DocuCore: Text & Code Summariser In Cloud

Garvita Sakhrani[1], Hiteshi Agrawal[2], Abhishek Gupta[3],
Mr. Praveen Kumar Yadav[4]
Department of Information Technology
Swami Keshvanand Institute of Technology, Management & Gramothan (SKIT)
Jaipur, Rajasthan, India

ABSTRACT

This project presents DocuCore — a cloud-based client-server system for automated document summarization and source code analysis. Leveraging Large Language Models (LLMs) for natural language processing and static analysis tools for code quality evaluation, the system enables users to upload files and receive structured insights in real-time. The solution is built on Microsoft Azure services, using Azure DevOps pipelines, a self-hosted agent, and Databricks to ensure scalable, efficient, and secure deployment. A user-friendly web interface facilitates file uploads and displays summarised results or code analysis reports. DocuCore significantly reduces manual effort, improves developer productivity, and enhances understanding of large technical documents and codebases.

I. INTRODUCTION

Large document management and complicated code analysis can be difficult and time-consuming in today's fast-paced world. Traditional approaches frequently lack the speed and efficiency needed to summarise large research publications or find problems in software code. This project presents a cloud-based system that makes advantage of cutting-edge technology like Large

Code analysis tools and Language Models (LLMs). The solution is made to assist users save time and concentrate on more important activities by making code analysis and document summarisation quicker and simpler. This cloud-based system represents a significant advancement in the way we approach document summarization and code analysis. By combining the power of Large Language Models (LLMs) and automated code inspection tools with the scalability and flexibility of the AWS cloud, the system offers an innovative solution to two of the most common and challenging problems in research and software development. In doing so, it not only makes these tasks more efficient but also frees up valuable time for users to focus on higher-level tasks, ultimately driving greater productivity, accuracy, and innovation.

II. LITERATURE REVIEW

Recent advances in artificial intelligence and cloud computing have enabled automation of tasks such as document summarization and code analysis. Tools like GPT-3.5, T5, and other Large Language Models (LLMs) have demonstrated strong performance in natural language understanding and summarization tasks [1]. According to Basyal's (2023) comparative study, LLMs such as MPT-7b-instruct, Falcon-7b-instruct, and OpenAI ChatGPT (text-davinci-003) exhibit varying capabilities in summarizing text, with ChatGPT outperforming others in terms of standard NLP metrics like BLEU, ROUGE, and BERT Scores [8]. This highlights the potential of LLMs in document summarization tasks, which is critical for our system.

Similarly, static code analysis tools like SonarQube and PyLint offer reliable mechanisms for detecting code quality issues and vulnerabilities [4]. However, recent research by Fang et al. (2024) suggests that LLMs may also play a crucial role in improving code analysis by providing context-aware suggestions and automated problem detection, bridging the gap between traditional static

analysis and AI-based methods [9]. While these tools offer great capabilities individually, they often operate in isolation. Existing systems lack a unified approach to process documents and source code together, particularly in cloud-based environments. This project addresses the gap by integrating LLMs and static analysis tools within a cloud-native DevOps pipeline using Azure Databricks, Azure DevOps, and Microsoft Self-Hosted Agents, making the entire process seamless and scalable.

III. Proposed Methodology

The development of DocuCore follows a structured and modular methodology combining traditional engineering practices with modern cloud-native DevOps principles. The system development life cycle was divided into well-defined phases to ensure systematic progression from requirements to deployment and feedback.

1. Requirement Analysis and Design

- Define functional and non-functional requirements.
- Identify system goals: summarisation, static analysis, real-time feedback.
- Deliverables: System requirement specifications and architectural overview.

2. Technology Stack Tool Selection

- Azure DevOps for CI/CD automation.
- Microsoft Self-Hosted Agent for deployment control.
- Azure Blob Storage for secure file storage.
- Azure Databricks for scalable ML code analysis execution.

3. Component Development

- Develop individual modules for summarisation, code analysis, and frontend interface.
- Backend logic implemented using Python/Flask.
- CI/CD pipeline defined in Azure YAML.

4. Integration and Automation

- All services integrated using REST APIs.
- Pipelines auto-deploy code to Databricks using SHA.

5. Testing and Validation

- Perform functionality, load, and usability testing.
- Evaluate model output accuracy and code analysis precision.

6. Feedback and Iterative Refinement

- Collect faculty/peer feedback and refine UX, models, and pipelines.

Deliverables

- Modular, scalable cloud-based application.
- Fully integrated DevOps workflow.

Conclusion

This methodology focuses on a step-by-step process to build a scalable, reliable, and user-friendly client-server system using Azure infrastructure. By incorporating cutting-edge technologies such as Large Language Models for summarization and specialized code analysis tools, the system delivers faster, more efficient document processing and code quality evaluation. Leveraging Azure services ensures that the system remains both scalable and resilient under varied workloads.

IV. ALGORITHM DESCRIPTION

The proposed system involves two key functionalities: Document Summarization and Code Analysis. Each functionality relies on a combination of sophisticated algorithms and cloud infrastructure to efficiently process user inputs and deliver actionable insights. Here's a detailed algorithmic breakdown of how the system operates:

A. Document Summarization Algorithm

The document summarization process is automated through an integrated cloud-based CI/CD pipeline. This pipeline leverages Azure DevOps, Databricks, and a Microsoft Self-Hosted Agent to ensure secure, scalable, and real-time summarisation using Large Language Models (LLMs).

Algorithm Steps:

1) **User Upload:**
   - The user uploads a PDF document via the chatbot interface deployed on Databricks.
   - The upload triggers a backend function call for preprocessing and summarisation.
2) **CI/CD Orchestration via Azure DevOps:**
   - The pipeline is defined in a YAML file and managed through Azure Repos.
   - The Microsoft Self-Hosted Agent picks the pipeline job and deploys the latest summarisation logic to Databricks.
   - The summarisation app is automatically updated and restarted.
3) **Preprocessing:**
   - The uploaded document is cleaned to remove unwanted formatting, symbols, and metadata.
   - The text is segmented into smaller parts (paragraphs or sentences) for efficient summarisation.
4) **Model Inference (LLM Execution on Databricks):**
   - Azure DevOps pipeline deploys the summarisation logic to a Databricks notebook.
   - A pre-trained LLM such as GPT-3.5 or T5 is used for generating an abstractive summary.
   - The model runs within a Databricks cluster optimized for parallel processing.
5) **Postprocessing and Storage:**
   - The raw summary is cleaned, checked for redundancy, and formatted.
   - Final output is stored in Azure Blob Storage for retrieval.
6) **Result Delivery:**
   - The summarised text is displayed on the chatbot interface.
   - Users have the option to copy the results.

B. Code-Based Analysis Algorithm

The code-based analysis module is designed to automatically evaluate source code for quality, security, and maintainability using static analysis techniques. This process is fully automated through a CI/CD pipeline orchestrated via Azure DevOps and executed using a Microsoft Self-Hosted Agent, with deployments targeted to a Databricks-hosted app.

Algorithm Steps:

1) **User Upload:**
   - The user uploads source code files (e.g., `.py`, `.java`, `.cpp`) via the chatbot interface hosted on Databricks.
   - The upload triggers a backend event to begin the static analysis process.
2) **CI/CD Pipeline Execution:**
   - A CI/CD pipeline defined in Azure DevOps is automatically triggered when changes are made or code is uploaded.
   - The Microsoft Self-Hosted Agent is responsible for building, packaging, and deploying the analysis logic to the Databricks runtime.
3) **Preprocessing:**
   - The uploaded source code is scanned for syntax correctness.
   - Code is tokenized and segmented into functions or modules to enable granular analysis.
4) **Static Code Analysis (in Databricks):**
   - The analysis app deployed on Databricks uses integrated static analysis tools (e.g., SonarQube API or custom parsers).
   - Metrics such as cyclomatic complexity, code duplication, vulnerabilities, and code smells are extracted.
   - AI-powered enhancements (optional) generate suggestions for improvement.
5) **Postprocessing and Report Generation:**
   - The extracted analysis results are formatted into a structured report (JSON/HTML).
   - Summaries of the most critical findings are highlighted.
   - The report is uploaded to Azure Blob Storage.
6) **Result Delivery:**
   - The analysis results are displayed via the chatbot interface on Databricks.
   - Users can view the report or interact further for clarification through the AI assistant.

V. RESULTS

The DocuCore system was evaluated based on its performance across two primary functionalities: document summarisation and source code analysis. The results obtained reflect the efficiency, reliability, and accuracy of the platform when integrated with cloud-native tools and intelligent automation.

- **Document Summarisation:** The summarisation model deployed on Azure Databricks successfully processed and summarised diverse documents (research papers, articles, project reports) with an average response time of 10–15 seconds per document. The summaries retained all critical information while reducing text length by approximately 60–70%.
- **Code Analysis:** Uploaded source code files were scanned using static analysis tools integrated with Databricks. Reports included metrics such as cyclomatic complexity, code duplication, and identified vulnerabilities. All critical issues were correctly flagged with 100% detection on pre-seeded error cases.
- **System Automation:** Azure DevOps pipelines (with a Microsoft Self-Hosted Agent) ensured consistent deployment and testing of both apps. Pipelines were completed in under 2 minutes, with zero failures during execution after initial debugging.
- **User Feedback:** Peer and faculty reviewers reported that the system was easy to use, responsive, and provided clear outputs. The chatbot interface simplified interaction for non-technical users.

In summary, DocuCore achieved its objective of integrating LLMs, static analysis, and cloud DevOps tools into a unified, scalable platform. The performance was consistent and production-ready under academic and test scenarios.

REFERENCES

[1] OpenAI, "GPT-3.5 Documentation," 2024. [Online]. Available: https://platform.openai.com/docs

[2] Microsoft Azure, "Azure Databricks Documentation," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/databricks

[3] Microsoft Docs, "Azure DevOps Documentation," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops

[4] SonarSource, "Static Code Analysis with SonarQube," 2024. [Online]. Available: https://www.sonarqube.org

[5] Microsoft, "Using Self-Hosted Agents in Azure Pipelines," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/devops/pipelines/agents/agents

[6] Databricks, "Deploying Apps and Jobs on Azure Databricks," 2024. [Online]. Available: https://docs.databricks.com

[7] Microsoft Azure, "Blob Storage for Scalable Object Storage," 2024. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/

[8] Chongzhou Fang, Ning Miao, Shaurya Srivastav, Jialin Liu, Ruoyu Zhang, Ruijie Fang, Asmita, Ryan Tsang, Najmeh Nazari, Han Wang, and Houman Homayoun, "Large Language Models for Code Analysis: Do LLMs Really Do Their Job?," University of California, Davis, and Temple University. [Online]. Available: https://arxiv.org/abs/2301.10551

[9] Lochan Basyal, "Summarization Using Large Language Models: A Comparative Study of MPT-7b-instruct, Falcon-7b-instruct, and OpenAI Chat-GPT Models," arXiv, 2023. [Online]. Available: https://arxiv.org/abs/2310.10449v2