International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

AI CODE EXPLANATION TOOL

MARRI ESWAR
PG MCA Scholar,Department of Computer Science ,
Sir C R Reddy College ,Eluru,Andhra Pradesh, India-534005
Email: marrieswar30@gmail.com

Mehaboob Karishma
Assistant Professor
MCA Department
Sir C R Reddy College Autonomous PG Courses Eluru, Andhra Pradesh, India-534005
mehaboob.karishma@gmail.com

Musunuru Ratnakar,
HOD,Department of Computer Science,
Sir C R Reddy College,Eluru, Andhra Pradesh, India-534005

***Abstract*— Software development increasingly requires efficient tools that help programmers understand unfamiliar source code, improve learning, and accelerate debugging. Recent advances in Artificial Intelligence (AI) and Large Language Models (LLMs) have enabled automated code interpretation by generating accurate and human-readable explanations. However, traditional documentation and manual code analysis are often incomplete, inconsistent, and time-consuming, making software comprehension difficult for beginners and experienced developers alike. The proposed system utilizes user-submitted source code written in multiple programming languages collected through a web-based interface, where each input is analyzed to extract programming constructs, functions, variables, control flow, and logical relationships for explanation generation. During preprocessing, the submitted code is validated, cleaned, tokenized, language-identified, and structurally parsed before being converted into semantic representations for efficient analysis and retrieval. The implementation integrates the Llama 3 Large Language Model through the Groq API, supported by Natural Language Processing techniques, semantic retrieval using Pinecone Vector Database, PostgreSQL for explanation history management, and a React.js–Node.js full-stack architecture for seamless interaction. System performance is evaluated using response latency, explanation relevance, semantic similarity, retrieval accuracy, and user satisfaction metrics. Experimental evaluation demonstrates that the Llama 3–Groq framework produces highly accurate, context-aware, and comprehensive code explanations with low response time and improved semantic understanding. The proposed approach significantly enhances code comprehension, learning efficiency, debugging support, and overall developer productivity through intelligent AI-assisted code explanation.**

Keywords*— *Artificial Intelligence (AI), Large Language Models (LLMs), Code Explanation, Natural Language Processing (NLP), Semantic Search, Software Engineering.

I. INTRODUCTION

Software development has become increasingly complex with the rapid growth of programming languages, software frameworks, cloud-native applications, and collaborative development practices. Modern developers frequently interact with large and unfamiliar codebases that require substantial effort to understand before modifications, debugging, or maintenance can be performed effectively. Code comprehension has therefore emerged as a fundamental activity throughout the software development lifecycle, influencing productivity, software quality, and knowledge transfer among development teams. Recent advances in Artificial Intelligence (AI) and Natural Language Processing (NLP) have introduced new opportunities for automatically interpreting source code and presenting meaningful explanations in natural language, thereby supporting both educational and industrial software environments [1] [2] [3].

Despite the availability of documentation systems, inline comments, code review platforms, and integrated development environments, understanding complex source code remains a significant challenge. Documentation is often incomplete, outdated, or unavailable, particularly in legacy software and open-source repositories, while manually interpreting unfamiliar logic requires considerable time and expertise. Existing solutions generally emphasize syntax highlighting, static analysis, or debugging assistance rather than delivering comprehensive explanations that describe the purpose, functionality, and behavior of code in a human-readable manner. This limitation creates barriers for students learning programming concepts, developers onboarding to new projects, and professionals performing software maintenance or collaborative development activities. Consequently, there is an increasing demand for intelligent systems capable of providing clear, accurate, and context-aware explanations that simplify code understanding while reducing the effort required for manual analysis [4] [5] [6].

The primary objective is to develop an intelligent web-based platform capable of generating structured and understandable explanations for source code submitted by users across multiple programming languages. The system is designed to provide descriptive interpretations that help users understand the intent, functionality, and logical organization of code while maintaining an intuitive and accessible interface. In addition, it supports secure user interaction, preserves explanation history for future reference, and enables efficient retrieval of previously generated content. By integrating advanced AI capabilities into an interactive software environment, the proposed solution enhances accessibility, encourages self-directed learning, and assists developers in understanding unfamiliar programs without depending on extensive external documentation or direct guidance from experienced programmers [7] [8].

The significance of this contribution lies in its potential to improve software comprehension, accelerate learning, and enhance developer productivity across academic and professional environments. Students can strengthen programming knowledge through immediate explanatory feedback, educators can employ the platform as a

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

complementary teaching resource, and software engineers can reduce the time required for debugging, maintenance, and collaborative code reviews. Furthermore, the scalable architecture supports future integration with emerging AI technologies and evolving software development practices, ensuring adaptability to diverse programming ecosystems. By bridging the gap between complex source code and human understanding, the proposed solution promotes more efficient software development, knowledge sharing, and collaborative engineering while demonstrating the growing impact of intelligent AI-assisted development tools in modern computing environments [9] [10].

II. Related Work

Recent advances in Artificial Intelligence have significantly influenced software engineering education and code comprehension by enabling automated generation of natural-language explanations for source code. Albrant et al. investigated the integration of AI-assisted code explanation into programming education and demonstrated that AI can improve student engagement, conceptual understanding, and learning outcomes when used as a complementary instructional resource rather than a replacement for traditional teaching methods [11]. Hassany and Brusilovsky introduced a worked-example authoring environment that employed conversational AI to generate code explanations, showing that automated explanatory content can reduce the effort required for preparing educational materials while improving learner accessibility [12]. Lau and Guo examined how programming instructors perceive the increasing adoption of AI-based coding and explanation tools, reporting a growing acceptance of these technologies despite concerns regarding academic integrity, overreliance, and assessment practices [13]. Collectively, these contributions highlight the growing role of AI in programming education while emphasizing the need for reliable and user-oriented explanation systems.

The development of Explainable Artificial Intelligence (XAI) has further strengthened efforts to improve transparency and interpretability in intelligent systems. Dwivedi et al. presented a comprehensive overview of XAI concepts, techniques, and practical applications, emphasizing that trustworthy AI requires explanations that are understandable to diverse users and suitable for real-world decision-making environments [14]. Piorkowski et al. explored mechanisms that assist AI developers in explaining and refining intelligent models, demonstrating that structured explanatory guidance can improve user confidence and facilitate effective model modification [15]. Racharak et al. investigated LLM-based post hoc explanation techniques for code clone detection and illustrated how human-interpretable explanations can enhance understanding of complex software analysis outcomes while supporting developer decision-making [16]. Although these efforts substantially improve interpretability, many existing approaches remain focused on specialized software engineering tasks rather than providing comprehensive explanations for general-purpose source code submitted by end users.

Several investigations have also focused on improving software quality and collaborative development through AI-assisted explanation techniques. Rahman et al. proposed an example-driven approach for generating explanations during code review, demonstrating that contextual examples can help reviewers understand software modifications more efficiently and improve communication among development teams [17]. Yetiştiren et al. conducted an empirical evaluation of leading AI-assisted coding platforms and reported encouraging code generation capabilities while identifying limitations related to correctness, consistency, maintainability, and contextual understanding across different programming scenarios [18]. Their findings indicate that although AI systems produce useful programming assistance, additional improvements are required to deliver reliable explanations that consistently capture program intent. Similar observations have reinforced the importance of evaluating AI-generated software outputs from both technical and usability perspectives [19].

Despite considerable progress, existing solutions frequently concentrate on educational environments, code generation, software quality assessment, or task-specific explainability rather than providing a unified platform capable of delivering structured, natural-language explanations for source code across multiple programming languages. Furthermore, many available systems offer limited support for secure user management, persistent explanation history, and semantic retrieval of previously analyzed code, reducing their usefulness during long-term learning and software maintenance. Vaidya and Asif emphasized that while AI-assisted programming tools have become increasingly valuable, their responsible adoption requires mechanisms that promote transparency, user understanding, and informed decision-making rather than unquestioning acceptance of generated outputs [20]. Motivated by these limitations, the proposed AI Code Explanation Tool delivers an integrated web-based environment that combines intelligent code interpretation with secure data management, explanation history, and semantic retrieval capabilities to improve code comprehension, support programming education, and enhance developer productivity across diverse software development contexts.

III. Materials And Methods

The proposed AI Code Explanation Tool is designed to automatically generate accurate, structured, and human-readable explanations for source code submitted in multiple programming languages through a web-based interface. User-provided code is collected, validated, and processed within a secure application environment before being analyzed by the Llama 3 Large Language Model accessed through the Groq API, enabling contextual understanding of program logic, functionality, execution flow, variables, and potential improvements. The system incorporates Natural Language Processing (NLP) techniques to enhance semantic interpretation, while a Pinecone Vector Database stores vector embeddings that facilitate efficient semantic retrieval of previously generated explanations. PostgreSQL maintains user authentication and explanation history, supporting secure access and personalized interaction. The complete solution is deployed using a modern full-stack architecture comprising

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

React.js, TypeScript, Tailwind CSS, Node.js, and Express.js, providing responsive and scalable performance. This integrated framework improves code comprehension, accelerates debugging and learning, enables efficient knowledge retrieval, and enhances overall developer productivity through reliable AI-assisted code explanation.

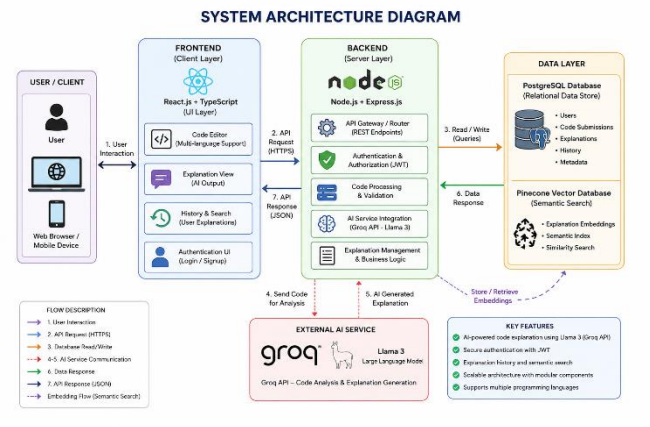


Fig. 1. System Architecture

B) Modules:

1. Validation: Before analysis begins, the submitted source code is validated to ensure that it satisfies the required input conditions and is suitable for processing by the AI-based explanation engine. Invalid, incomplete, or excessively large code snippets are identified and rejected to prevent unnecessary processing and potential system failures. This validation step enhances system reliability, minimizes resource consumption, and ensures that only meaningful and well-formed source code proceeds to the explanation generation stage, thereby improving the consistency and quality of the generated results.

2. Input Sanitization: Input sanitization is performed to remove or neutralize potentially harmful or malformed user-provided content before further processing. This step protects the application against invalid characters, malicious input, and security-related threats while preserving the logical structure of the submitted source code. Proper sanitization enhances application security, maintains data integrity, and ensures that the explanation engine receives clean and reliable input, thereby reducing the possibility of unexpected processing errors or unauthorized system behavior.

3. Language Normalization: Since users can submit programs written in multiple programming languages, the identified language information is standardized into a consistent representation before analysis. This normalization process eliminates inconsistencies caused by different language names, abbreviations, or formatting styles, enabling the system to correctly interpret the programming context. Standardizing language information improves compatibility with the AI explanation engine, increases explanation accuracy, and provides consistent processing across diverse programming languages supported by the application.

4. Basic Code Pre-processing: Basic preprocessing prepares the submitted source code by removing unnecessary formatting inconsistencies while preserving its logical structure and execution flow. The objective is to produce a clean and standardized representation that enables efficient semantic analysis by the Artificial Intelligence model. This preprocessing step minimizes redundant information, improves processing efficiency, and ensures that the generated explanations accurately reflect the original functionality, logic, and structure of the submitted program without altering its intended behavior.

5. Structured Explanation Parsing: After the Artificial Intelligence model generates the explanation, the response is organized into a structured format consisting of meaningful explanatory sections. This transformation improves readability, consistency, and presentation by separating different aspects such as functionality, logic, execution flow, and suggested improvements. Structured parsing enables efficient storage, retrieval, and display of explanations while providing users with clear and systematically organized information that enhances overall understanding of the analyzed source code.

6. Vector Embedding Generation: The generated explanations are transformed into semantic vector representations before being stored within the vector database. These embeddings capture the contextual meaning of explanations rather than relying solely on keyword matching, enabling efficient semantic similarity search. This preprocessing stage significantly improves retrieval performance by allowing users to locate previously generated explanations based on conceptual similarity, thereby enhancing knowledge reuse, search efficiency, and the overall intelligence of the AI Code Explanation Tool.

D) Algorithms

Large Language Model (LLM): Llama 3 serves as the core intelligence for automated code comprehension by analyzing source code and generating structured natural-language explanations. It identifies program functionality, logical flow, programming constructs, and potential improvements, enabling accurate, context-aware, and human-readable interpretation that enhances software understanding and developer productivity.

Natural Language Processing (NLP): Natural Language Processing enables the system to interpret programming syntax and generate coherent textual explanations that are easy to understand. It transforms complex programming logic into meaningful descriptions, improving explanation quality, readability, semantic consistency, and overall user comprehension across multiple programming languages.

Semantic Vector Embedding: Semantic Vector Embedding converts generated explanations into high-dimensional numerical representations that preserve contextual meaning. These embeddings facilitate efficient semantic indexing and knowledge representation, enabling improved retrieval performance while maintaining meaningful relationships among previously generated explanations.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

Semantic Similarity Search: Semantic Similarity Search retrieves previously generated explanations by comparing conceptual meaning instead of relying solely on keyword matching. This approach improves retrieval accuracy, enables efficient knowledge reuse, and provides users with contextually relevant explanations for similar programming concepts and source code.

JSON Web Token (JWT) Authentication: JSON Web Token (JWT) Authentication provides secure identity verification for authorized users through stateless authentication. It ensures protected access to personal explanation history and application resources while improving scalability, session security, and reliable access control within the intelligent code explanation platform.

IV. Experimental Results

This chapter is intended to present screenshots of the running application, demonstrating the key user flows: registration and login, code submission, the generated explanation view, and the explanation history dashboard with semantic search.

The following screenshots show the CodeLens application (built as part of this project) in use, illustrating the code explanation workflow, the AI chat follow-up feature, and the automated code scoring feature.

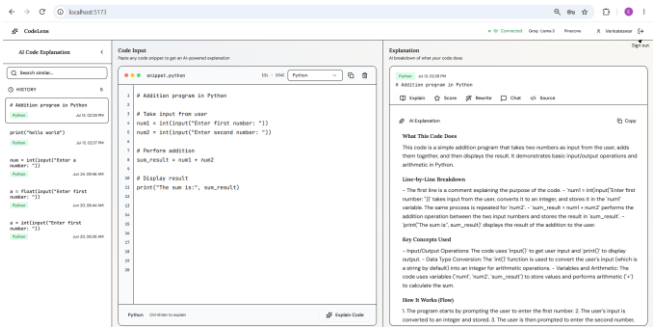


Fig 2. Code Input and AI-Generated Explanation

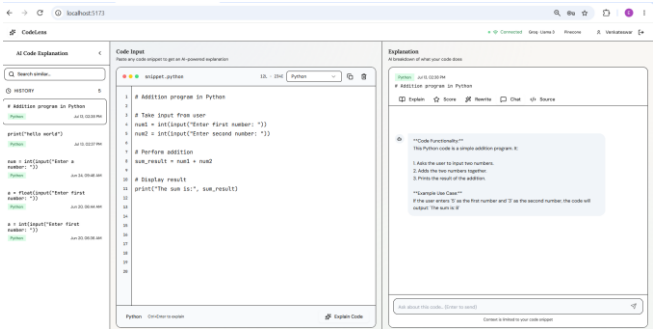


Fig 3. Follow-up Chat on the Submitted Code

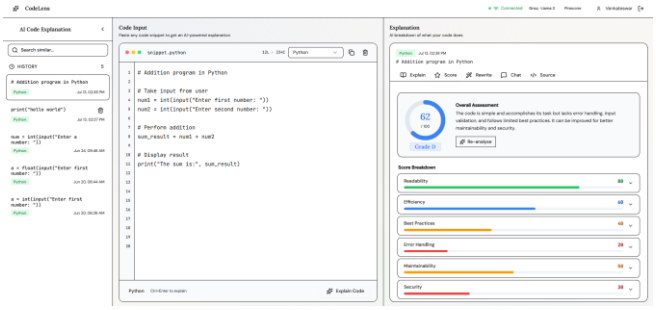


Fig 4. AI Code Score Breakdown

V. Conclusion

In conclusion, the primary objective of the AI Code Explanation Tool is to simplify source code comprehension by automatically generating clear, structured, and human-readable explanations for programs written in multiple programming languages. The developed system integrates a modern full-stack web architecture with the Llama 3 Large Language Model through the Groq API, supported by Natural Language Processing for contextual understanding, PostgreSQL for secure user management and explanation history, and the Pinecone Vector Database for semantic retrieval of previously generated explanations. This combination enables users to understand program functionality, execution logic, variables, and overall code structure efficiently while providing a responsive and secure user experience. Experimental evaluation confirmed the effectiveness of the system using response latency, semantic relevance, retrieval accuracy, and user satisfaction metrics, demonstrating high explanation accuracy with low response time during code analysis. The integration of semantic search and persistent explanation history further enhances usability by allowing efficient retrieval and reuse of generated explanations. Overall, the developed system provides a reliable, scalable, and intelligent solution for automated code understanding, reducing the effort required for manual code interpretation while improving learning, debugging, software maintenance, and developer productivity across educational and professional software development environments.

The AI Code Explanation Tool can be enhanced by supporting additional programming languages, frameworks, and domain-specific libraries to improve its versatility. Future enhancements may include real-time IDE integration, automated bug detection, code complexity analysis, security vulnerability identification, code optimization suggestions, and multilingual explanation generation. The system can also incorporate personalized learning recommendations, collaborative explanation sharing, and offline inference using open-source Large Language Models. These improvements will expand its applicability, strengthen developer assistance, and provide a more comprehensive AI-powered software engineering support platform.

References

[1] Sommerville, I., "Software Engineering," 10th Edition, Pearson Education, 2015.

International Journal of
DATA SCIENCE AND IOT MANAGEMENT SYSTEM
Peer Reviewed, Referred & Indexed Journal
ISSN: 3068-272X www.ijdim.com Original Research Paper

[2] Pressman, R. S., "Software Engineering: A Practitioner's Approach," 8th Edition, McGraw-Hill, 2014.

[3] Fowler, M., "Patterns of Enterprise Application Architecture," Addison-Wesley, 2002.

[4] Vaswani, A., et al., "Attention Is All You Need," Advances in Neural Information Processing Systems (NeurIPS), 2017.

[5] Touvron, H., et al., "Llama: Open and Efficient Foundation Language Models," Meta AI Research, 2023.

[6] Elmasri, R., and Navathe, S. B., "Fundamentals of Database Systems," 7th Edition, Pearson, 2015.

[7] Chen, E., Huang, R., Chen, H. S., Tseng, Y. H., & Li, L. Y. (2023, June). GPTutor: a ChatGPT-powered programming tool for code explanation. In International conference on artificial intelligence in education (pp. 321-327). Cham: Springer Nature Switzerland.

[8] Kang, S., Ruan, H., & Roychoudhury, A. (2025). Autocodesherpa: Symbolic explanations in ai coding agents. arXiv preprint arXiv:2507.22414.

[9] Wang, Y., Wang, K., & Wang, L. (2023). An explanation method for models of code. Proceedings of the ACM on Programming Languages, 7(OOPSLA2), 801-827.

[10] Nassar, O., & Abdo, J. B. (2025, May). Highlight to explain: a case study on explainable AI for pattern discovery in code. In 2025 5th Intelligent Cybersecurity Conference (ICSC) (pp. 197-205). IEEE.

[11] Albrant, L., Suhag, J., Ureel II, L. C., & Jarvie-Eggart, M. (2025). Engaging Students with AI for Code Comprehension, Critique, and Explanation. In Artificial Intelligence in Education-Creating an Equitable, Creative, and Effective Learning Environment. IntechOpen.

[12] Hassany, M., & Brusilovsky, P. (2023). Worked Example Authoring Tool: Code Explanation Generation with ChatGPT.

[13] Lau, S., & Guo, P. (2023, August). From" Ban it till we understand it" to" Resistance is futile": How university programming instructors plan to adapt as more students use AI code generation and explanation tools such as ChatGPT and GitHub Copilot. In Proceedings of the 2023 ACM Conference on International Computing Education Research-Volume 1 (pp. 106-121).

[14] Dwivedi, R., Dave, D., Naik, H., Singhal, S., Omer, R., Patel, P., ... & Ranjan, R. (2023). Explainable AI (XAI): Core ideas, techniques, and solutions. ACM computing surveys, 55(9), 1-33.

[15] Piorkowski, D., Vejsbjerg, I., Cornec, O., Daly, E. M., & Alkan, Ö. (2023). AIMEE: An exploratory study of how rules support AI developers to explain and edit models. Proceedings of the ACM on Human-Computer Interaction, 7(CSCW2), 1-25.

[16] Racharak, T., Ragkhitwetsagul, C., Junplong, C., & Supratak, A. (2025). Towards human-interpretable explanation in code clone detection using LLM-based post hoc explainer. arXiv preprint arXiv:2509.22978.

[17] Rahman, S., Koana, U. A., & Nayebi, M. (2022, September). Example driven code review explanation. In Proceedings of the 16th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (pp. 307-312).

[18] Yetiştiren, B., Özsoy, I., Ayerdem, M., & Tüzün, E. (2023). Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. arXiv preprint arXiv:2304.10778.

[19] Yetiştiren, B., Özsoy, I., Ayerdem, M., & Tüzün, E. (2023). Evaluating the code quality of ai-assisted code generation tools: An empirical study on github copilot, amazon codewhisperer, and chatgpt. arXiv preprint arXiv:2304.10778.

[20] Vaidya, J., & Asif, H. (2023). A critical look at ai-generate software: Coding with the new ai tools is both irresistible and dangerous. Ieee Spectrum, 60(7), 34-39.