978C.

[2] V. Prajapati, “Advances in Software Development Life Cycle Models: Trends and Innovations for Modern Applications,” *JGREC*, vol. 1, no. 4, 2025, doi: 10.5281/zenodo. 15222129.

[3] S. Pandya, “Predictive Modeling for Cancer Detection Based on Machine Learning Algorithms and AI in the Healthcare Sector,” *TIJER – Int. Res. J.*, vol. 11, no. 12, 2024.

[4] J. Mishra, S. Rai, B. Moharana, V. K. Singh, S. Dey, and T. Sarkar, “Revolutionizing Financial Fraud Detection in Healthcare Insurance Through Blockchain & AI-Driven Predictive Analysis,” in *2025 International Conference on Technology Enabled Economic Changes (InTech)*, IEEE, Feb. 2025, pp. 1516–1522. doi: 10.1109/InTech64186.2025.11198190.

[5] H. Kali, “Optimizing Credit Card Fraud Transactions identification and classification in banking industry Using Machine Learning Algorithms,” *Int. J. Recent Technol. Sci. Manag.*, vol. 9, no. 11, pp. 85–96, 2024.

[6] V. Verma, “Security Compliance and Risk Management in AI-Driven Financial Transactions,” *Int. J. Eng. Sci. Math.*, vol. 12, no. 7, July, pp. 107–121, 2023.

[7] S. Shivam, V. Nutalapati, T. P. Patel, A. K. Padhy, M. Kumari, and R. Purushothaman, “Transformer-Based Framework for Imbalanced Transaction Fraud Detection in FinTech Systems,” in *2026 14th International Symposium on Digital Forensics and Security (ISDFS)*, IEEE, Mar. 2026, pp. 1–6. doi: 10.1109/ISDFS69419.2026.11459102.

[8] U. Korat and A. Alimohammad, “Efficient Hardware Implementation of Eigen Solver,” in *2026 IEEE 16th Annual Computing and Communication Workshop and Conference (CCWC)*, Las Vegas, NV, USA: IEEE, 2026, pp. 1376–1385, January. doi: 10.1109/CCWC67433.2026.11393888.

[9] P. G. Todorov, “The Application of Artificial Intelligence in Software Engineering,” *Int. J. Adv. Multidiscip. Res. Stud.*, vol. 2, no. 5, pp. 835–842, 2022, doi: 10.2139/ssrn.4943407.

[10] V. Verma, “Deep Learning-Based Fraud Detection in Financial Transactions : A Case Study Using Real-Time Data Streams,” vol. 3, no. 4, pp. 149–157, 2023, doi: 10.56472/25832646/JETA-V3I8P117.

[11] S. Dodda, H. Volikatla, and J. R. Vummadi, “Exploring the Role of AI-Enhanced Chatbots in Automating Recruitment Processes in Human Capital Management Systems,” *Int. J. Emerg. Trends Comput. Sci. Inf. Technol.*, vol. 6, no. 3, pp. 28–36, 2025, doi: 10.63282/3050-9246.IJETCSIT-V6I3P104.

[12] G. Maddali, “Efficient Machine Learning Approach Based Bug Prediction for Enhancing Reliability of Software and Estimation,” *SSRN Electron. J.*, vol. 8, no. 6, 2025, doi: 10.2139/ssrn. 5367652.

[13] H. W. Marar, “Advancements in software engineering using AI,” *Comput. Softw. Media Appl.*, vol. 6, no. 1, p. 3906, 2024, doi: 10.24294/csma.v6i1.3906.

[14] S. R. P. Madugula and N. Malali, “AI-powered life insurance claims adjudication using LLMs and RAG Architectures,” *Int. J. Sci. Res. Arch.*, no. April, 2025, doi: 10.30574/ijsra.2025.15.1.0867.

[15] V. B. S. Tarakampet, R. R. Koilakonda, and S. Tatavarthi, “Domain Aware Prompt Engineering,” *World J. Adv. Eng. Technol. Sci.*, vol. 16, no. 01, pp. 569–574, July, 2025, doi: https://doi.org/10.30574/wjaets.2025.16.1.1249.

[16] D. Patel, “AI-Enhanced Natural Language Processing for Improving Web Page Classification Accuracy,” vol. 4, no. 1, pp. 133–140, 2024, doi: 10.56472/25832646/JETA-V4I1P119.

[17] S. Gajula, “Cloud Transformation in Financial Services: A Strategic Framework for Hybrid Adoption and Business Continuity,” *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 11, no. 2, pp. 1244–1254, Mar. 2025, doi: 10.32628/CSEIT25112464.

[18] B. P. Singh and H. Singh, “Using LLMs for Autonomous Cloud Infrastructure Entitlement Management to Prevent Overprivileged Access,” *J. Eng. Comput. Sci.*, vol. 5, no. 4, pp. 1–14, April, 2026, doi: https://doi.org/10.5281/zenodo.19488212.

[19] B. Krishnan, S. Perla, S. Maddela, and R. Lingam, “Adaptive Multi-Cloud Infrastructure for CRM Analytics: Real-Time ML and Data Sync with LLMs,” in *2025 IEEE 3rd Global Conference on Wireless Computing and Networking (GCWCN)*, IEEE, Nov. 2025, pp. 1–8. doi: 10.1109/GCWCN66157.2025.11448404.

[20] S. Pandya, “Comparative Analysis of Large Language Models and Traditional Methods for Sentiment Analysis of Tweets Dataset,” *Int. J. Innov. Sci. Res. Technol.*, vol. 9, no. 12, 2024, doi: 10.5281/zenodo.14575886.

[21] H. P. Kapadia and K. B. Thakkar, “Generative AI for Real-Time Customer Support Content Creation,” *J.*

Emerg. Technol. Innov. Res., vol. 10, no. 12, pp. 36–43, 2023.

[22] H. P. Kapadia, “Generative AI for Real-Time Conversational Agents,” *Int. J. Curr. Sci.*, vol. 13, no. 3, pp. 201–208, 2023.

[23] N. K. R. Choppa and N. Kolli, “Contextual Frameworks for Agentic AI: Engineering Adaptive Memory and Retrieval Mechanisms,” *Comput. Fraud Secur.*, vol. 2024, no. 11, pp. 395–406, 2024, doi: https://doi.org/10.52710/cfs.747.

[24] H. Derouiche, Z. Brahmi, and H. Mazeni, “Agentic AI Frameworks: Architectures, Protocols, and Design Challenges,” 2025.

[25] S. Rongala, S. A. Pahune, H. Velu, and S. Mathur, “Leveraging Natural Language Processing and Machine Learning for Consumer Insights from Amazon Product Reviews,” in *2025 3rd International Conference on Smart Systems for Applications in Electrical Sciences (ICSSES)*, 2025, pp. 1–6. doi: 10.1109/ICSSES64899.2025.11009528.

[26] S. Thoom, “Understanding Agentic Frameworks in AI Development: A Technical Analysis,” *Int. J. Sci. Res. Comput. Sci. Eng. Inf. Technol.*, vol. 11, pp. 518–527, 2025, doi: 10.32628/CSEIT25111249.

[27] M. Anitha, S. Hazarabi, N. Kunala, and P. Deekshitha, “Code Generation and Automated Bug Fixing,” *IARJSET*, vol. 11, 2024, doi: 10.17148/IARJSET.2024.11414.

[28] M. R. C. MUKKOLAKKAL, “IntelliStore: An Intelligent AI Agent Framework for Autonomous Storage and Database Optimization in Cloud-Native Microservices,” *Int. J. Sci. Res. Mod. Technol.*, vol. 3, no. 12, pp. 243–250, Dec, 2024, doi: https://doi.org/10.38124/ijsrmt.v3i12.1024.

[29] J. Xiang, X. Xu, X. Chu, H. Tian, and Y. Zhang, “Empowering Autonomous Debugging Agents with Efficient Dynamic Analysis,” 2026. doi: https://doi.org/10.48550/arXiv.2604.24212.

[30] N. Mohammed, A. Lal, A. Rastogi, R. Sharma, and S. Roy, “LLM Assistance for Memory Safety,” in *2025 IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, 2025, pp. 1717–1728. doi: 10.1109/ICSE55347.2025.00023.

[31] D. B. Acharya, K. Kuppan, and B. Divya, “Agentic AI: Autonomous Intelligence for Complex Goals A Comprehensive Survey,” *IEEE Access*, vol. 13, pp. 18912–18936, 2025, doi: 10.1109/ACCESS.2025.3532853.

[32] P. J. Sheng Xuan and Y. Lee, “Automated Code Review and Bug Detection,” in *2024 19th International Joint Symposium on Artificial Intelligence and Natural Language Processing (iSAI-NLP)*, 2024, pp. 1–6. doi: 10.1109/iSAI-NLP64410.2024.10799479.

[33] P. Pornphol and S. Chittayasothorn, “Using LLM Artificial Intelligence Systems as Complex SQL Programming Assistants,” in *2024 12th International Conference on Information and Education Technology (ICIET)*, 2024, pp. 477–481. doi: 10.1109/ICIET60671.2024.10542806.

[34] D. D. D. H. Jayasuriya and O. Thawalampola, “CodeCoach : An Interactive Programming Assistance Tool,” no. December, 2023, doi: 10.52783/tjjpt.v44.i4.2562.

[35] Y. Malysheva and C. Kelleher, “Assisting Teaching Assistants with Automatic Code Corrections,” in *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, in CHI ’22. New York, NY, USA: Association for Computing Machinery, 2022. doi: 10.1145/3491102.3501820.

[36] M. Menghnani, “Modern Full Stack Development Practices for Scalable and Maintainable Cloud-Native Applications,” vol. 10, no. 2, 2025, doi: 10.5281/zenodo. 14959407.

[37] S. P. Bheri and G. Modalavalasa, “Advancements in Cloud Computing for Scalable Web Development: Security Challenges and Performance Optimization,” *J. Comput. Technol. Int. J.*, vol. 13, no. 12, 2024.

[38] V. S. Thokala, “A Comparative Study of Data Integrity and Redundancy in Distributed Databases for Web Applications,” *Int. J. Res. Anal. Rev.*, vol. 8, no. 04, pp. 383–390, 2021.

[39] S. Pahune and M. Chandrasekharan, “Several Categories of Large Language Models (LLMs): A Short Survey,” *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 11, no. 7, pp. 615–633, 2023, doi: 10.22214/ijraset. 2023.54677.

[40] S. Pahune and Z. Akhtar, “Transitioning from MLOps to LLMOps: Navigating the Unique Challenges of Large Language Models,” *Information*, vol. 16, no. 2, p. 87, Jan. 2025, doi: 10.3390/info16020087.