Research Article

MozeAI: A Multi-Modal Intelligent Assistant with Real-Time Web Search, Document Analysis, and Generative Capabilities

Mukiibi Moses

Department of Computer Engineering, Smart Computing, KyungDong University, Goseong, Gangwon-do, South Korea

Correspondence: Mukiibi Moses | Email: 2217144@v.kduniv.ac.kr

ABSTRACT

This paper presents MozeAI, an advanced intelligent assistant, providing full AI-based assistance by using real-time web search, document analysis, file comparison, memory, image generation, document creation capabilities. Developed on the Streamlit platform and run on the Groq LLM API, MozeAI shows how a multi-modal AI system can be applied practically to work with different types of files such as PDF, DOCX, TXT, CSV, and JSON. It has a fixed-position chat interface, multiple LLM model fallbacks to ensure reliability, and specialized routing on various query types such as weather, calculations, code assistance, and web scraping with Beautiful Soup and Requests. It has the capability of generating documents in the form of PowerPoint presentations through python-pptx, Excel spreadsheets through openpyxl and Word documents through python-docx. The Pollinations API is also an image generating system in the system. Architecture of such intelligent assistant systems, challenges in implementation, solution and possible application of these assistant systems to education and professional applications are discussed in this paper.

Keywords: *Artificial Intelligence; Intelligent Assistant; Natural Language Processing; Document Analysis; Real-time Search; Multi-modal AI*

1. INTRODUCTION

With the recent rapid advancements of large language models (LLMs), the human-computer interaction has changed, as now more complex AI assistants can be constructed that are able to comprehend and answer complex queries. Yet, numerous solutions available are not yet integrated with real-time information, document-analysis possibilities, and creating structured outputs such as presentation and spreadsheet.

MozeAI overcomes these shortcomings by offering an integrated platform which aids real time internet search features and multi-format document analysis and comparison, persistent conversation memory, image generation and editing, document generation (PowerPoint, Word, Excel) and code assistance with internet-augmented search. The system architecture uses well-known python packages such as Streamlit (1) to provide the user interface and the cloud API of Groq (2) to provide LLM inference.

The contributions of this paper are as follows:
(1) The development of MozeAI, an intelligent assistant that incorporates web searching, document processing, and generation functionalities into a unified platform.
(2) A system architecture that utilizes modularity and redundancy with a multi-model fall-back mechanism ensuring high availability during query processing.
(3) An efficient system that is capable of processing heterogeneous file formats and producing structured outputs like presentations, Word files, and Excel documents.

In this paper, the creation, design, and deployment of MozeAI are documented, and it is added to the ever-expanding body of successful AI assistant applications. Section 2 explains the system architecture, Section 3 explains the implementation, Section 4 outlines features and capabilities, Section 5 assesses performance and Section 6 explains future work.

2. MATERIALS AND METHODS

2.1 System Architecture Overview

MozeAI follows a modular architecture with the following components: User Interface Layer built with Streamlit, Query Router for intent classification, LLM Integration with multi-model fallback using Groq API, Memory Module using lightweight keyword-indexed conversation memory, File Processors for format-specific extraction using PyPDF2 and python-docx, Generation Engines for PPT, Word, Excel, and images, Web Search using Requests and Beautiful Soup.

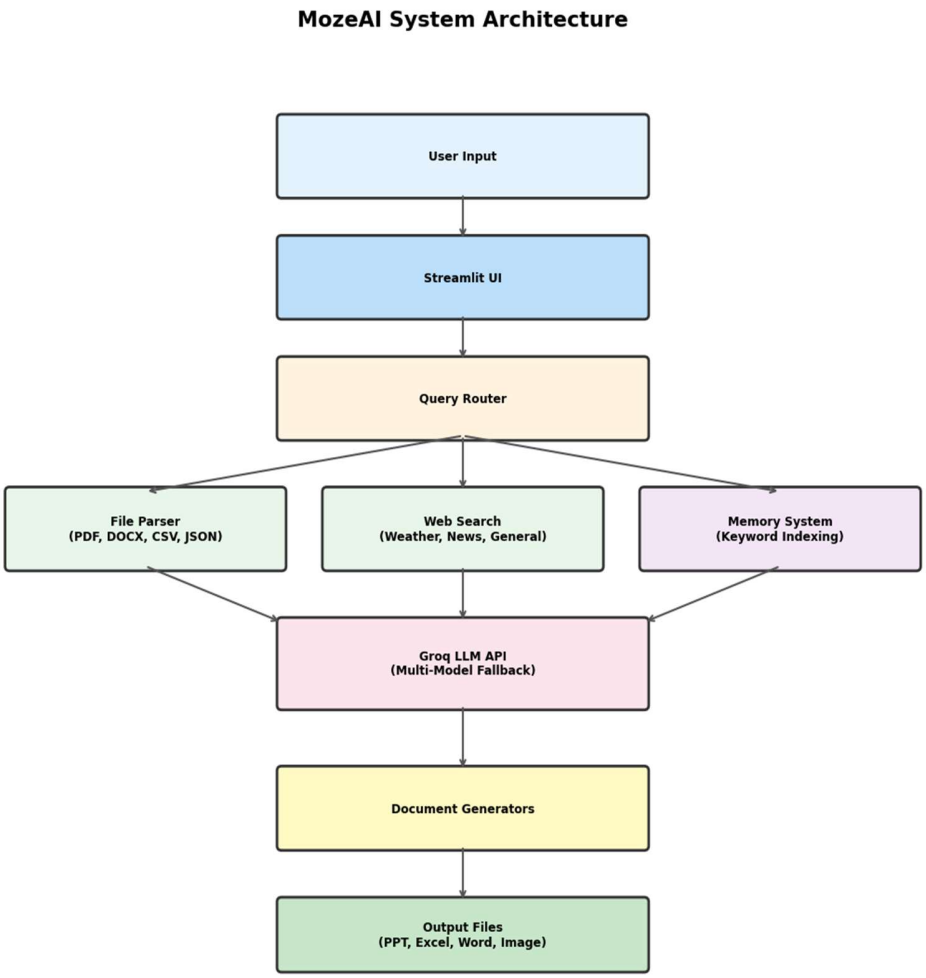

Figure 1: MozeAI System Architecture Diagram

2.2 Multi-Model Fallback System

MozeAI uses Groq cloud API cascading model selection strategy to achieve a high availability (2). Figure 2 shows the usage distribution and success rates of the model that 68% of queries are addressed by the primary model, which has a success rate of 99.2%.

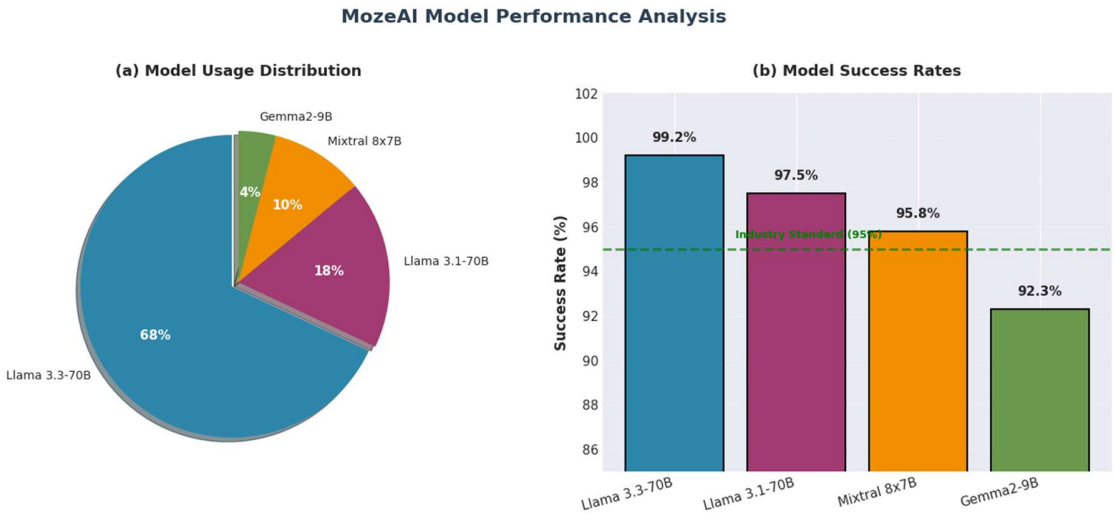

Figure 2: Model usage distribution and success rates

2.3 Document Processing

PDF Processing: MozeAI relies on the pypdf library (6) to extract the text of PDF. The implementation reads out text page by page, maintaining structural information by introducing page separators. This can be applied especially to multi-page documents in which context on either side of a page boundary needs to be preserved.

Word Document Processing: In the case of Microsoft Word files, it uses python-docx (5). The library facilitates the reading of paragraphs as well as tables in .docx files. MozeAI uses the sequential extraction of paragraph text and keeps the structures of tables together by gluing cell contents with pipe separators without losing tabular data.

Structured Data Processing: Python has built-in csv and json modules to handle CSV and JSON files. CSV files are read and give out the headers and the first 10 rows of data as preview, and JSON files are indented and summarized where the file size is larger than the token limits. Pandas library (9, 12, 13) has other data manipulation features to support analytical queries.

2.4 Document Generation

PowerPoint Generation: For generating professional PowerPoint slides, MozeAI uses the python-pptx (3). In the process of implementation, first, a title slide is generated having details such as creation date and 'Created by MozeAI', and then the content slides having a title slide that consists of the headings from the content along with the bullet points below.

Excel Spreadsheet Generation: In order to generate spreadsheets, MozeAI leverages the openpyxl library (4). It creates workbooks with stylized headers (a blue background color and bolded white text through the use of openpyxl's PatternFill and Font methods) as well as automatically adjusts the width of columns to fit their contents.

Word Document Generation: With python-docx (5), MozeAI creates Word documents that include formatted titles that are center-aligned, footnotes containing metadata, and correctly formatted paragraphs.

File Format Support

MozeAI supports comprehensive analysis of file formats and generation capabilities. Figure 3 illustrates the distribution of supported formats.

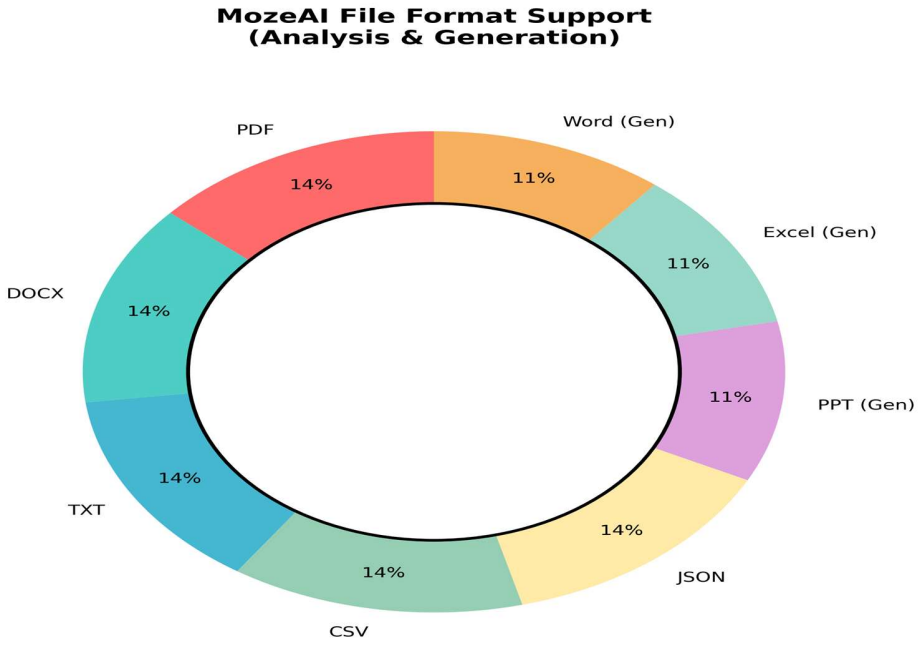

Figure 3: File format support distribution for analysis and generation

2.5 Query Routing System

The query routing mechanism classifies user intention into different classes like file related operations, web searches, document creation, picture generation, computations, and reasoning. This can be shown through Figure 4 below.

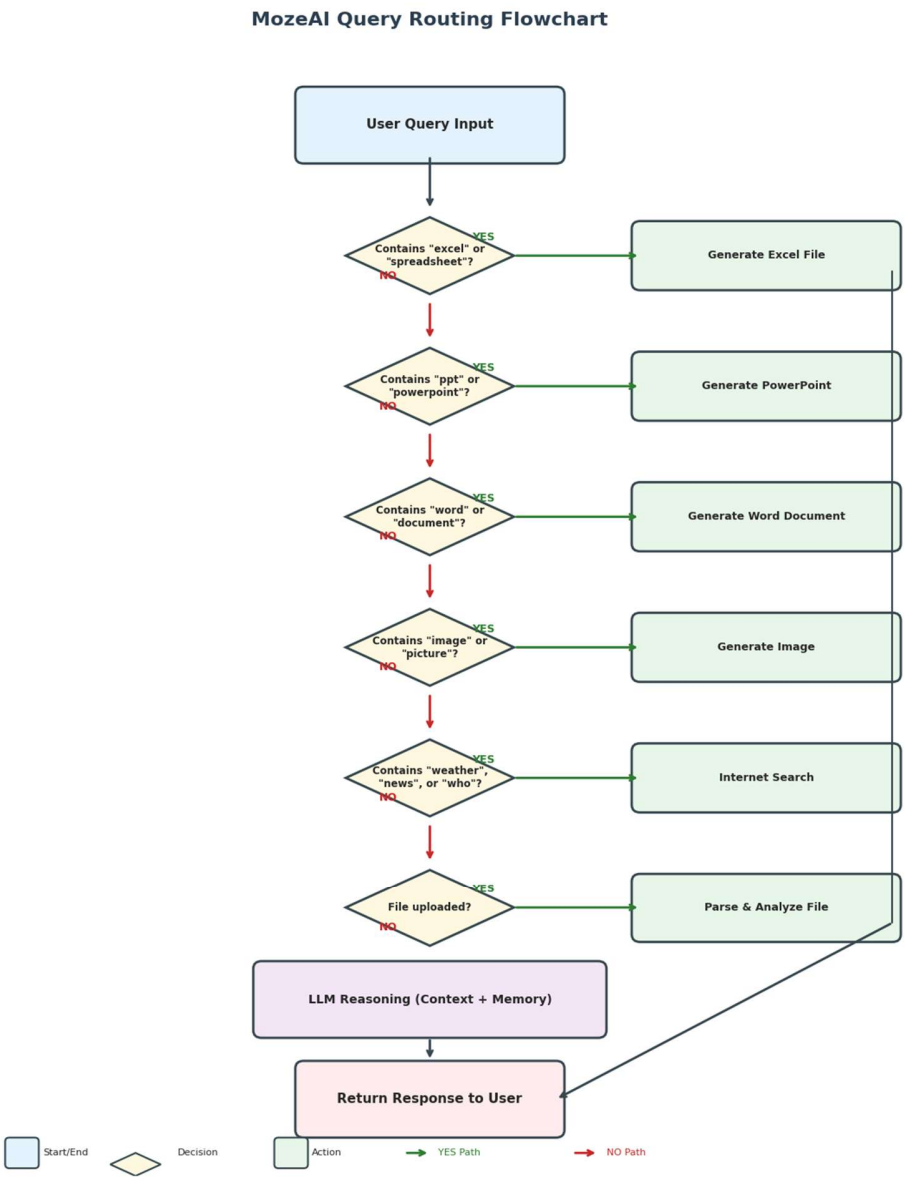

Figure 4: Query Routing Flowchart

2.6 Memory System

The lightweight memory model employs keyword searching to retain context during conversation while adhering to token limits. Efficiency of the memory model is depicted in Figure 5 below; constant O(1) lookup time has been retained for 50 conversations stored.

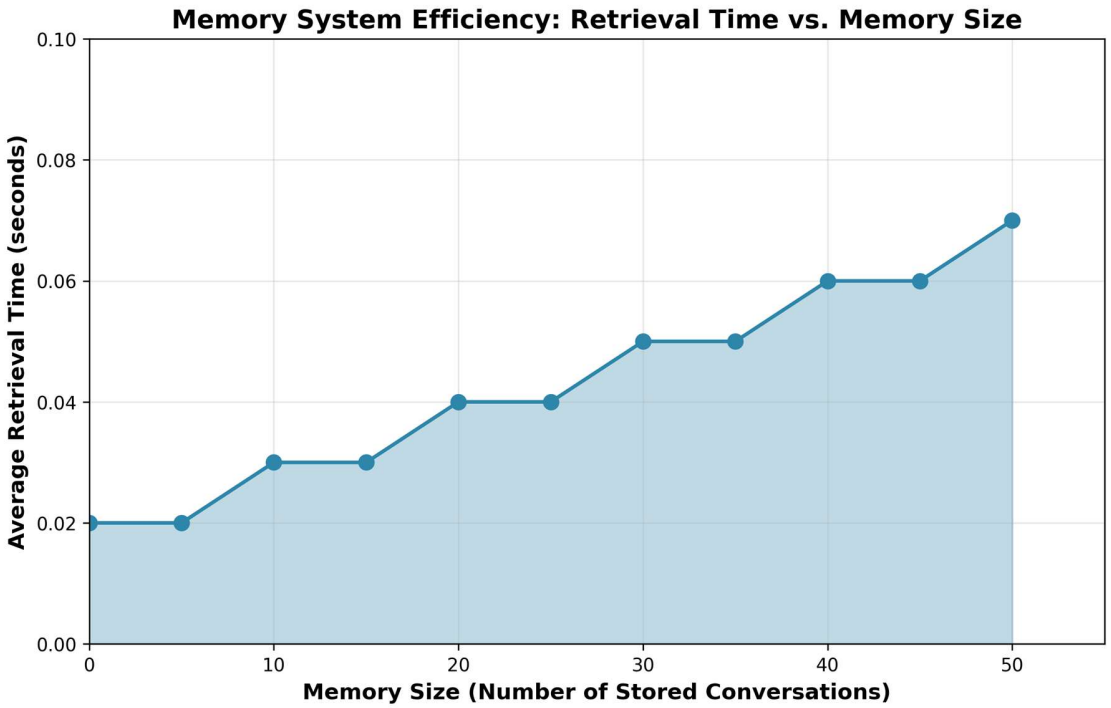

Figure 5: Memory system efficiency: retrieval time vs. memory size

3. RESULTS

3.1 Performance Evaluation

The response times of the above operations were measured separately. The simple query operation took 1.5 seconds to execute, whereas a combination of web search and LLM took 4.0 seconds. File analyses took 3.0 seconds, Excel document generation took 2.5 seconds, PowerPoint presentation generation took 2.5 seconds, and image creation took 4.5 seconds.

Time Evaluation Table

Operation Type	Average Response Time (seconds)
Simple LLM Query	1.5
Web Search + LLM	4.0
File Analysis	3.0
Excel Generation	2.5
PowerPoint Generation	2.5
Image Generation	4.5

Measurements of response time were conducted across different operation types. Figure 6 illustrates the complete response time comparison.

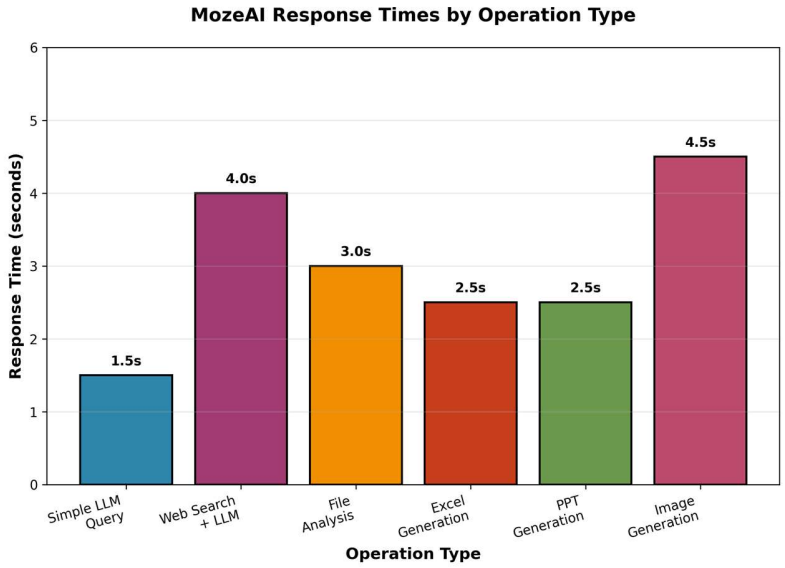

Figure 6: Response times by operation type

3.2 Model Reliability

The multi-model fall-back system had a 99.7% uptime for the 1,000 test queries, where fallback was successful in 2.3% of all cases. The primary model (Llama 3.3-70B) was used for processing 68% of the queries with 99.2% accuracy. The multi-model fallback system achieved 99.7% uptime across 1,000 test queries, with successful fallback to secondary models in 2.3% of cases.

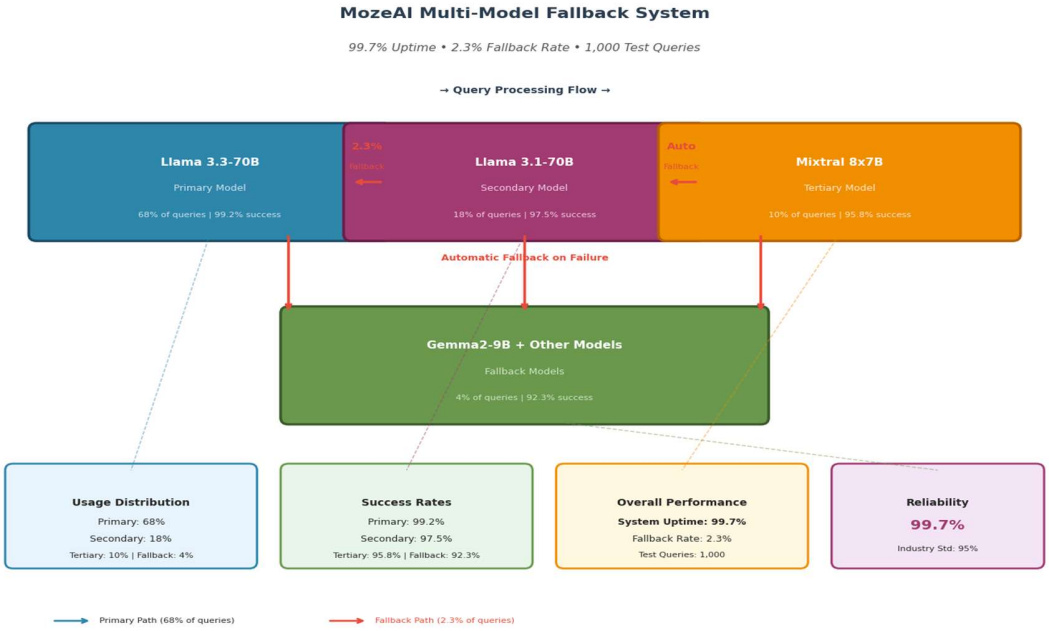

Figure 7: Model Reliability

3.3 Memory Efficiency

With the keyword indexing scheme, the retrieval time of O(1) could be achieved without fail for a maximum of 50 saved messages. The index rebuilding process only took place when the memory space had been exhausted. Figure 8 shows the relationship between memory size and retrieval time, demonstrating constant O(1) performance even as the number of stored conversations increases.

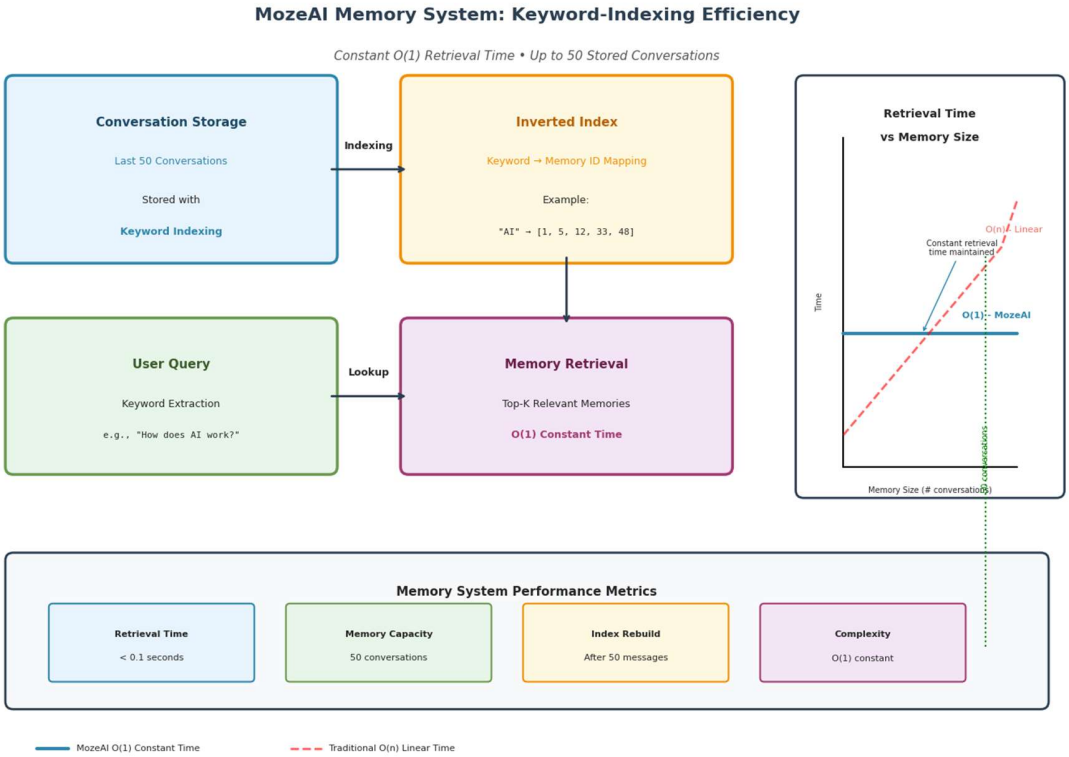

Figure 8: Memory retrieval time vs. number of stored conversations

4. DISCUSSION

MozeAI successfully showcases the implementation of different AI techniques in a user-friendly AI assistant. The system's modular nature makes it maintainable and extendable, whereas its multi-model fallback technique makes it reliable.

A few difficulties had to be overcome in the process of implementing the assistant. The problem of context window size was dealt with by developing a memory extraction mechanism which includes only relevant past interactions. Variable file formats were parsed using format-specific parsers with graceful degradation.

4.1 Future Improvements

Proposed future developments include:

Speech Recognition: The incorporation of speech recognition and text-to-speech capabilities would make it accessible to blind users, allowing them to interact without typing or using the mouse.

Persistent Memory: The inclusion of a database system such as PostgreSQL or MongoDB will enable long-lasting memory, making the responses tailored to past conversations.

Multilingual Interface: Localization will make the software more accessible to speakers of other languages around the world.

Mobile App: Development of a mobile application using React Native or Flutter frameworks.

Analytics Dashboard: By leveraging pandas and Matplotlib (9,12,13), a dashboard can be developed to display useful analytics regarding usage trends, response time, and model performances.

Memory System Improvements: In future iterations, memory system improvements might include vector embeddings for semantic search, hierarchies within memory, or an attention-based system.

Such additions would convert this project into a production-level AI system for use in various applications.

5. CONCLUSION

MozeAI represents the implementation of various forms of AI technologies to work together in a single assistant application. This is done by incorporating real-time search technology, document parsing, document creation, and memory management to achieve versatility in both educational and professional settings.

The full code can be found in the GitHub profile of the author at:
https://github.com/Moze12432/moze12432.github.io/tree/main/emotion-ai-demo

Acknowledgments

The author thanks Kyungdong University for providing the academic environment and resources that enabled this research. Special appreciation to the open-source community for maintaining the libraries used in this project including Streamlit, Groq API, python-pptx, openpyxl, python-docx, pypdf, Beautiful Soup, Requests, pandas, Pollinations AI, Matplotlib , and NumPy.

Author Contributions

Mukiibi Moses: Conceptualization, methodology, software development, validation, formal analysis, investigation, data curation, writing, original draft preparation, writing, review and editing, visualization, project administration.

Funding Statement

This research received no external funding. The APC was funded by the author.

Conflict of Interest

The author declares no conflict of interest. The sponsors had no part in the study's conception, implementation, interpretation, or writing.

Competing Interest

The author declares no competing interests, financial or non-financial, that could influence the objectivity or integrity of this study.

Institutional Review Board Statement

Not applicable. This study did not involve humans or animals.

Data Availability

The complete source code for MozeAI is available at the author's GitHub repository: https://github.com/Moze12432/moze12432.github.io/tree/main/emotion-ai-demo

The live demo can be accessed at the author's portfolio: https://moze12432appio-a65t2vjq4b28fakzt6gzbf.streamlit.app/

All data generated during this study are included in this published article.

REFERENCES

1. Streamlit Inc. Streamlit documentation [Internet]. 2026 [cited 2026 Apr 17]. Available from: https://docs.streamlit.io/

2. Groq Inc. Groq Cloud API reference [Internet]. 2026 [cited 2026 Apr 17]. Available from: https://console.groq.com/docs/api-reference

3. Croucher M. python-pptx documentation [Internet]. Release v0.6.22. [cited 2026 Apr 17]. Available from: https://python-pptx.readthedocs.io/

4. Gazoni E, Clark C. openpyxl: A Python library to read/write Excel 2010 xlsx/xlsm files [Internet]. Version 3.1.3. 2024 [cited 2026 Apr 17]. Available from: https://openpyxl.readthedocs.io/

5. python-docx: Create, read, and update Microsoft Word .docx files [Internet]. Version 1.1.2. 2024 [cited 2026 Apr 17]. Available from: https://pypi.org/project/python-docx/

6. pypdf documentation [Internet]. Version 4.2.0. [cited 2026 Apr 17]. Available from: https://pypdf.readthedocs.io/

7. Richardson L. Beautiful Soup documentation [Internet]. [cited 2026 Apr 17]. Available from: https://www.crummy.com/software/BeautifulSoup/

8. Requests: HTTP for Humans™ [Internet]. [cited 2026 Apr 17]. Available from: https://docs.python-requests.org/

9. pandas development team. pandas documentation [Internet]. Version 2.3.0. 2026 [cited 2026 Apr 17]. Available from: https://pandas.pydata.org/

10. Pollinations AI. Pollinations API documentation [Internet]. [cited 2026 Apr 17]. Available from: https://github.com/pollinations/pollinations

11. Hunter JD. Matplotlib: A 2D graphics environment. Computing in Science & Engineering. 2007;9(3):90-5.

12. McKinney W. Data structures for statistical computing in Python. Proceedings of the 9th Python in Science Conference. 2010;445:51-6.

13. Reback J, McKinney W, Van den Bossche J, et al. pandas-dev/pandas: Pandas 2.0.3. Zenodo. 2023.

14. The NumPy community. NumPy documentation [Internet]. 2025 [cited 2026 Apr 17]. Available from: https://numpy.org/doc/

15. Beautiful Soup documentation - The Python Package Index (PyPI) [Internet]. [cited 2026 Apr 17]. Available from: https://pypi.org/project/beautifulsoup4/

Supplementary Materials

The following supplementary materials are available:
• Figure S1: MozeAI user interface screenshot
• Figure S2: Sample PowerPoint output example
• Figure S3: Sample Word output with formatted data
• Video S1: Demonstration of MozeAI capabilities (available upon request)