Repository Question Answering System with LLM Interface for Codebases

K Chandrasekar[1], Mrs R Revathy[2]
[1]PG Student, Master of Computer Applications, SRM Valliammai Engineering College, Kanttankulathur.
[2]Assitant Professor, Department of Computer Applications, SRM Valliammai Engineering college, Kantankulathur.

Abstract: The complexity of managing and comprehending big codebases has increased due to the increasing growth of software development. The Local AI-Powered Coding Assistant shown in this project is intended to increase developer productivity by facilitating intelligent natural language interaction with source code. In order to carry out intricate coding tasks like code analysis, debugging, and multi-file updates, the system incorporates a React-based agent framework that combines reasoning, action, and observation. The suggested solution offers flexibility for various user processes by utilizing a FastAPI-based backend and supporting both web and command-line interfaces. In addition to enabling smooth interaction with both local and remote (GitHub) repositories, it integrates sophisticated code intelligence functions including repository analysis, bug identification, and security evaluation. By integrating a modular tool execution layer that permits integration with a variety of development utilities, including file operations, version control systems, and search techniques, the system highlights extensibility. privacy by providing users with control over their data and computing resources.

Index Terms: Artificial Intelligence, Code Intelligence, Software Engineering, ReAct Agent, Code Analysis, Code Review, Natural Language Processing, Developer Assistant, FastAPI, Large Language Models (LLMs), Repository Management, GitHub Integration, Automation, Human-Computer Interaction, AI-Powered Development Tools.

I. INTRODUCTION

Large codebases are difficult for developers to effectively comprehend, maintain, and change due to the growing complexity of contemporary software systems. The requirement for sophisticated tools that can help with navigating and comprehending code becomes crucial as projects grow. Deep code comprehension is not well supported by traditional development tools, which frequently necessitate manual file and documentation searches. Particularly in collaborative and quickly changing workplaces, this process is laborious and error-prone. New opportunities to improve software development workflows have been made possible by recent developments in artificial intelligence, namely in Natural Language Processing (NLP) and Large Language Models (LLMs). In order to close the gap between human intent and machine-level code execution, this project presents a Local AI-Powered Coding Assistant. Through conversational interfaces, the system enables users to engage with code repositories, facilitating operations like code analysis, debugging, and editing. The assistant integrates a React-based agent architecture to carry out multi-step problem-solving tasks by combining action and reasoning. This method improves the system's capacity to manage intricate questions and provide precise, context aware answers. The suggested system's architecture is scalable and modular, offering both web-based and command-line interfaces and leveraging a FastAPI backend. With an integrated toolkit, the assistant may manage repositories, analyze code structures, and carry out development-related tasks. Furthermore, the solution offers flexibility, efficiency, and data protection by supporting both local and cloud based AI models.

II. LITERATURE REVIEW

In recent years, there has been an increase in interest in the use of artificial intelligence in software development. Early studies concentrated on rule-based systems and static code analysis tools that might identify syntax mistakes and enforce coding standards. Although these tools enhanced the quality of the code, they were unable to comprehend context and intent. Automated bug identification, code summarization, and recommendation systems became possible with the development of machine learning techniques. Nevertheless, these systems

were frequently constrained by their dependence on pre-established datasets and rules, which hindered their capacity to adapt to various programming contexts. The field of intelligent software engineering tools has changed dramatically with the introduction of Large Language Models (LLMs). Transformer-based architectures and other models have shown impressive skills to comprehend and produce code from natural language inputs. AI-powered coding assistants that offer real-time recommendations, auto completion, and documentation generation have been the subject of several research. Although these tools have demonstrated increases in developer productivity, they frequently function as passive helpers, without the capacity to carry out multi-step problem-solving tasks across whole codebases and deeper thinking.

Large Language Models (LLMs) have brought about a significant shift in the field of intelligent software engineering tools. Transformer-based architectures and other models have demonstrated remarkable abilities to understand and generate code from natural language inputs. Numerous studies have focused on AI-powered coding assistants that provide real-time advice, auto-completion, and documentation production. Despite the fact that these tools have been shown to boost developer productivity, they often serve as passive assistants, unable to perform deeper thinking and multi-step problem-solving tasks across whole codebases. Large Language Models (LLMs) have brought about a significant shift in the field of intelligent software engineering tools. Transformer-based architectures and other models have demonstrated remarkable abilities to understand and generate code from natural language inputs.

III. PROBLEM STATEMENT

It is challenging for developers to effectively comprehend, evaluate, and maintain code because modern software development requires working with big and complicated codebases. Code editors, static analyzers, and documentation systems are examples of traditional tools that offer little assistance in deciphering the context and intent of code, frequently necessitating developers to manually search through numerous files and resources. Particularly in collaborative and fast-paced development contexts, this procedure is laborious, prone to errors, and lowers overall productivity. While current AI-based coding assistants provide functions like code completion and simple recommendations, they are unable to carry out automated task execution, multi-file reasoning, or deep code analysis.

The majority of these systems are not active problem-solving agents but rather passive tools, which limits their ability to handle complicated development tasks like code change, debugging, and repository-level comprehension. Furthermore, a lot of solutions mostly rely on cloud-based models, which raises issues with latency, data privacy, and reliance on outside services. An intelligent, interactive, and effective system that can comprehend entire code repositories, react to natural language questions, and actively help developers complete coding jobs is thus required. The solution should provide flexibility, scalability, and data security while offering sophisticated features like code analysis, automated debugging, and multi-step reasoning.

IV. OBJECTIVE

1. Development of AI-Powered Coding Assistant:

The primary objective of this project is to design and develop a local AI-powered coding assistant that helps developers interact with codebases efficiently using intelligent automation and natural language processing.

2. Natural Language Code Interaction:

The system aims to enable users to understand and analyze source code by allowing them to ask questions in natural language, making code exploration easier and more intuitive.

3. Implementation of ReAct Agent Framework:

A key objective is to implement a ReAct-based agent that combines reasoning, action, and observation to perform complex coding tasks and improve decision-making capabilities.

4. Automated Code Review and Bug Detection:

The project focuses on building features that automatically review code, identify potential bugs, and highlight security vulnerabilities to improve overall code quality.

5. Multi-File Code Editing Capability:

Another objective is to support intelligent multi file code editing, enabling the assistant to modify and update multiple files based on user instructions.

6. Repository Management Integration:

The system is designed to manage code repositories efficiently by supporting both local project folders and remote repositories from platforms like GitHub.

7. Dual Interface Support (Web and CLI):

To provide flexibility, the project includes both a web-based interface and a command-line interface, allowing users to interact with the assistant in different environments.

8. Persistent Session and Memory Management:

The assistant aims to maintain conversational context by storing session data using a database, ensuring continuity and better user experience across interactions.

9. Integration with Large Language Models:

The project integrates advanced AI models through APIs to provide accurate, context-aware responses for coding queries and tasks.

10. Enhancement of Developer Productivity:

The ultimate objective is to reduce manual effort, save time, and improve efficiency in software development by providing an intelligent and automated coding assistant.

V. SYSTEM ANALYSIS

A crucial stage in the creation of the AI-powered coding assistant is system analysis, which establishes the system's functional and non-functional requirements. The suggested method uses natural language interaction to help developers comprehend, analyze, and change codebases. By automating tedious coding chores and offering insightful information about software repositories, it aims to increase productivity.

5.1 Existing System:

To comprehend and manage codebases, developers currently rely on conventional tools like code editors, documentation, and static analysis tools. Although these tools are helpful, searching, interping, and debugging code requires a lot of manual labor. Current AI-based assistants are limited in their ability to handle multi-file and repository-level tasks, and they only offer basic features like code completion and suggestions. Furthermore, the majority of tools rely significantly on cloud-based services, which raises issues with latency and data privacy.

5.2 Limitations of Existing System:

- Requires significant manual effort to understand large codebases.
- Limited understanding of full project context across files.
- Cannot effectively handle multi-file or repository-level tasks.
- Lacks advanced reasoning for complex coding problem solving.
- Debugging process remains time consuming and mostly manual.

5.3 Proposed System:

The suggested system is a Local AI-Powered Coding Assistant that lets developers communicate with codebases in natural language for effective software project comprehension, analysis, and modification. For tasks like code review, debugging, and multi-file editing, it employs a ReAct-based agent framework to carry out intelligent reasoning, carry out actions, and deliver context-aware replies. With a FastAPI backend and a modular architecture, the system allows for customizable user interaction through both online and command-line interfaces. To ensure thorough repository-level analysis, it can load local repositories or clone remote repositories from websites like GitHub. local approaches to lessen reliance on cloud services and preserve data privacy. In order to ultimately increase developer productivity and streamline intricate coding procedures, the system incorporates permanent session management.

5.4 Advantages of Proposed System:

- ✓ Reduces manual effort in understanding complex codebases.
- ✓ Provides intelligent code analysis and insights.
- ✓ Supports multi-file editing and repository-level operations.
- ✓ Improves debugging efficiency with automated suggestions.
- ✓ Reduces latency compared to fully cloud-based systems.

5.5 System Design:

The FastAPI backend, which manages all operations and API calls, is part of the system's modular architecture. For user interaction, it has a command-line interface and a web interface. To process

queries, carry out reasoning, and carry out operations like code analysis and editing, the main component makes use of a ReAct-based agent. For improved privacy, the technology offers local models and interfaces with large language models. Additionally, it stores session data in a SQLite database, guaranteeing context-aware interactions and effective performance.

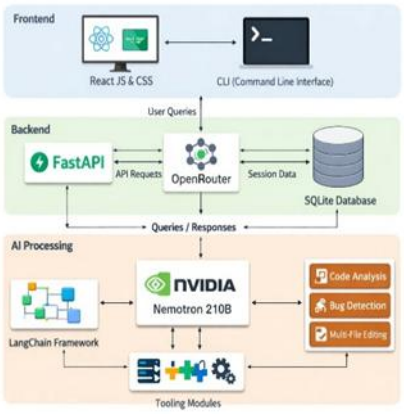

Fig 1.0 System Design

5.6 Feasibility Analysis:

1. Technical Feasibility:

The proposed system is technically feasible as it uses well-established technologies such as Python, FastAPI, React JS, and SQLite. Efficient AI processing is ensured by integrating Large Language Models using frameworks like LangChain and routing via OpenRouter.

2. Economic Feasibility:

The system is cost-effective since it primarily uses open-source technologies and frameworks. By supporting local model execution and minimizing dependence on expensive cloud services, the overall operational cost is reduced.

3. Operational Feasibility:

The system is user-friendly and designed to support both web and command-line interfaces, making it accessible to different types of users. Developers can easily interact with the system using natural language without requiring extensive training.

5.7 System Architecture:

The system architecture of the Repository Question Answering System with LLM Interface is designed using a modular and layered approach to ensure scalability, efficiency, and flexibility. The architecture consists of three main layers: frontend, backend, and AI processing layer. The frontend, developed using React JS and CSS, provides an interactive interface where users can submit queries and view responses through both web and command-line interfaces. The backend, built with FastAPI, handles API requests, manages communication between components, and stores session data in an SQLite database for maintaining context. The AI processing layer integrates LangChain with the NVIDIA Nemotron 210B model through OpenRouter, enabling advanced natural language understanding, code analysis, and multi-step reasoning using the ReAct approach. The system also includes tool modules for performing operations such as repository loading, code analysis, bug detection, and multi-file editing.

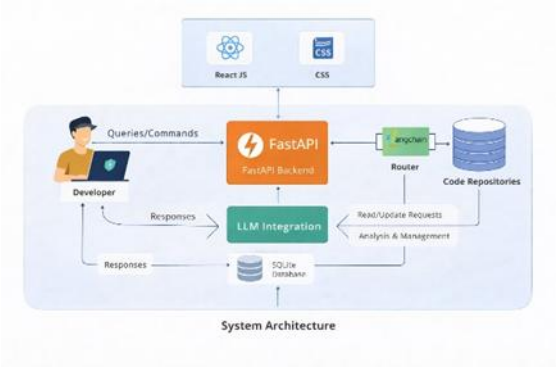

Fig 1.1 System Architecture

VI. IMPLEMENTATION

To facilitate effective engagement with a codebase, the suggested solution is implemented utilizing a combination of frontend, backend, and artificial intelligence technologies. FastAPI, which manages API calls, answers user inquiries, and interacts with the AI model, is used in the development of the backend. The source code files are extracted after the system loads the chosen GitHub repository. A language model is then used to preprocess and transform these files into embeddings. The NVIDIA Nemotron 210B model receives the retrieved context using OpenRouter, which uses LangChain to control answer creation and prompt chaining. After analyzing the context, the model produces precise, human-like responses.React.js and CSS are used in the frontend's construction to create an easy-to-use chat interface for system interaction. The user sees the AI model's final response in real time. This approach guarantees an interactive developer

experience, quicker query resolution, and effective code comprehension.

6.1 System Data Flow:

When a user sends a query via the React-based frontend interface, the system data flow starts. The FastAPI backend receives the query, processes it, and uses the language model to transform it into embeddings. Concurrently, the chosen GitHub repository's codebase is preloaded, divided into manageable portions, and converted into embeddings that are kept in a vector database. The system retrieves the most pertinent code snippets by conducting a similarity search in the vector database upon receiving a query. The user query and these collected results are then sent via OpenRouter via LangChain to the NVIDIA Nemotron 210B model. The model provides a relevant response after analyzing the given context. Finally, the generated answer is sent back through the backend API and displayed to the user in the frontend interface, ensuring a smooth and interactive query-response cycle.

Fig 1.2 Data Flow Diagram

6.2 System Database Design:

The system database is made to effectively handle the semantic information and structured data needed for the repository question answering system. While a vector database is used to store code embeddings for quick similarity searches, **SQLite** is used to store structured data, including user information, repository details, code chunks, and query history. The code is split up into smaller sections and saved with file metadata when a repository is loaded. After being transformed into vector embeddings, these segments are saved in the vector database, making it possible to quickly retrieve pertinent code in response to user queries. In order to enhance system performance and enable historical monitoring, the database also keeps track of user queries and generated responses.

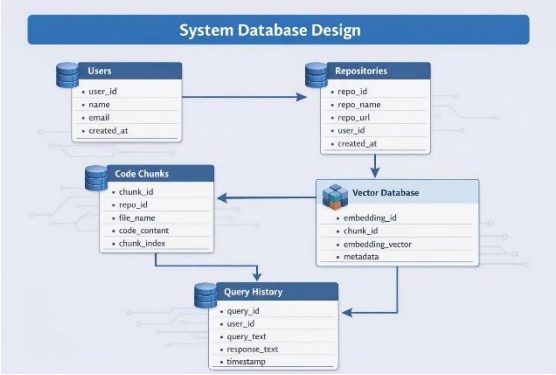

Fig 1.3 Database Diagram

VII. SYSTEM TESTING

System testing is carried out to ensure that the entire application functions correctly and meets the specified requirements. In this project, testing is performed on both the frontend and backend components to verify the integration of all modules, including code retrieval, embedding generation, vector database storage, and AI response generation. Various test cases are executed to validate user inputs, API responses, and the accuracy of retrieved code snippets. The system is also tested for performance, ensuring quick response times when handling queries. Error handling and edge cases, such as invalid repository links or empty queries, are thoroughly checked to maintain system stability. Overall, system testing ensures that the application operates smoothly, delivers accurate results, and provides a reliable user experience.

VIII.RESULTS

The successful development and testing of the "Repository Question Answering System with LLM Interface for Codebases" showed that it could efficiently analyze and communicate with software repositories. Based on user inquiries, the system accurately processes code from GitHub repositories and finds pertinent code samples. It produces accurate and context-aware responses that help users comprehend, debug, and navigate complex codebases by utilizing the LLM interface. The findings demonstrate a notable decrease in the amount of manual labor needed for code analysis and an increase in the effectiveness of obtaining pertinent data. Additionally, the system maintains backend processing performance while offering a seamless and interactive user experience through its frontend interface. All things considered, the project achieves its goals by providing a clever, quick, and

easy-to-use solution for repository-based question answering.

8.1 Repository Selection and Initialization Interface: The AI-powered code assistant's first interface is represented by this screen, where users can choose an existing repository from the repository list or clone a GitHub repository. It has features to load codebases for analysis, maintain sessions, and enter a repository URL. Before allowing contact with the AI assistant, the system makes sure that a repository is chosen, which serves as the foundation for code comprehension and query processing.

Fig 1.4 Repository Selection and Initialization Interface

8.2 Interface for AI Coding Assistance: With this interface, users may inspect file structures, clone and explore GitHub repositories, and perform natural language ueries to interact with the codebase. It facilitates code comprehension, problem solving, and increased developer productivity.

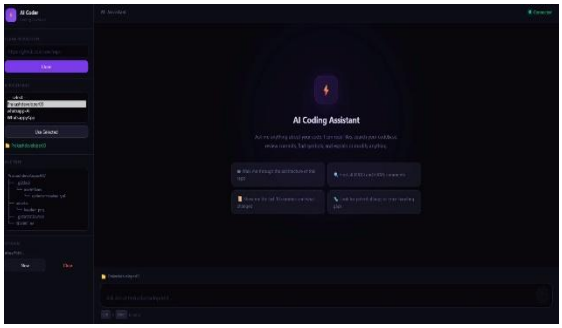

Fig 1.5 Interface for AI Coding Assistance

8.3 AI-Powered Interface Repository Analysis: An AI-powered coding assistant interface that enables users to clone and examine GitHub repositories is seen in this screenshot. The system maps the repository structure, reads files like README and workflow configurations, and gives a conversational architectural analysis. To assist developers in effectively understanding codebases, it offers capabilities including repository selection, file tree visualization, and step-by-step AI insights.

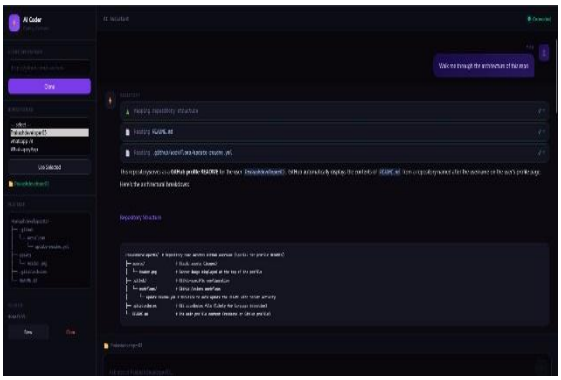

Fig 1.6 AI-Powered Interface Repository

IX. CONCLUSION

The Repository Question Answering System with LLM Interface for Codebases effectively illustrates how contemporary AI methods may streamline developer productivity and code comprehension. The solution allows users to engage with complex repositories using natural language queries by combining Large Language Models with code embedding and retrieval techniques. This expedites the troubleshooting, learning, and development processes and lessens the need for manual code exploration. The project demonstrates how sophisticated LLMs, React, FastAPI, and LangChain can be combined to produce an effective, scalable, and user-friendly solution. All things considered, the system demonstrates how AI-driven tools can revolutionize conventional software development processes and enhance code accessibility.

X. FUTURE ENHANCEMENTS

Several sophisticated features can be added to the Repository Question Answering System with LLM Interface for Codebases to make it even better. Larger repositories with improved efficiency and support for several programming languages could be future improvements. Multiple users can interact with the system at once if real-time collaboration tools are integrated. Large Language Models that are more precise and refined can also be added to the system to improve context comprehension and answer quality. The system can be made more user-friendly by incorporating chatbot interfaces and voice-based questions. The system can become a full-fledged AI-powered development assistant by adding automated code recommendations, bug

detection, and security analysis. Accessibility and efficiency will be further enhanced by deployment on cloud platforms with scalable infrastructure.

REFERENCES

[1] N. Mulla and P. Gharpure, "Automatic Question Generation: Methodologies and Applications," Progress in Artificial Intelligence, vol. 12, 2023.
[2] S. Ansari, R. Raut, and B. K. Shah, "A Review of Question Answering Systems: Approaches and Challenges," International Journal of Computer, 2023.
[3] P. Upadhyay, T. Panda, P. Jaidka, and N. Gupta, "AskAI: BERT-Based Contextual Framework for Question Answering," IEIE Transactions on Smart Processing & Computing, 2025.
[4] S. Sharma, S. Jain, M. K. Tiwari, and S. Lal, "Knowledge-Based Question Answering: Patterns and Progress," International Journal of Computers and Applications, 2025.
[5] C. Agarwal, K. Dutta, and P. Singh, "Anaphora Resolution for Indian Languages in NLP," International Journal of Computational and Experimental Science and Engineering, 2025.
[6] K. D. Singha, "AI-Based Code Generator and Context-Aware Question Answering System," International Journal for Multidisciplinary Research (IJFMR), 2025.
[7] A. Tripathi and D. Anand, "Advances in Artificial Intelligence Applications," in Proc. World Conference on AI (WCAIAA), Springer, 2023.
[8] A. Tripathi and D. Anand, "Recent Trends in AI and NLP Systems," in Proc. WCAIAA, Springer, 2024.
[9] C. S. Alaparthi, "Learning to Rank Passages for Web Question Answering," Microsoft AI Challenge India, 2019.
[10] R. Goel et al., "Hybrid RAG for Legal Reasoning in India," arXiv preprint, 2025.
[11] J. Gupta, A. Sharma, and S. Singhania, "Legal Assist AI: Transformer-Based QA System for India," arXiv preprint, 2025.
[12]Indian Journal of Artificial Intelligence Research, "AI Research and Applications in India," INDJAIR, 2024.