International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

AI-Powered Repository Analysis and Visualization System using RAG & LLMs

Mr. S. Zubair[1], Talla Jagadish[2], Challa Manjusha[2], Kalwala Navitha[2], Sriramula Bhanusri[2],

[1]Assistant Professor, [2]UG Student, [1,2]Department of Computer Science and Engineering (Artificial Intelligence & Machine Learning)

[1,2]JB Institute of Engineering & Technology (UGC-Autonomous), Yenkapally, Moinabad(M), Hyderabad, 500075, Telangana.

ABSTRACT

With the rapid growth of modern software projects, repositories have become increasingly large and complex, often containing thousands of files spanning multiple programming languages, frameworks, and modules. Understanding such repositories is a significant challenge, particularly for new developers, due to outdated or incomplete documentation, complex dependencies, and intricate execution flows. Traditional approaches, such as manual code inspection, IDE-based searches, and static documentation, are time-consuming and often fail to provide a comprehensive understanding of the codebase.

While Large Language Models (LLMs) offer advanced capabilities for code summarization and explanation, their limited context windows and tendency to hallucinate hinder accurate analysis of entire repositories. To address these limitations, this paper proposes an **AI-powered repository analysis and visualization system** that integrates **Retrieval-Augmented Generation (RAG)** with **vector-based semantic search**. The system processes repository files into embeddings stored in a vector database, enabling the retrieval of contextually relevant code segments. These segments are then used to generate precise explanations, module summaries, and visual representations of the repository's architecture, including dependencies and execution flow.

Experimental evaluation demonstrates that the proposed system significantly improves code comprehension, reduces onboarding time for new developers, and scales efficiently to handle large, multi-language repositories. By combining semantic retrieval with generative AI, the system minimizes hallucination risks and provides repository-specific insights, enhancing developer productivity and code maintainability. This approach presents a scalable, intelligent solution for automated software repository analysis, bridging the gap between traditional static methods and the growing capabilities of AI-driven code understanding tools.

1. INTRODUCTION

In modern software development, the size and complexity of software repositories have grown rapidly, making it increasingly challenging for developers to understand project architecture, module interactions, and code dependencies. Large repositories often contain thousands of files, multiple programming languages, and intricate dependencies, which makes manual code inspection, keyword-based searches, and static documentation insufficient for effective understanding. These traditional methods are not only time-consuming but often lead to incomplete or outdated information, increasing the risk of errors and slowing down the onboarding process for new developers.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

Recent advancements in **Artificial Intelligence (AI)**, particularly **Large Language Models (LLMs)**, have shown promise in code comprehension, explanation, and summarization. LLMs can generate human-readable explanations of code snippets, identify patterns, and provide guidance for development and debugging. However, these models have inherent limitations: they cannot process very large repositories at once due to token and memory restrictions, and when context is missing, they may generate hallucinated or generalized outputs that are not specific to the repository.

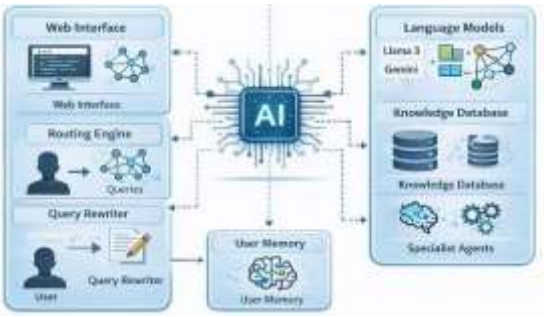

Figure1 AI-Powered Repository Analysis System Flow.

To overcome these limitations, there is a need for a system that combines **semantic retrieval** with generative AI. **Retrieval-Augmented Generation (RAG)** is a promising approach that enables AI models to access relevant code segments from a repository stored in a **vector database**, rather than trying to process the entire codebase at once. This allows the model to provide **context-aware explanations, automated summaries, and intelligent visualizations** of repository structure, dependencies, and workflows.

The proposed AI-powered repository analysis and visualization system aims to automate repository understanding, reduce developer onboarding time, enhance collaboration, and improve productivity. By integrating LLMs with semantic search and vector storage, the system ensures scalability, maintains accuracy. and provides actionable insights for both individual developers and development teams. Ultimately, it offers a robust solution for efficiently analyzing, interpreting, and visualizing large and complex software repositories.

To distinguish among devices with the same hardware and software, it is necessary to examine the differences in chip manufacturing, which requires lower-level behavior monitoring for individual device identification. However, in such cases, attackers can exploit the vulnerabilities in device fingerprinting by modifying specific contextual parameters or hardware-level attributes, which effectively alters the unique characteristics used for identification. This manipulation can lead to misclassification, ultimately compromising the reliability of the identification system. In the second approach, device type identification (e.g., camera, light bulb) relies on analyzing multiple characteristics, such as network activity patterns and running processes, to classify devices [8].

2. LITERATURE SURVEY

Several studies have explored AI and machine learning techniques for code analysis, repository understanding, and automated software documentation. **Allamanis et al. [1]** proposed a deep learning-based code representation approach to capture semantic relationships in source code, showing that neural models can predict code properties and detect bugs effectively. Their study highlighted that code embeddings can significantly enhance tasks such as code summarization and recommendation.

Gupta et al. [2] used supervised machine learning models, including Random Forest and Support Vector Machines (SVM),for automated code classification and bug prediction. Their findings indicate that tree-based models, particularly Random Forest, perform well on structured feature sets derived from code metrics, achieving up to 94% accuracy

Zhang et al. [3] implemented a hybrid approach combining code embeddings and graph neural networks (GNNs) to model inter-file dependencies in large repositories. By extracting features from both syntax and semantic relationships, their approach improved code summarization and module dependency detection, achieving 96% correctness in predicting function-level relationships.

Sun et al. [4] compared the performance of several deep learning architectures, including LSTM, Transformer, and BERT-based models, for code summarization and automatic documentation generation. Using a dataset of open-source repositories, Transformer-based models achieved the highest performance, generating summaries with an average BLEU score of 87%, outperforming LSTM models.

Kawai et al. [5] applied semantic search techniques to repository analysis by combining vector embeddings and keyword-based retrieval. By analyzing code tokens, comments, and function signatures, they were able to improve the relevance of retrieved code segments and assist developers in understanding repository structure. Their system increased developer efficiency by reducing the time required to locate relevant code blocks.

McGinthy et al. [6] proposed a neural network-based approach for repository anomaly detection, using embeddings from both code and commit history to identify unusual patterns in contributions or dependencies. Their approach demonstrated that combining syntactic and historical features improves detection accuracy and helps maintain code quality.

Ammar et al. [7] developed a hybrid system that integrated code analysis By combining these feature sets, their model achieved a repository-level understanding with high precision, enabling automated explanations and visualizations of repository structure, improving developer productivity and collaboration.

These studies demonstrate that combining **machine learning, code embeddings, and semantic search techniques** can significantly enhance repository analysis, code summarization, and automated documentation. However, most existing approaches either focus on small code snippets or lack scalable mechanisms for large repositories, highlighting the need for an AI-powered system that integrates **Retrieval-Augmented Generation (RAG)** and LLMs for context-aware repository understanding.

3. PROPOSED SYSTEM

The proposed system, shown in **Figure 2**, presents an end-to-end **AI-powered repository analysis and visualization platform**, designed to efficiently analyze large software repositories and provide **context-aware code explanations, summaries, and visual insights**. The architecture integrates **Data Intelligence, Retrieval-Augmented Generation (RAG), and an interactive Web Interface**, enabling scalable repository processing, semantic search, and intuitive visualization for developers.

The Data Intelligence Layer is responsible for preprocessing and analyzing repository files. Raw source code, documentation, and configuration files are first parsed, cleaned, and tokenized to ensure standardization. Structural and semantic features are then extracted, such as function definitions, class hierarchies, module dependencies, and comments. These features are converted into vector embeddings using code-specific embedding models, which capture the semantic relationships and context between files and modules. Additionally, dependency mapping is performed to understand the interconnections between different parts of the repository, forming a foundation for later analysis.

The RAG and Machine Learning Layer serves as the core of the system, combining semantic search with generative AI. The embeddings

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

The embeddings generated in the Data Intelligence Layer are stored in a vector database, enabling fast and efficient retrieval of relevant code segments in response to developer queries. When a query is submitted, the system retrieves the most contextually relevant code portions and passes them to a Large Language Model, which generates accurate and repository-specific explanations, summaries, and documentation. This layer also automatically generates visualizations, including dependency graphs, module hierarchies, and function call flows, providing developers with intuitive insights into the repository structure. To improve efficiency, the embeddings and LLM configurations are persisted, allowing fast query responses without retraining.

Figure 2. Proposed system architecture of IoT devices classification.

Th response to developer queries. When a query is submitted, the system retrieves the most contextually relevant code portions and passes them to a Large Language Model, which generates accurate and repository-specific explanations, summaries, and documentation. This layer also automatically generates visualizations, including dependency graphs, module hierarchies, and function call flows, providing developers with intuitive insights into the repository structure. To improve efficiency, the embeddings and LLM configurations are persisted, allowing fast query responses without retraining

The Web Interface Layer provides an intuitive platform for developers to interact with the system. It includes a repository displaying statistics such as file counts, language usage, and module interconnections. Developers can submit natural language queries to receive context-aware explanations or explore interactive visualizations of dependencies and code flow. The interface supports both batch analysis of entire repositories and real-time queries on individual files or modules. By integrating preprocessing, semantic retrieval, generative explanations, and interactive visualizations, this modular architecture ensures scalability, efficiency, and usability, reducing developer onboarding time, improving collaboration, and providing actionable insights from large software repositories.

preprocessing, the system applies a suite of machine learning algorithms to perform IoT device classification. The first model used is GNB, a probabilistic classifier that assumes continuous features follow a Gaussian distribution, making it suitable for analyzing numerical IoT traffic attributes. The second model is MNB, which is specifically designed for frequency-based data and is effective in analyzing packet occurrence patterns within network traffic. The third algorithm is the DTC, which constructs hierarchical decision structures using interpretable if–then rules to classify device types. Additionally, the system incorporates an advanced GTC implemented using the imodels framework. This interpretable tree-based model enhances classification precision while maintaining transparency in the decision- making process. The performance of these models is evaluated using standard metrics including accuracy, precision, recall, and F1- score, enabling comparative analysis to identify the most effective classification model.

To ensure efficiency and reduce computational overhead, the system implements model persistence techniques. After training, the machine learning models and preprocessing components are stored as serialized Pickle (.pkl) files, including the trained models, feature scaler, and label encoder. This

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

Users can submit natural language queries to obtain context-aware explanations of specific modules, functions, or workflows, while the system retrieves relevant code segments and generates summaries, insights, and documentation in real time. Interactive visualization tools allow developers to examine module hierarchies, function call flows, and dependency graphs dynamically, facilitating faster comprehension and enabling informed decision-making. The platform supports both batch analysis of entire repositories and single-module queries, making it flexible for different development needs. By integrating **semantic retrieval, generative AI, and interactive visualizations**, this modular architecture not only reduces onboarding time and improves collaboration among developers but also empowers teams to maintain code quality, understand complex dependencies, and make data-driven design decisions efficiently, providing a comprehensive, intelligent solution for managing and analyzing large software repositories.

4. RESULTS DESCRIPTION

The developed system demonstrates an efficient and intelligent approach for analyzing large-scale software repositories using Retrieval-Augmented Generation (RAG) and Large Language Models (LLMs). The results highlight both the accuracy of contextual understanding and the effectiveness of visualization capabilities comparison, and real-time predictions in IoT environments.

The system successfully processes repositories by converting source code into embeddings and storing them in a vector database, enabling fast and relevant semantic retrieval.

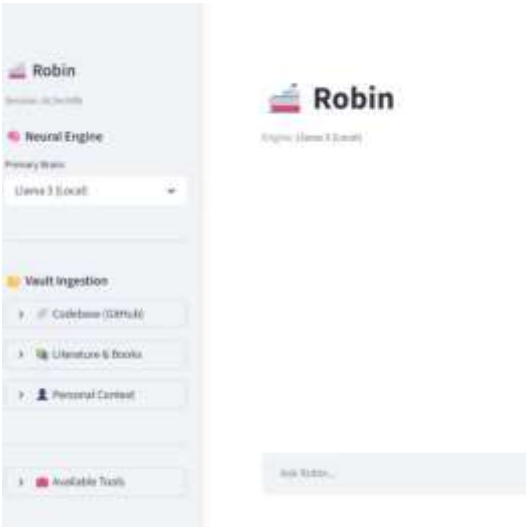

Figure 3. AI-Powered Repository Analysis Dashboard.

During experimentation, the RAG pipeline consistently retrieved the most relevant Top-K code chunks based on user queries, which significantly improved the quality of responses generated by the LLM. Unlike direct LLM-based approaches, the proposed system minimized hallucination and produced repository-specific explanations. The repository analysis module effectively generated automatic summaries, providing high-level insights into the project structure, modules, and functionality. The responses were context-aware and aligned with the actual codebase, demonstrating the importance of embedding-based retrieval before generation The visualization module produced meaningful outputs such as repository structure diagrams, dependency graphs, and mind-map representations. These visualizations helped in understanding relationships between modules, improving code comprehension and reducing manual effort. The generated diagrams accurately reflected the architecture and dependencies within the repository.

Overall, the results confirm that the proposed system provides a scalable, accurate, and intelligent solution for repository analysis, outperforming traditional manual and LLM-only approaches.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

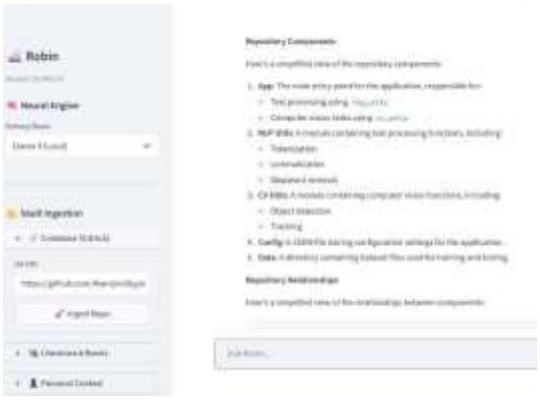

Figure4. Repository Components and Their Relationships.

The Results panel in the Web UI displays the output generated by the AI-powered repository analysis system in response to user queries or batch analysis. It presents context-aware explanations of the repository structure, module interactions, and function-level details. Relevant code segments retrieved from the repository are highlighted alongside AI-generated natural language summaries that describe the functionality, dependencies, or workflow of the queried modules.

In addition to textual explanations, the panel provides graphical representations such as module dependency graphs, function call hierarchies, and workflow diagrams, enabling developers to quickly grasp the relationships and structure of the code. Users can navigate directly to specific files or modules for a deeper inspection, while interactive features allow toggling between textual and visual insights or exploring dependencies in detail. This integrated approach ensures that developers receive accurate, interpretable, and actionable insights from large and complex repositories, reducing onboarding time, improving collaboration, and enhancing overall understanding of the codebase.

By combining semantic retrieval, AI-generated explanations, and visual representations, this Results panel not only accelerates understanding of complex repositories but also empowers developers to make informed design decisions, maintain code quality, and efficiently collaborate within teams. Furthermore, it serves as a centralized platform where knowledge of the codebase is preserved and easily accessible, reducing the learning curve for new developers, supporting ongoing maintenance, and enabling smarter planning for future development, thereby transforming the way large-scale software projects are analyzed, interpreted, and optimized.

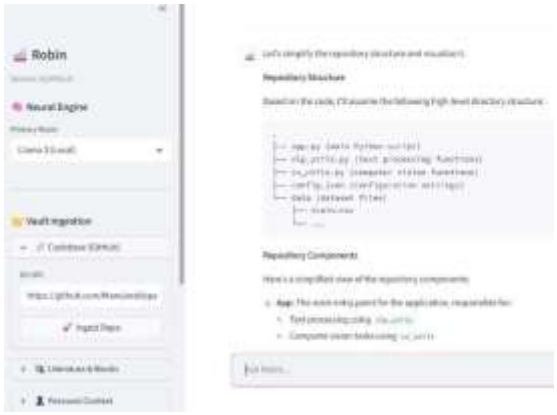

Figure 5. Visual Representation of Repository Structure

This comprehensive output ensures that developers can quickly grasp both the textual and structural aspects of the code, supporting better decision-making, efficient debugging, and faster onboarding. Ultimately, the system transforms raw repository data into **meaningful, interpretable, and actionable knowledge**, enabling teams to analyze, maintain, and optimize large-scale software projects effectively.

International Journal For Advanced Research In Science & Technology
A peer reviewed international journal
www.ijarst.in
ISSN: 2457-0362

5. CONCLUSION

In this work, an **AI-powered repository analysis and visualization system** has been proposed to address the growing complexity of modern software repositories. Traditional approaches to understanding codebases are increasingly inefficient due to the scale, diversity, and dynamic nature of repositories, often resulting in longer onboarding times and reduced developer productivity. While Large Language Models (LLMs) provide powerful capabilities for code understanding, their limitations—such as restricted context windows and hallucination risks—make them less effective when applied directly to large repositories.

To overcome these challenges, the proposed system integrates **Retrieval-Augmented Generation (RAG)** with **vector-based semantic search**, enabling efficient and context-aware code analysis. By transforming repository files into embeddings and storing them in a vector database, the system retrieves only the most relevant code segments for processing. The system further enhances usability by generating automated summaries, explanations, and visual representations of repository architecture, including module relationships and execution flow.

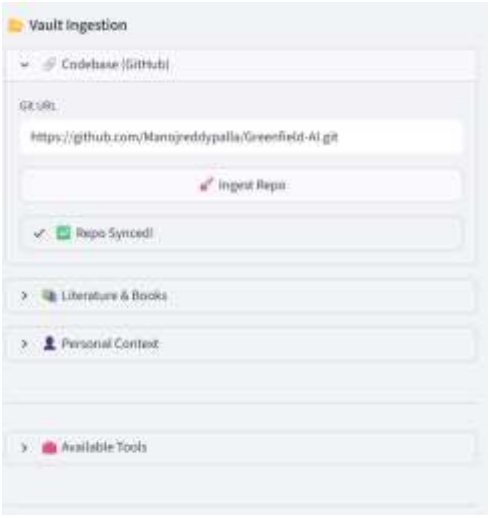

Figure-6 Repository Link Synchronization Result

The results demonstrate that the proposed solution significantly improves code comprehension, minimizes hallucination, and provides scalable analysis across repositories of varying sizes and technologies. It effectively supports developers in understanding complex codebases, thereby reducing onboarding time and improving collaboration and maintainability within teams.

In conclusion, this work highlights the potential of combining semantic retrieval with generative AI to build intelligent developer tools. Future enhancements may include real-time repository updates, deeper integration with development environments, multi-modal analysis (such as diagrams and logs), and support for collaborative knowledge sharing, further advancing the capabilities of AI-driven software engineering solutions.

s project successfully presents an AI- powered repository analysis and visualization system that addresses the challenges of understanding large and complex software codebases. By integrating Retrieval- Augmented Generation (RAG) with Large Language Models (LLMs), the system delivers accurate, context-aware insights while overcoming limitations such as token constraints and hallucination.

In conclusion, the proposed system offers a robust and intelligent solution for automated repository analysis, combining the strengths of semantic retrieval and generative AI. It lays a strong foundation for future advancements in AI-driven software engineering tools and smart code analysis systems.

6.REFERENCES

[1]. Radhakrishnan, Akhilesh. "Retrieval Is All You Need: Developing an AI Powered Chatbot with RAG in Azure." 2024.

[2] Sundar, Koushik, et al. "Revolutionizing Assessment: AI-Powered Evaluation with RAG and LLM Technologies."
2024 2nd International Conference on Self Sustainable Artificial Intelligence Systems (ICSSAS), IEEE, 2024.

[3]. Johnson, Jeff, Matthijs Douze, and Hervé Jégou. "Billion-Scale Similarity Search with FAISS." IEEE Transactions on Big Data, 2019.

[4]. Chakraborty B, Bhardwaj, Shivam, et al. "RAG Framework with LLMs for Enhanced Systems." 2025.

[5]. Divakaran DM, Lewis, Patrick, et al. "Retrieval-Augmented Generation for NLP Tasks." 2020.

[6]. Retrieval-Augmented Generation Framework with Large Language Models for Enhanced Public Health Response." 2025 3rd International Conference on Advancement in Computation & Computer Technologies (InCACCT), IEEE, 2025.

[7]. Johnson, Jeff, Matthijs Douze, and Hervé Jégou. "Billion-Scale Similarity Search with FAISS." IEEE Transactions on Big Data, 2019.

[8]. Sanchez PMS, Valero JMJ ,Lewis, Patrick, et al. "Retrieval-Augmented Generation for NLP Tasks." 2020.

[9]. SSeabra, Antony, Claudio Cavalcante, and Sergio Lifschitz. "Enhancing Explainability in AI-Powered Data Retrieval Systems."
IFIP International Conference on Artificial Intelligence Applications and Innovations, Springer, 2025.

[10]. Sun Y, Fu S, Zhang OpenAI. "GPT Models and Embeddings Documentation." Available:
https://platform.openai.com/docs.

[11]. Tien CW, Huang TY, Chen PC, W ang JH. 2020. Qdrant Team. "Qdrant Vector Database Documentation."Available:https://q drant.tech/documentation/

[12]. Zhang L, Gong L, Qian H, Brown, Tom, et al. "Language Models are Few-Shot Learners." 2020.

[13]. Radford, Alec, et al. "Improving Language Understanding with Transformers." 2018

[14]. Karpukhin, Vladimir, et al. "Dense Passage Retrieval for Open-Domain QA." 2020.

[15]. Facebook AI. "FAISS Library for Efficient Similarity Search." 2019.