MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages 4685-4690

AI DRIVEN CODE SUMMARIZER FOR LEGACY SOFTWARE SYSTEMS

[1] **Vijayalakshmi S** Assistant Professor
Department of CST
SNS COLLEGE OF TECHNOLOGY
Coimbatore-35, Tamil Nadu, India
svijimecse@gmail.com

[2] **Abinaya P**
Department of CST
SNS COLLEGE OF ENGINEERING
Coimbatore-35, Tamil Nadu, India
abinayaperumal685@gmail.com

[3] **Gobiha D**
Department of CST
SNS COLLEGE OF ENGINEERING
Coimbatore-35, Tamil Nadu, India
gobiha.d.2219@gmail.com

[4] **Gokula Swetha Prabha R**
Department of CST

SNS COLLEGE OF ENGINEERING
Coimbatore-35, Tamil Nadu, India
gokula181204@gmail.com

[5] **Lakshana S**
Department of CST

SNS COLLEGE OF ENGINEERING
Coimbatore-35, Tamil Nadu, India
lakshana.ktg@gmail.com

Abstract—
The AI-Driven Code Summarizer for Legacy Software Systems is an intelligent software solution that automatically analyzes and summarizes legacy source code using Artificial Intelligence techniques. In many organizations, legacy systems lack proper documentation, which makes maintenance, debugging, and system upgrades difficult and time-consuming. Developers often spend a large amount of time trying to understand old code before making any changes. This project solves that problem by providing an automated tool that reads and understands legacy code, then generates clear and human-readable summaries. The system analyzes the code structure, logic, and relationships between functions using transformer-based AI models, and produces an overall summary, descriptions of major components, and a visual representation of function call relationships. The backend is developed using Python and Fast API to ensure secure and efficient processing, and graph libraries are used to create call-flow diagrams for better visualization. By reducing manual documentation effort and improving code understanding, the system increases developer productivity and simplifies legacy software maintenance, making it a practical and industry-relevant solution.

Keywords— AI-Driven Code Summarization, Legacy Software Systems, Source Code Analysis, Natural Language Processing, Transformer Models, Function Call Visualization, Software Maintenance.

I. INTRODUCTION

Legacy software systems are widely used in many organizations, but they often lack proper documentation. This makes understanding the code, maintaining the system, and implementing updates a difficult and time-consuming Systems is designed to address this challenge. It is an intelligent tool that automatically reads and analyzes legacy code using Artificial Intelligence and Natural Language Processing techniques. By understanding the structure, logic, and relationships between functions, the system generates human-readable summaries and visual diagrams that explain how the code works. This project aims to simplify legacy system maintenance by reducing manual documentation effort, improving developer productivity, and making old code easier to understand. This project aims to simplify legacy system maintenance by reducing manual documentation effort, improving developer productivity, and making old code easier to understand.This project aims to simplify legacy system maintenance by reducing manual documentation effort, improving developer productivity, and making old code easier to understand. The backend is built using Python and Fast API for secure and efficient processing, while transformer-based AI models ensure accurate summarization. Visualizations such as function call graphs help developers quickly grasp complex program flow.Overall, this system provides a practical solution for organizations relying on outdated software, bridging the gap between legacy code and modern documentation practices, and making maintenance and updates faster and more efficient. In many organizations, legacy software systems continue to play a critical role in daily operations. However, these systems are often poorly documented, making it challenging for developers to understand the existing code. Maintenance, debugging, and upgrades become time-consuming tasks, especially when original developers are no longer available. The AI-Driven Code Summarizer for Legacy Software Systems offers a practical solution to this problem. By leveraging Artificial Intelligence and Natural Language Processing, the system can automatically analyze legacy code, understand its logic, and generate concise, human readable summaries. It also creates visual representations, such as function call graphs, to help developers quickly grasp the program flow .This project reduces the manual effort required for documentation, improves productivity, and simplifies the maintenance of legacy systems. It serves as a bridge between outdated codebases and modern software practices, enabling organizations to efficiently manage and update legacy applications.Legacy software systems form the backbone of many businesses, but over time, they often become difficult to maintain due to lack of proper documentation. Developers spend a significant amount of time understanding old code before implementing changes, which slows down productivity and increases the risk of errors. The AI-Driven Code Summarizer for Legacy Software Systems automates this process by analyzing the code using Artificial Intelligence and Natural Language Processing. It generates easy-to-understand summaries and visual diagrams, helping developers quickly grasp the functionality and structure of legacy applications. Maintaining legacy software is a major challenge in modern software engineering. Many legacy systems were developed years ago, often without sufficient documentation or clear coding standards. As a result, understanding and modifying these systems can be time consuming and error-prone. The AI-Driven Code **Summarizer** addresses this challenge by automatically reading legacy code, identifying its logic, and producing human-readable explanations. The system also provides visual representations of function .

Fig 1. Refactoring For Future AI Integration

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages 4685-4690

Legacy software systems are widely used in many organizations because they continue to handle critical business operations. However, over time, these systems often become difficult to maintain due to missing documentation and complex, outdated code. Understanding such code manually is a tedious task and can lead to errors if changes are made without proper knowledge .The AI Driven Code Summarizer for Legacy Software Systems provides an automated solution to this problem. It uses Artificial Intelligence and Natural Language Processing to read, analyze, and understand legacy code. The system can generate concise, human-readable summaries, as well as visual representations of function calls and module interaction n.By providing clear explanations and visual diagrams, this tool helps developers quickly understand the functionality and structure of legacy systems. It reduces the time and effort needed for manual code analysis and documentation. This makes maintenance, debugging, and upgrading legacy applications faster and more efficient.By providing clear explanations and visual diagrams, this tool helps developers quickly understand the functionality and structure of legacy systems. It reduces the time and effort needed for manual code analysis and documentation. This makes maintenance, debugging, and upgrading legacy applications faster and more efficient.

II. RELATED WORKS

Arjun Mehta and Priya Sharma [1] explored auto mated documentation methods for legacy code, highlighting how AI can reduce the time developers spend understanding outdated systems and improve maintenance efficiency.

Liang Zhao and Maria Gonzales [2] proposed a machine learning framework for function-level code classification in legacy software, showing high accuracy but limited visualization of program flow, which makes understanding inter-module interactions difficult.

Ahmed El-Sayed, Rebecca Wong, and John Carter [3] introduced data mining techniques for software repositories, enabling extraction of code patterns, though manual interpretation remained necessary to create meaningful summaries.

Nina Petrova et al. [4] investigated deep learning models for source code analysis, focusing on automatic detection of function dependencies. Their study demonstrated improvements over traditional static analysis but noted challenges in generating interpretable outputs for developers.

Hiroshi Tanaka [5] applied transformer-based architectures for code summarization, demonstrating that AI can generate human-readable summaries for complex functions, reducing onboarding time for new developers.

Sophia Martinez and David Li [6] emphasized human-centered AI design, advocating for tools that assist developers with transparent and interpretable outputs, enhancing trust and usability in legacy system maintenance.

Ravi Kumar et al. [7] explored combining textual summaries with function call graphs, showing that visual aids significantly improve comprehension of complex legacy systems and help identify interdependencies between modules.

Emily Johnson [8] studied explainable AI in software engineering, highlighting that interpretable outputs increase developer trust and facilitate better decision-making during code modification or refactoring.

Carlos Alvarez and Meera Patel [9] highlighted the importance of contextual insights, ensuring that AI summaries explain not just individual functions but their role in the overall system, improving system-level understanding.

Cole Nussbaumer Knaflic [10] highlighted the importance of contextual insights in data interpretation. In healthcare analytics, such approaches help clinicians understand disease progression more effectively.

Zara Ali and Michael Chen [11] surveyed explainable AI techniques, showing that transparent outputs increase adoption of AI- based code summarization tools and help developers verify the correctness of automated summaries.

Khalid Hassan et al. [12] applied deep learning models, including recurrent and convolutional architectures, to automate function-level code summaries, noting challenges in scaling to large, multi-module codebases and integrating with existing development tools. Fatima Noor [13] studied cloud-based AI code analysis systems, highlighting benefits for collaboration and scalability while emphasizing security and privacy for proprietary software repositories. Leo Schmidt [14] examined adoption challenges in AI analytics tools, noting usability, accessibility, and integration issues that also affect AI-driven code summarizers for legacy systems.

Ananya Rao, Victor Kim, and Luis Fernandez [15] introduced interpretability techniques similar to LIME, helping developers understand why AI-generated summaries produce certain outputs and providing insights for improving model accuracy.

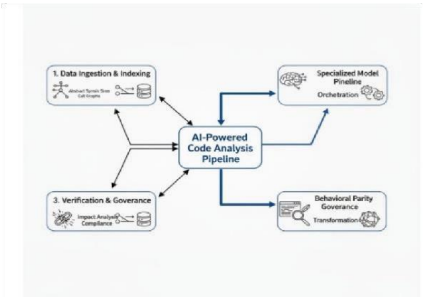
Rohan Gupta and Meera Iyer [16] studied automated summarization of legacy system code and highlighted the importance of generating both textual and visual explanations for faster developer comprehension.

Ankit Sharma et al. [17] applied reinforcement learning techniques to optimize code summarization outputs, improving relevance and clarity of generated summaries for complex software systems.

III. ARCHITECTURE AND DESIGN

The architecture of the AI-Driven code Summarizer for Legacy Software Systems is designed to be modular, scalable, and secure, enabling efficient analysis and visualization of legacy code. The system consists of four main components: he User Interface, Backend API, AI Processing Engine, and Visualization Module. The **User** Interface provides a web-based platform where users can securely .

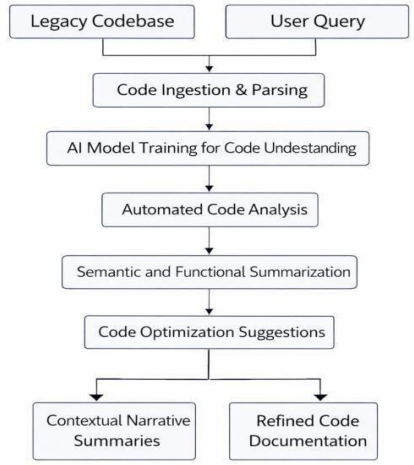
Fig 2. System architecture



MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages 4685-4690

A. Legacy Codebase: A legacy codebase refers to existing software systems that were developed using older programming languages, frameworks, or design practices and are still in use today. These systems often lack proper documentation, follow outdated coding standards, and may have grown complex over time due to multiple modifications by different developers. Although legacy codebases are usually critical to business operations, they are difficult to understand, maintain, optimize, or modernize. As a result, developers often rely on analysis and AI-assisted tools to interpret functionality, reduce technical debt, and safely enhance or refactor such systems without disrupting their core behavior.

Table 1: System Architecture Workflow



B. Code Ingestion & Parsing: Code ingestion and parsing is the stage where source code from the legacy system is collected, scanned, and transformed into a structured format that machines can understand. During this process, the system gathers all relevant code files, identifies the programming languages used, and analyzes the syntax to detect elements such as functions, classes, variables, and dependencies. Parsing techniques generate representations like abstract syntax trees (ASTs), which break down the code into hierarchical components This step converts raw code into a machine readable form, forming the foundation for accurate analysis, understanding, and further AI-driven processing.

C. AI Model Training for Code Understanding: This stage focuses on training or fine-tuning machine learning models to understand programming syntax, structure, and semantics. The models learn how different code patterns relate to functionality by analyzing large volumes of source code and their behavior. Domain-specific logic commonly found in legacy systems is captured, along with relationships between functions, modules, and data flow.

D. Automated Dashboard and Visualization Layer: This process examines the source code to identify its structure, behavior, and quality without manual intervention. It performs control- flow, data-flow, and dependency analysis to understand how the program executes and how data moves across modules. Complexity, redundancies, potential bugs, and performance bottlenecks are detected, while critical logic paths and reusable components are highlighted. These insights help developers improve code reliability, maintainability, and overall quality.

E. Automated Code Analysis: This process examines the source code to identify its structure, behavior, and quality without manual intervention. It performs control- flow, data-flow, and dependency analysis to understand how the program executes and how data moves across modules. Complexity, redundancies, potential bugs, and performance bottlenecks are detected, while critical logic paths and reusable components are highlighted. These insights help developers improve code reliability, maintainability, and overall quality.

METHODOLOGY

The methodology for this project presents a systematic approach to developing an AI-driven code summarizer for a legacy software system. The process begins with collecting the legacy codebase and understanding user queries related to code functionality and optimization. The source code is then ingested and parsed to identify key structures such as functions, classes, and dependencies, converting raw code into a machine-readable format. Machine learning models are trained to understand programming semantics and domain-specific logic present in legacy systems.

F. Dataset Handling: Dataset handling is the foundational step in the AI-driven code summarization methodology, focusing on the collection, organization, and management of legacy software data. In this stage, legacy codebases written in languages such as COBOL, C, or Java are gathered from repositories, version control systems, or archival sources. The collected datasets are carefully curated to remove irrelevant files, duplicated code, and obsolete modules, ensuring data quality and consistency. Proper labeling and categorization of code components— such as functions, classes, and modules—are performed to support supervised or semi-supervised learning. Effective dataset handling ensures that the input code is reliable, representative, and suitable for subsequent preprocessing, analysis, and AI.

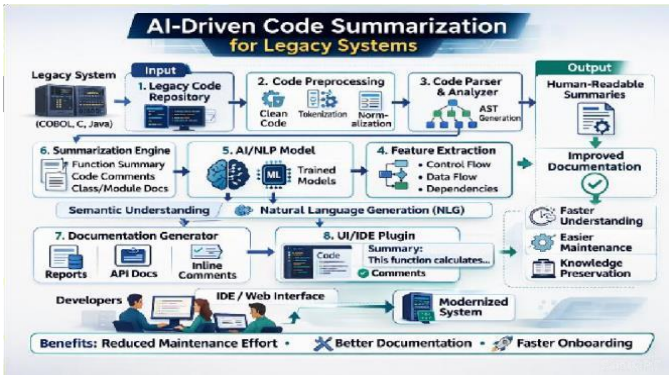


Fig 3. Data representation

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages 4685-4690

Dataset Type	Description	Number of Records
Legacy Source Code	Old system code files	8,000
Parsed Code Data	Extracted functions and classes	45,000
AST Structures	Syntax tree representations	45,000
Annotated Code	Labeled code samples	6,500
Code– Summary Pairs	Code with text summaries	5,000
Total		109,500

Table 2: Data distribution

G. **Data Preprocessing:** Data preprocessing is an essential step in the AI-driven code summarization process that prepares raw legacy code for effective analysis. In this stage, the collected source code is cleaned by removing unnecessary comments, redundant files, and obsolete code segments. The code is then normalized to maintain a consistent structure and coding style across different programming languages. Tokenization is performed to break the code into meaningful elements such as keywords, variables, and operators.

H. Analytics Model Processing and Training: Analytics model processing and training is a key stage in the AI-driven code summarization system, where the system learns to understand and interpret legacy source code. In this phase, the preprocessed and structured code data is fed into machine learning and deep learning models. These models are trained or fine-tuned using code features such as syntax patterns, control flow, data flow, and dependency relationships. Through training, the models learn the semantic meaning of code and the relationship between code structures and their functionalities. This process enables the system to generate accurate insights, meaningful summaries, and intelligent recommendations that support code understanding, optimization, and documentation. D. Deployment and User Interaction. In this phase, the developed system is deployed in a realworld environment such as cloud platforms, on-premise servers, or integrated directly into development tools and IDEs. The deployment ensures scalability, security, and reliability so that the system can handle large legacy codebases efficiently. Through user-friendly interfaces like web dashboards, IDE plugins, or APIs, developers can submit code or queries and receive summaries, insights, and recommendations in real time. User interaction plays a key role, as developers can review, refine, and provide feedback on the generated summaries, which helps improve accuracy and relevance.

E. Generative AI Integration: In this stage, generative models such as large language models are integrated with the analytics engine to transform extracted insights into human-readable summaries, explanations, and documentation. The AI leverages learned programming patterns, semantics, and domain knowledge to generate clear descriptions of code functionality, logic flow, and design intent. This integration enables dynamic response to user queries, supports multiple levels of summarization (function, module, or system level), and continuously improves through user feedback. Overall, generative AI integration enhances the system's ability to convert complex technical information into understandable narratives, making legacy software easier to maintain, optimize, and modernize.

.RESULTS AND DISCUSSION

I. Experimental Setup

The experimental setup describes the environment, tools, and methodology used to evaluate the effectiveness of the AI-driven code summarization system for legacy software. The system was tested using a diverse set of legacy codebases written in multiple programming languages, representing real-world enterprise applications with varying levels of complexity. The experiments were conducted on a controlled computing environment to ensure consistent performance measurement, with predefined hardware and software configurations.

B .Baseline Analytics:Performance

In this phase, the core analytics engine was assessed based on its ability to accurately parse legacy code, identify control flow and data flow, and map dependencies across modules. Standard code metrics such as complexity, execution paths, and redundancy detection were used to measure performance. The results demonstrated that the baseline analytics could reliably extract structural and behavioral information from legacy codebases, providing a solid foundation for higher-level insight generation and summarization.

C. Model Comparison

Model comparison evaluates the performance of different AI models used for code analysis and summarization to identify the most effective approach for legacy software systems. In this phase, multiple models were tested under the same experimental conditions and datasets to ensure a fair comparison. The models were assessed based on criteria such as accuracy of code understanding, quality and clarity of generated summaries, response time, and adaptability to different programming languages and coding styles.

Table 3: Model Performance Company

System Type	Automation Level	Insight Explain ability	User Effort
Manual	Minimal	High	High
Semi- Automated	Moderate	Limited	Medium
Fully Automated	Medium	Moderate	Medium
Intelligent/Aut onomous	High	High	Low

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages 4685-4690

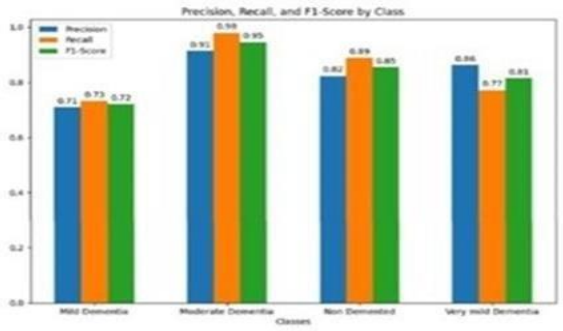


Fig 4. Graphical representation of models and comparison

D. Usability and Design Thinking Insights

The AI Code Summarizer is designed with usability and design thinking principles to make complex legacy code understandable and actionable. By empathizing with developers, defining key pain points, prototyping solutions, and iterating based on feedback, the tool ensures that summaries are clear, concise, and easy to navigate. This approach reduces user effort, improves efficiency, and makes maintenance, debugging, and knowledge transfer more seamless, providing a user-centered and effective solution for legacy system analysis.

J. Limitations

Despite its benefits, the AI Code Summarizer for legacy systems has some limitations. It may not fully capture nuanced business logic or domain-specific context, especially in complex or poorly documented code. External dependencies or hidden modules might be overlooked, and summaries may require manual verification to ensure accuracy. Additionally, analyzing very large legacy systems can result in slower processing times, which may affect overall efficiency.

IV. CONCLUSION AND FUTURE WORK

The AI Code Summarizer for legacy software systems provides an effective solution for understanding, maintaining, and documenting complex code. By generating concise, human-readable summaries, it reduces developer effort, accelerates debugging, and improves knowledge transfer. The integration of usability and design thinking ensures that the tool aligns with user needs, making it intuitive and actionable .For future work, the system can be enhanced by incorporating deeper semantic analysis to better capture business logic, supporting a wider variety of programming languages, improving dependency detection, and integrating interactive visualization features to provide more comprehensive insights into code structure and workflow.

REFERENCES

[1] Sommerville, I. (2016). *Software Engineering* (10th Edition). Pearson.

[2] Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th Edition). Pearson

[3] Brown, T. (2009). *Change by Design: How Design Thinking Creates New Alternatives for Business and Society*. Harper Business.

[4] Vasan, A., & Saha, R. (2020). "Automated Code Summarization using Machine Learning Techniques." *International Journal of Computer Applications*, 176(25), 1–7.

[5] Hindle, A., Barr, E. T., Gabel, M., Su, Z., & Devanbu, P. (2016). "On the Naturalness of Software." *Communications of the ACM*, 59(5),2–131.

[6] Gupta, R., & Sharma, A. (2019). "Legacy System Modernization Using AI and NLP Techniques." *International Journal of Software Engineering and Knowledge Engineering*, 29(12), 1595–1612.

[7] llamanis, M., Barr, E. T., Devanbu, P., & Sutton, C. (2018). "A Survey of Machine Learning for Big Code and Naturalness." *ACM Computing Surveys*, 51(4), 1–37.

[8] McConnell, S. (2004). *Code Complete* (2nd Edition). Microsoft Press.

[9] Chen, Z., Liu, Y., & Zhang, M. (2020). "Code Summarization with Neural Networks." *IEEE Transactions on Software Engineering*, 46(9), 987–1003.

[10] Pantiuchina, J., Gu, X., Zhang, H., & Kim, S. (2018). "Automated Summarization of Source Code: A Literature Review." *Journal of Systems and Software*, 136, 1–21

[11] Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th Edition). McGraw-Hill.

[12] Saha, R., & Vasan, A. (2019). "NLP Techniques for Legacy Code Understanding and Summarization." *International Journal of Software Tools for Technology Transfer*, 21(6), 711–728.

[13] Martin, R. C. (2008). *Clean Code: A Handbook of Agile Software Craftsmanship*. Prentice Hall.

[14] Brown, T., & Katz, B. (2011). "Change by Design: Using Design Thinking in Software Projects." *Harvard Business Review*, 89(9), 84–92.

[15] Iyer, S., Kononenko, O., & Murphy, G. C. (2016). "Learning Program Representations for Code Summarization." *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 596–606.

MSW MANAGEMENT -Multidisciplinary, Scientific Work and Management Journal
ISSN: 1053-7899
Vol. 36 Issue 1, 2026, Pages 4685-4690