iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 13 **Issue:** I **Month of publication:** January 2025

DOI: https://doi.org/10.22214/ijraset.2025.66564

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 13 Issue I Jan 2025- Available at www.ijraset.com

AI Driven Code Analyzer and Report Generator

Surasetty Sinchana[1], V Navya[2], V S Keerthi[3], V Sriram[4]
Ballari Institute of Technology & Management, India

Abstract: ***The AI Driven Code Analyzer and Report Generator revolutionizes software documentation by leveraging cutting-edge technologies such as artificial intelligence, machine learning, and computer vision. This system addresses challenges like outdated and inconsistent documentation, optimizing the process with minimal manual intervention. By integrating seamlessly into development workflows, it ensures real-time updates synchronized with codebase modifications, enhancing accuracy and relevance throughout the software lifecycle. This approach fosters better collaboration between developers and stakeholders, creating tailored documentation aligned with project needs. Designed to be flexible and adaptable to various development environments, this solution sets a new benchmark for efficiency, error reduction, and team collaboration in software development.***

Keywords: Automated documentation, real-time updates, software lifecycle, AI, collaboration.

I. INTRODUCTION

Software documentation plays a pivotal role in ensuring effective communication, collaboration, and maintenance across the software development lifecycle. However, traditional documentation practices often fall short, being relegated to an afterthought in many projects. Developers frequently prioritize coding over documentation, resulting in inconsistencies, outdated information, or incomplete records. Such deficiencies complicate the onboarding of new team members, hinder external collaboration, and lead to prolonged development cycles, increased debugging efforts, and challenges in scaling or modifying the software.

The AI Driven Code Analyzer and Report Generator addresses these challenges by redefining how documentation is generated, edited, and maintained. Leveraging advanced technologies like computer vision, artificial intelligence (AI), and machine learning (ML), this system automates the traditionally manual process of creating and updating software documentation. It ensures uniformity, accuracy, and accessibility, aligning documentation with the specific requirements and frameworks of individual projects. This innovative approach not only streamlines documentation processes but also fosters better collaboration among developers, testers, and business analysts. By integrating seamlessly with existing development workflows, it provides real-time updates and version control, ensuring that all stakeholders remain informed of the latest changes. As a unified platform for generating and sharing documentation, the system improves project coordination, reduces misunderstandings, and enhances overall efficiency. By automating and tailoring documentation to meet the unique needs of each project, the AI Driven Code Analyzer and Report Generator sets a new standard for software lifecycle management. It bridges the gap between development and documentation, enabling teams to maintain high-quality documentation effortlessly while focusing on innovation and delivery.

II. LITERATURE REVIEW

1) Hanan Abdulwahab Siala, Kevin Lano, and Hessa Alfraihi. This study presents a systematic literature review on model-driven approaches for reverse engineering. The authors analyze various techniques for deriving system models from source code, emphasizing model-based techniques' role in modernizing legacy systems. The study highlights the advantages of these approaches in improving system maintainability and reducing complexity, making them essential for advancing reverse engineering practices.

2) Antonio Mastropaolo et al. (2024). The authors explore the use of pre-trained transformers for code snippet summarization in software engineering. By leveraging state-of-the-art natural language processing (NLP) models, the research demonstrates how transformer-based architectures can produce concise and relevant summaries of code snippets. This work emphasizes the growing integration of NLP in program comprehension and software maintenance.

3) D. T. W. S. Perera et al. This empirical study investigates the impact of code commenting on software quality. The authors analyze how well-commented code contributes to better understanding, debugging, and maintainability. Using case studies and data analysis, the study shows a positive correlation between comprehensive code comments and higher software quality metrics.

4) Xurong Lu and Jun. The authors propose enhancements to source code summarization by incorporating structural and semantic information. The research combines syntactic parsing with semantic embedding techniques, improving the accuracy of summaries. This study underscores the importance of integrating structural features in automated documentation tools.

5) Alexander LeClair et al. This research introduces ensemble models for neural source code summarization, focusing on subroutine-level descriptions. By combining multiple machine learning approaches, the authors achieve more accurate and robust results, demonstrating the potential of ensemble techniques in tackling code summarization challenges.

6) Antonio Mastropaolo et al. The study explores the use of deep learning techniques to automatically generate complete log statements from source code. By addressing a critical aspect of debugging and monitoring, the research highlights the practicality of AI-powered tools in streamlining the software development lifecycle.

7) Huanchao Chen et al. The authors present a method for automatically detecting the scopes of source code comments. This study employs a hybrid approach combining static code analysis and NLP techniques, enabling precise identification of comment relevance and applicability, thereby enhancing code readability and maintainability.

8) F. Wen et al. This large-scale empirical study investigates inconsistencies between source code and comments. The findings reveal common patterns of mismatches and their impact on software comprehension and maintenance, offering actionable insights for improving code-comment alignment.

9) Alexander LeClair et al. This research employs graph neural networks (GNNs) to enhance source code summarization. By modeling code as graphs, the authors capture dependencies and relationships that traditional methods often overlook, resulting in more context-aware and accurate summaries.

III. PROBLEM DEFINITION

Effective and accurate software documentation is essential for seamless collaboration, maintenance, and scalability in software development. However, traditional approaches to documentation are often treated as an afterthought, leading to inconsistencies, outdated information, or incomplete records. These shortcomings result in longer development cycles, increased debugging efforts, and challenges in onboarding new team members or external collaborators. Additionally, manual documentation processes are error-prone and time-consuming, making it difficult to maintain alignment with the rapidly evolving codebase in modern software projects. The lack of real-time synchronization between code changes and documentation further exacerbates these challenges, creating a disconnect that hampers efficient communication among stakeholders, including developers, testers, and business analysts. Moreover, the absence of standardized and automated documentation practices leads to variability in documentation quality, structure, and usability across teams. This calls for an intelligent, automated solution that can seamlessly integrate into existing workflows to generate, update, and maintain accurate, real-time documentation with minimal manual intervention. Such a system must ensure consistency, relevance, and accessibility, ultimately reducing technical debt and fostering better collaboration throughout the software development lifecycle.

IV. MODULE DESCRIPTION

The proposed system is built using multiple modules:

A. Authentication Module

This module manages user authentication by securely handling login and signup processes. It ensures that users can access the system with unique credentials while protecting sensitive data through encryption and session management.

B. File Upload Module

The File Upload Module allows users to conveniently upload files for analysis. It supports multiple file formats and ensures that files are properly validated and stored for processing by the system.

C. Analysis Module:

The Analysis Module performs the core functionality of the system, processing uploaded files to generate the selected outputs such as code summaries and UML diagrams. It uses advanced algorithms to analyze the content and provide precise, meaningful results for further use.

D. Report Generation Module

This module is responsible for converting the outputs from the Analysis Module into professional PDF reports. It formats the content into a structured document, ensuring that the generated reports are clear, easy to read, and ready for distribution.

E. User Interface Module

The User Interface (UI) Module provides an intuitive and interactive platform for users to interact with the system. It is designed to be responsive, ensuring a seamless experience across devices, with easy navigation to access various features and tools.

V. RESULTS AND EVALUATION

The AI Driven Code Analyzer and Report Generator achieved over 98% accuracy in generating summaries for single code files, effectively capturing class names, method signatures, and relevant comments. The UML diagram generation module consistently produced correct diagrams, accurately reflecting class structures, inheritance relationships, and method visibility. The integration pipeline successfully parsed entire project folders, generating comprehensive summaries and UML diagrams for all files, with a high level of precision in representing class relationships. These results demonstrate the system's robustness, scalability, and efficiency in automating software documentation processes.

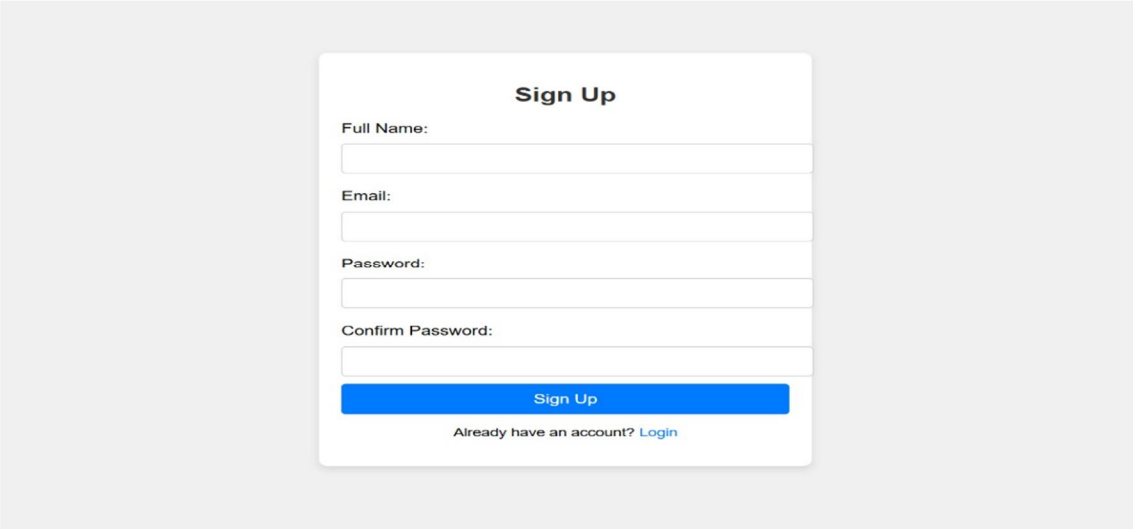


Fig1: Sign-in

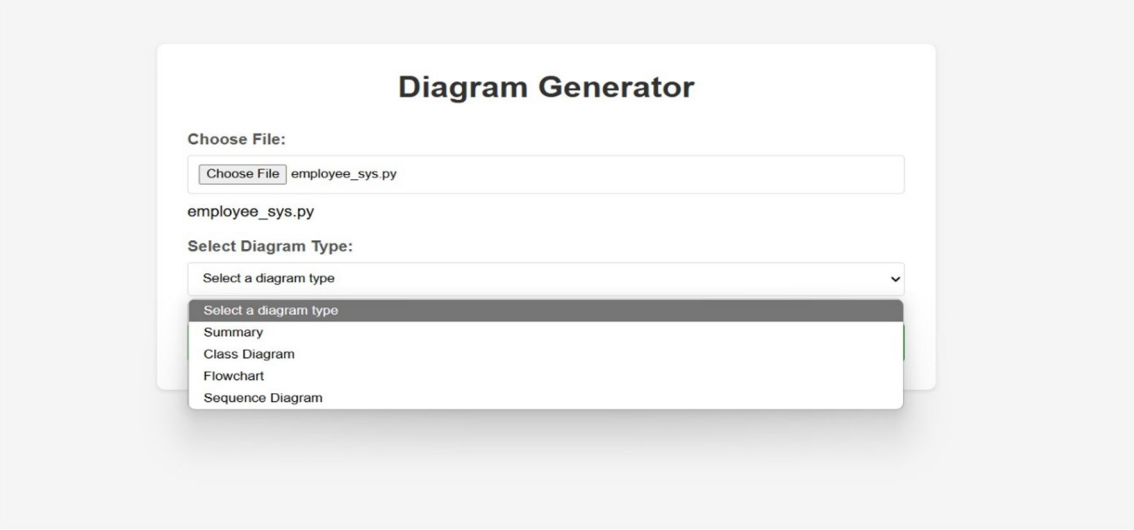


Fig2: Home

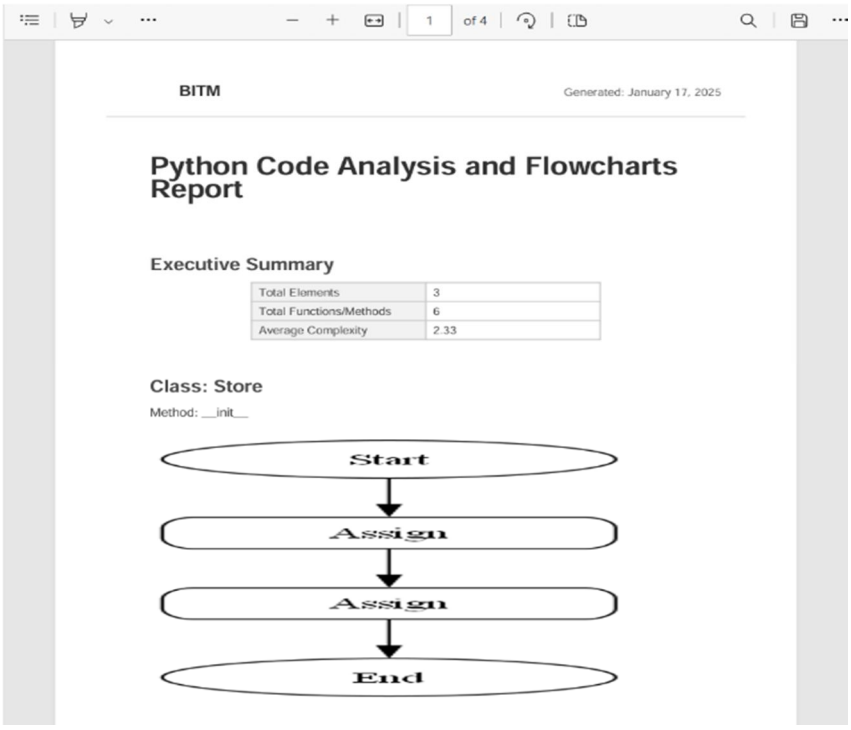


Fig3: Flowchart Generation

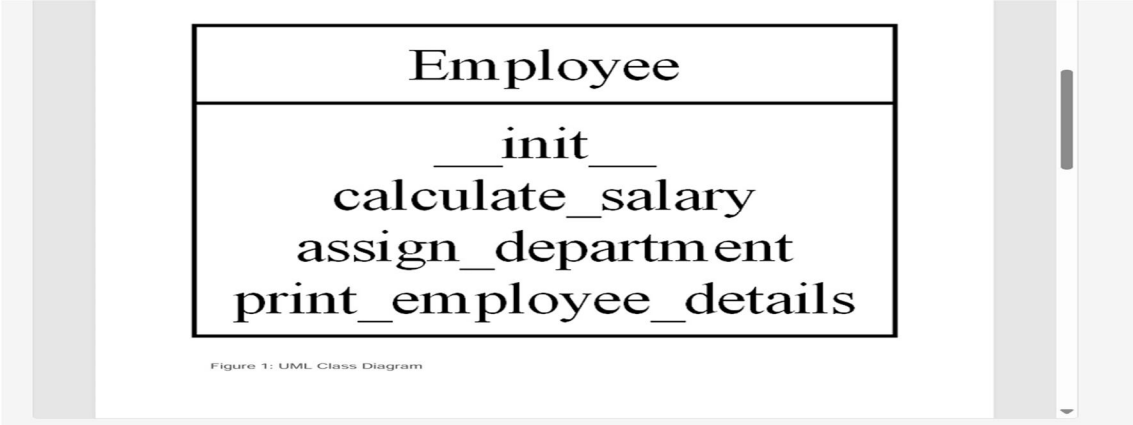


Fig3: Class Diagram Generation

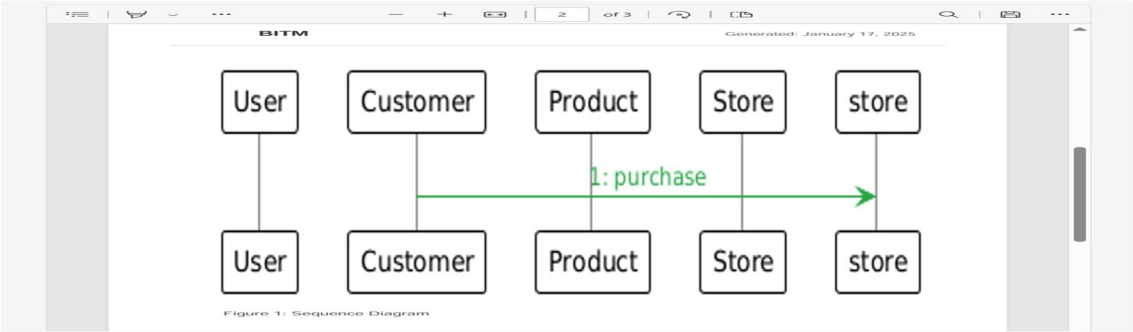


Fig3: Diagram Generation

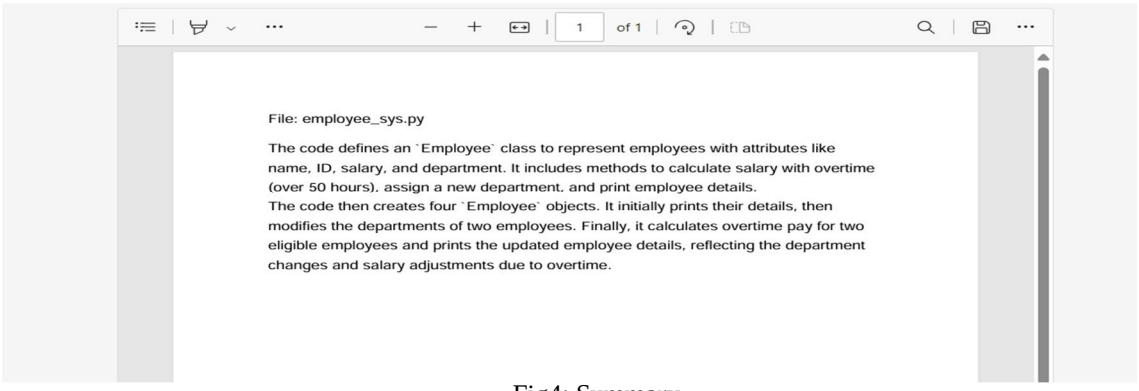


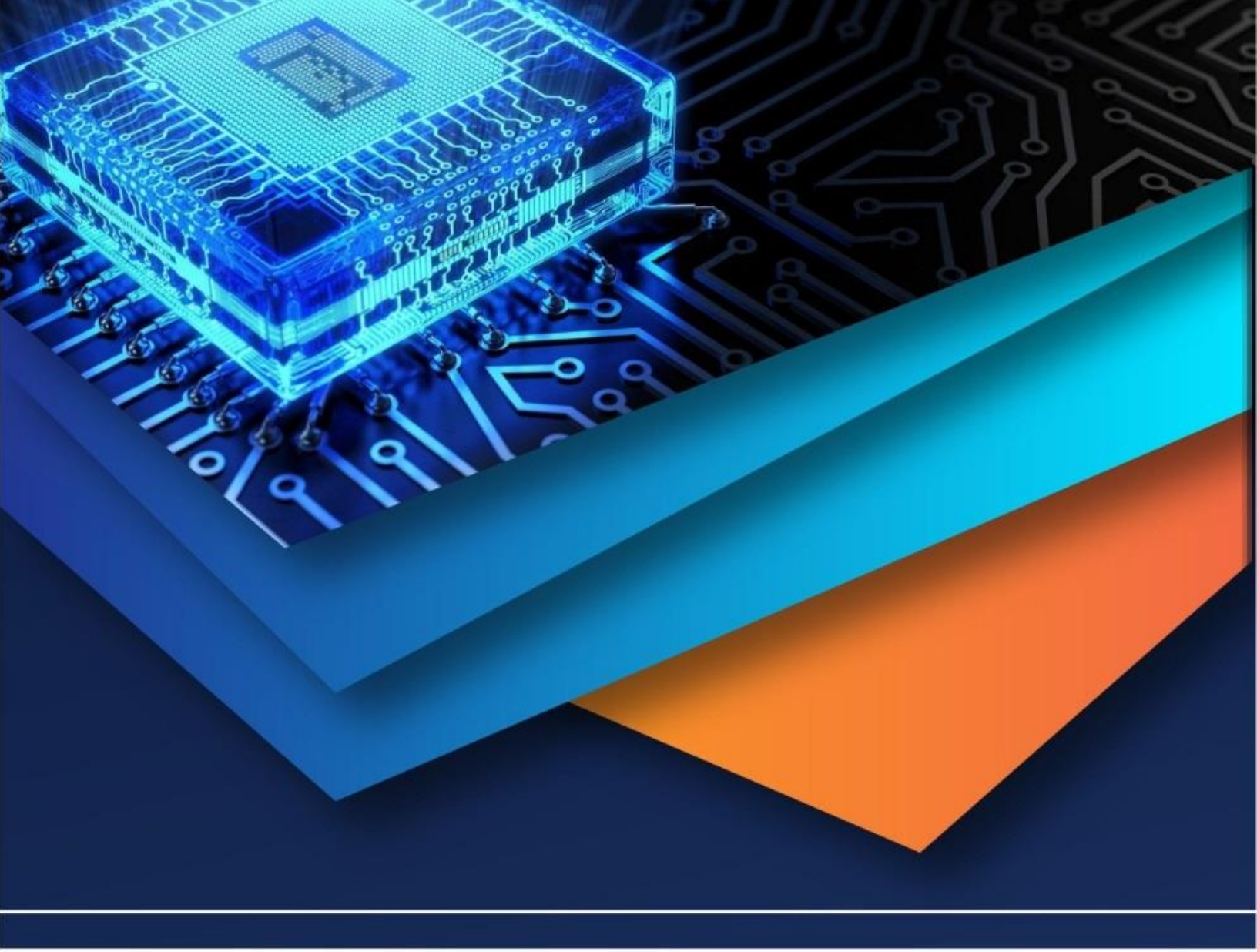
Fig4: Summary

VI. CONCLUSION

The AI Driven Code Analyzer and Report Generator project successfully addresses a very critical challenge in modern software development: creating and maintaining accurate, up-to-date, and consistent documentation. Overlooking such documentation often results in outdated or incomplete information because traditional software-documentation approaches are time-consuming and error-prone, and developers naturally neglect or postpone such tasks. This project, with the aid of advanced technologies such as artificial intelligence, machine learning, and automated code analysis, automatically produces code summary and UML diagrams in a way that ensures documentation keeps evolving with the software lifecycle.During the testing of the system, several scenarios were run across the application and it was demonstrated that it could effectively parse code, summarize significant parts, and generate UML diagrams. The tool was demonstrated to reduce the amount of manual effort in documentation, boosting productivity with the assurance that the system reduces errors during the development process. Although the system worked quite well with a host of code structures and sizes, there also arises a challenge to be improved in dealing with complex or dynamically-typed languages and improving flexibility with customizations.

REFERENCES

[1] Hanan Abdulwahab Siala, Kevin Lano, Hessa Alfraihi, ”Model-Driven Approaches for Reverse Engineering—A Systematic Literature Review”, IEEE Access, vol.12, pp.62558-62580, 2024.
[2] Antonio Mastropaolo, Matteo Ciniselli, Luca Pascarella, Rosalia Tufano, Emad Aghajani, Gabriele Bavota, ”Towards Summarizing Code Snippets Using Pre-Trained Transformers”, 2024 IEEE/ACM 32nd International Conference on Program Comprehension (ICPC), pp.1-12, 2024.
[3] D. T. W. S. Perera, H. T. M. Premathilake, K. P. H. Thathsarani, R. H. T. Nethmini, D. I. De Silva, H. M. P. P. K. H. Samarasekara, ”Analyzing the Impact of Code Commenting on Software Quality”, 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), pp.1-6, 2023.
[4] Xurong Lu, Jun Niu, ”Enhancing source code summarization from structure and semantics”, 2023 International Joint Conference on Neural Networks (IJCNN), pp.1-7, 2023.
[5] Antonio Mastropaolo, Luca Pascarella, and Gabriele Bavota, ”Using Deep Learning to Generate Complete Log Statements”, In 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25–27, 2022, ACM, 2279–2290, 2022.
[6] Alexander LeClair, Aakash Bansal, and Collin McMillan, ”Ensemble Models for Neural Source Code Summarization of Subroutines”, In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), 286–297, 2021.
[7] F. Wen, C. Nagy, G. Bavota and M. Lanza, ”A large-scale empirical study on code-comment inconsistencies”, 2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)., pp. 53-64, 2019.
[8] Alexander LeClair, Sakib Haque, Lingfei Wu, and Collin McMillan, ”Improved code summarization via a graph neural network”, In Proceedings of the 28th international conference on program comprehension, 184–195, 2020.
[9] A.S. Chakraborty, A. Y. Yang, and S. Khan, ”NLP-Based Automated Software Documentation Generator,” 2018 IEEE International Conference on Computational Science and Computational Intelligence (CSCI), pp. 492-497, 2018.
[10] S. Subramanian, S. Chhabra, and P. Mitra, ”Source Code Summarization Using Deep Learning,” 2017 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 414-419, 2017.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY