International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 13 Issue: 11 | Nov 2026 www.irjet.net p-ISSN: 2395-0072

Intelligent GitHub Repository Analyzer

[1]Sure Mamatha, [2]B. Aniket , [3]B. Pranai Raghav Reddy , [4]B. Madhu Lasya , [5]C. Deepika

[12345]Department of Computer Science and Engineering, Hyderabad Institute of Technology and Management, Hyderabad, Telangana State, India.

Abstract - *The Intelligent GitHub Repository Analyzer with AI Assistant helps users understand GitHub repositories by extracting available information, examining files and directories, and generating AI-assisted summaries and recommendations. The application accepts a repository URL, retrieves repository details and documentation, and processes the available information through a structured workflow. Python and FastAPI support backend processing, while Streamlit provides the user interface. The Gemini API generates summaries and recommendations based on the extracted context. ReportLab supports PDF report generation, and Matplotlib can provide visualizations where implemented. The system is intended to help students, beginners, and developers explore unfamiliar repositories with less manual effort. Its main components include repository input, data extraction, structural analysis, AI-assisted summarization, results display, and PDF reporting. Validation and error handling help manage invalid URLs, inaccessible repositories, and API failures. Future improvements can include deeper code analysis, dependency detection, security checks, repository comparison, and repository history analysis.*

Keywords - GitHub Repository Analysis, Artificial Intelligence, Gemini API, Python, FastAPI, Streamlit, Code Summarization, PDF Report Generation.

I. INTRODUCTION

GitHub repositories often contain multiple directories, source-code files, configuration files, dependencies, and documentation. Understanding an unfamiliar repository can be time-consuming, especially for beginners who must manually inspect files and read project documentation to identify its purpose and structure. This process becomes more difficult when repositories are large or their documentation is incomplete.

Repository analysis techniques make it possible to collect project information and organize it into a structured overview. AI-based language models can further assist by generating summaries and recommendations from the available repository information. A repository analyzer can combine these capabilities to make project exploration more convenient.

The proposed Intelligent GitHub Repository Analyzer with AI Assistant brings repository data extraction, structural analysis, AI-generated summaries, and report generation into a single application. It uses Python and FastAPI for backend processing, Streamlit for the user interface, and the Gemini API for AI-assisted analysis.

The main contributions of the proposed system are:

1. A unified interface for analyzing GitHub repositories.
2. Extraction of repository information and directory structure.
3. Analysis of available repository files and project documentation.
4. Integration with the Gemini API for summaries and recommendations.
5. Display of analysis results through a Streamlit interface.
6. Generation of downloadable PDF reports.
7. Input validation and error handling for unsuccessful requests.

II. RELATED WORK

A. Software Repository Mining

Software repository mining involves collecting and analyzing information from software projects to understand their structure, evolution, and development practices. Repository files, documentation, and metadata provide useful information about the purpose and organization of a project. The proposed system applies these ideas by extracting available GitHub repository information and organizing it for easier understanding.

B. AI-Based Code Understanding

Pre-trained language models such as CodeBERT, GraphCodeBERT, and CodeT5 demonstrate the use of machine learning techniques for programming-language understanding and code-related tasks. AI-based models can help interpret technical information and produce natural-language explanations. The proposed application uses the Gemini API to generate summaries and recommendations from the repository information supplied to it.

C. Web Frameworks and Report Generation

Web frameworks support the development of applications that separate user interaction from backend processing. FastAPI provides an interface for backend operations, while Streamlit supports interactive presentation of analysis results. Report-generation libraries such as ReportLab can create downloadable documents, and Matplotlib can produce visual representations of supported analysis data. Combining these technologies provides a practical approach to presenting repository analysis results.

D. Research Gap

Understanding a GitHub repository may require users to inspect its files, read documentation, identify its directory structure, and determine the purpose of different components. These steps can be challenging for users who are unfamiliar with the project or its programming language.

Although repository-hosting platforms provide source files and project metadata, users may still need to perform considerable manual work to develop an overall understanding. AI-based summarization can help reduce this effort by presenting available information in natural language.

The proposed application addresses this need by combining repository data extraction, structural analysis, AI-assisted summaries, and PDF report generation in one workflow.

The proposed system provides a centralized workflow containing:

- GitHub repository URL input.
- Retrieval of available repository information.
- Extraction of directory and file details.
- Analysis of README content and other available information.
- AI-generated summaries and recommendations.
- Presentation of results through Streamlit.
- PDF report generation and error handling.

III. METHODOLOGY

A. System Design

The proposed application follows a modular architecture in which the frontend collects the repository URL and displays the analysis results, while the backend manages repository processing and AI integration. The system retrieves available repository information, examines its structure and relevant files, and prepares the extracted information for further analysis.

The Gemini API processes the provided repository context to generate a natural-language summary and recommendations. The results are returned to the interface, where users can review the analysis. ReportLab is used for PDF report generation, while Matplotlib can support visual representations where required.

Design validation: The application should validate the submitted repository URL, handle unavailable or inaccessible repositories, and display clear messages when extraction or AI requests fail.

The overall architecture is:

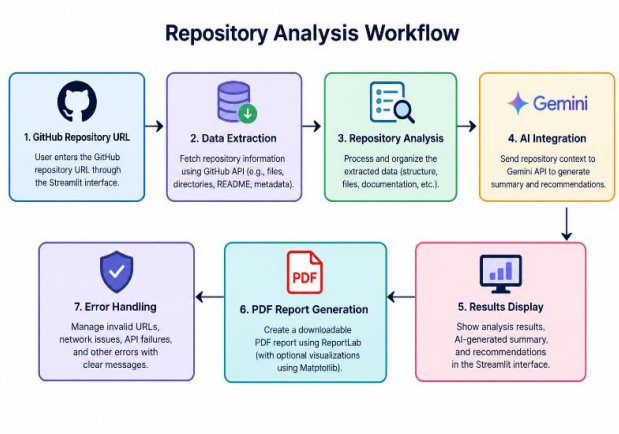

Fig. 1. Repository Analysis Workflow

B. System Modules

1. Repository Input

The input module allows users to provide a GitHub repository URL through the Streamlit interface. The URL is checked before processing to reduce errors caused by empty or incorrectly formatted input. Repository accessibility may affect whether the system can retrieve the required information.

2. Repository Data Extraction

The extraction module retrieves available repository information, including directory structure, file names, README content, and other relevant files supported by the implementation. The extracted information provides the foundation for subsequent analysis.

3. Repository Analysis

The analysis module organizes the retrieved information into a structured representation. It identifies relevant repository details and prepares the available context for AI processing. The quality of the analysis depends on repository accessibility and the information successfully extracted.

The analysis can include:

- Repository details.
- Directory and file structure.
- README and documentation content.
- Relevant source-file information.
- Extracted information for AI processing.

4. AI Summary Generation

The AI module sends the prepared repository context to the Gemini API. The API generates a natural-language summary and recommendations based on the information supplied. The generated response is returned to the application for presentation. Its accuracy depends on the quality and completeness of the provided context.

5. Results Display

The results module presents the extracted repository information, AI-generated summary, and recommendations through the Streamlit interface. The information is organized to help users understand the repository without manually inspecting every available file.

6. PDF Report Generation

The reporting module generates a downloadable PDF report using ReportLab. The report can contain the available repository details, analysis summary, and recommendations. Matplotlib may be used to create visual representations of supported analysis data.

7. API Integration

The API integration module coordinates communication between the backend and external services. FastAPI supports backend request handling, while the Gemini API provides AI-assisted summarization. The application processes responses and handles unsuccessful requests. API credentials should be stored securely rather than exposed in the frontend.

8. Error Handling

The error-handling module manages invalid repository URLs, inaccessible repositories, missing information, network problems, and unsuccessful API responses. Clear messages help users identify issues and retry requests when appropriate.

9. Repository Analysis Workflow

The system processes repository information through a sequence of stages. The general workflow is:

GitHub URL → Data Extraction → Repository Analysis → Gemini API → Summary and Recommendations → Streamlit Display / PDF Report.

The actual output depends on the repository's contents, accessibility, API availability, and the quality of the extracted information.

IV. IMPLEMENTATION

A. Frontend

The frontend is developed using Streamlit and provides the interface for entering a GitHub repository URL, initiating analysis, and viewing the results. It displays the extracted repository information and AI-generated summary and recommendations. The interface also provides access to the PDF report when report generation succeeds.

The frontend organizes information so that repository details and AI-generated explanations remain easy to read. Appropriate status messages can indicate processing progress and communicate errors encountered during analysis.

B. Backend

The backend is developed using Python and FastAPI. It coordinates repository processing, data extraction, and communication with the Gemini API. The backend validates inputs, processes repository information, handles API responses, and returns analysis results to the frontend.

A repository-analysis request conceptually includes:

POST /analyze

for submitting repository information for analysis, if this endpoint is implemented.

The response can contain extracted repository details, an AI-generated summary, recommendations, and processing status. The actual request and response structure depends on the implemented API.

C. Database

The application can process repository information directly without requiring a database if persistent storage is not part of the implementation. Repository details and analysis results can be held during the request and used to generate the displayed output and PDF report.

If persistent analysis history is added in future development, a record could contain:

- Repository URL.
- Repository metadata.
- Extracted file information.
- Generated summary.
- Recommendations.
- Report status.
- Analysis timestamp.

Any stored repository information should be handled with appropriate access and security safeguards. Database storage and analysis history should be described as implemented features only if they are actually included in the submitted prototype.

V. RESULTS

The proposed application can be evaluated using accessible GitHub repositories with different directory structures and documentation formats. Functional testing should cover repository URL input, data extraction, structural analysis, Gemini API integration, results display, and PDF report generation.

Testing should also include invalid URLs, inaccessible repositories, incomplete repository information, network failures, and unsuccessful API responses. AI-generated summaries should be checked against the extracted repository content to assess whether the descriptions are relevant and supported by the available information.

TABLE I FUNCTIONAL TESTING OF THE PROPOSED SYSTEM

Module	Test Case	Expected Result	Actual Result
Repository Input	Enter a valid GitHub URL	URL is accepted	URL accepted successfully
Data Extraction	Retrieve repository data	Repository information is extracted	Information retrieved successfully
Structure Analysis	Analyze files and directories	Repository structure is organized	Structure displayed successfully
AI Integration	Generate summary and recommendations	AI generates a summary	Summary received successfully
Results Display	Display analysis output	Results are shown to the user	Results displayed successfully
PDF Generation	Generate a PDF report	PDF report is created	Report generated successfully
Error Handling	Enter an invalid URL	Appropriate error message is displayed	Error message displayed

Table I summarizes the core functional checks. Successful results should be recorded only for features that have actually been tested and verified in the submitted prototype. Summary quality and processing time can vary according to repository size, available documentation, network conditions, and external API responses.

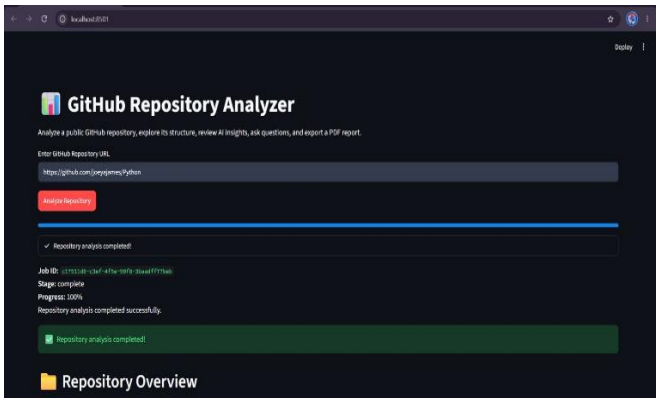

Fig. 2. Application Dashboard



Fig. 3. Repository Analysis PDF Report

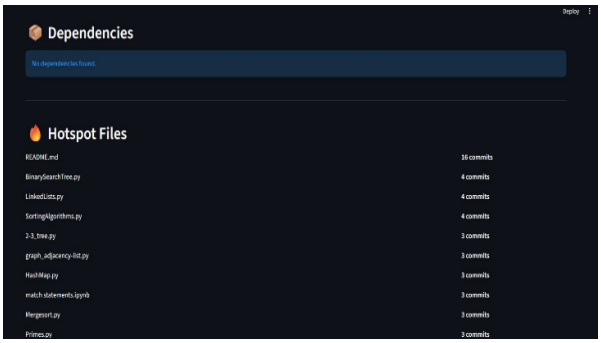

Fig. 4. Repository Structure Visualization

The Results Dashboard displays the analysis results of the selected GitHub repository in an organized format. It provides information about the repository structure, programming languages, commits, contributors, and dependencies. The PDF report option allows users to download and save the analysis results for future reference. These features help users understand the repository easily and quickly.

VI. DISCUSSION

The proposed application combines repository data extraction, structural analysis, AI-assisted summarization, and report generation in one workflow. This can reduce the effort required to understand unfamiliar GitHub repositories and make project information more accessible to students, beginners, and developers.

The quality of the output depends on the information available in the repository, the success of data extraction, and the response generated by the Gemini API. Incomplete documentation or inaccessible files may limit the usefulness of the summary. AI-generated recommendations may also require manual review before being used for technical decisions.

The system provides a centralized interface through Streamlit and uses FastAPI to organize backend processing. PDF report generation allows the analysis results to be saved and reviewed outside the application. These features support a more convenient workflow for exploring repository information.

However, the application should not be considered a complete substitute for detailed code review, execution-based testing, or professional security analysis. Its conclusions are limited by the repository information it processes and the capabilities of the integrated AI service.

Future improvements can include deeper source-code analysis, dependency identification, automated code-quality checks, security analysis, repository comparisons, and analysis of commit history. These additions could expand the system from a repository overview tool into a more comprehensive software analysis assistant.

VII. CONCLUSION

The Intelligent GitHub Repository Analyzer with AI Assistant provides a unified workflow for extracting and understanding GitHub repository information. Its main stages include repository input, file and directory extraction, structural analysis, Gemini API integration, presentation of AI-generated summaries and recommendations, and PDF report generation.

The system combines Python, FastAPI, Streamlit, the Gemini API, ReportLab, and Matplotlib to support repository analysis and result presentation. By organizing available repository information into a structured overview, the application can reduce manual effort and help users understand unfamiliar projects more conveniently.

The modular design supports the separation of data extraction, AI processing, interface presentation, and report generation. The usefulness of the output depends on repository accessibility, the quality of extracted information, and the accuracy of AI-generated responses.

Future work can include advanced code-quality analysis, dependency detection, security checks, support for more programming languages, repository comparison, commit-history analysis, and improved reporting. Evaluation with a wider range of repositories can further help assess the reliability and usefulness of the system.

ACKNOWLEDGMENT

The successful completion of the Intelligent GitHub Repository Analyzer with AI Assistant would not have been possible without the guidance, support, and encouragement of many people.

We express our sincere gratitude to our project guide for valuable guidance, continuous support, and constructive suggestions throughout the development of this project.

We are grateful to the Head of the Department for providing the necessary facilities, resources, and encouragement required to carry out this work. We would also like to thank the Principal and faculty members of our institution for their support and for providing a suitable academic environment.

We extend our sincere thanks to our friends and classmates for their suggestions and technical assistance during the development and testing of the system.

Finally, we express our gratitude to our parents and family members for their constant support, motivation, and encouragement throughout the project. We are thankful to everyone who directly or indirectly contributed to the development of this application.

REFERENCES

[1] Kagdi, H., Collard, M. L., and Maletic, J. I., "A Survey and Taxonomy of Approaches for Mining Software Repositories in the Context of Software Evolution," Journal of Software Maintenance and Evolution: Research and Practice, 2007. https://doi.org/10.1002/smr.344

[2] Feng, Z., et al., "CodeBERT: A Pre-Trained Model for Programming and Natural Languages," Findings of EMNLP, 2020. https://aclanthology.org/2020.findings-emnlp.139/

[3] Guo, D., et al., "GraphCodeBERT: Pre-training Code Representations with Data Flow," International Conference on Learning Representations, 2021. https://arxiv.org/abs/2009.08366

[4] Wang, Y., et al., "CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation," EMNLP, 2021. https://aclanthology.org/2021.emnlp-main.685/

[5] Zhang, J., et al., "A Survey of Automatic Source Code Summarization," Symmetry, 2022. https://doi.org/10.3390/sym14030471

[6] Vidoni, M., "A Systematic Process for Mining Software Repositories: Results from a Systematic Literature Review," Information and Software Technology, 2022. https://doi.org/10.1016/j.infsof.2021.106791

[7] GitHub, "GitHub REST API Documentation," GitHub Developer Documentation. https://docs.github.com/en/rest

[8] Google, "Gemini API Documentation," Google AI for Developers. https://ai.google.dev/gemini-api/docs

[9] FastAPI, "FastAPI Documentation." https://fastapi.tiangolo.com/

[10] Streamlit, "Streamlit Documentation." https://docs.streamlit.io/

AUTHOR BIOGRAPHIES

First Author

C. Deepika is a third-year student in the Department of Computer Science and Engineering at Hyderabad Institute of Technology and Management (HITAM), Hyderabad, Telangana. Her project contribution focuses on data extraction, processing, and API integration for the Intelligent GitHub Repository Analyzer with AI Assistant. She is interested in learning new technologies and developing software applications that simplify repository analysis and improve the understanding of code structure.