# A Hybrid Retrieval-Augmented Conversational AI System using LangChain and FAISS

Abdul Hai[1], Dr. Gousiya Begum[2], Dr. K. Rajitha[3] and R. Mohan Krishna Ayyappa[4]
[1]*Student, Mahatma Gandhi Institute of Technology*
[2,3,4]*Assistant Professor, Mahatma Gandhi Institute of Technology*

***Abstract*—This paper presents a conversational AI system capable of retrieving and synthesizing information from PDFs and web URLs using a hybrid retrieval-augmented generation (RAG) framework. The system integrates FastAPI for backend processing and Streamlit for a lightweight, user-friendly interface. By combining BM25 for keyword search with FAISS-based semantic analysis, the platform offers high retrieval accuracy and contextual relevance. Features such as document summarization, persistent conversation history (via SQLite), and modular architecture make the system well-suited for academic, research, and professional use.**

***Index Terms*—BM25, FAISS, LangChain, RAG**

## I. INTRODUCTION

In recent years, conversational AI systems have become integral to various domains such as customer service, education, healthcare, and enterprise knowledge management. However, traditional language models often struggle with retrieving accurate and up-to-date information, particularly when faced with domain-specific queries or unstructured datasets like PDFs and web content. Retrieval-Augmented Generation (RAG) has emerged as a promising solution to address these limitations by combining retrieval mechanisms with generative models, thereby enhancing accuracy and mitigating hallucinations [1] [9].

This project presents a conversational AI system specifically designed to retrieve relevant information from PDFs and URLs, focusing on technical content such as computer networks. The system integrates a hybrid RAG framework that combines BM25 for keyword-based retrieval [2] and FAISS for semantic similarity matching using dense vector embeddings [10]. By leveraging the strengths of both retrieval strategies, the system ensures high precision in keyword matching while maintaining contextual relevance through semantic search.

The backend is developed using FastAPI, providing efficient API handling for queries and document ingestion. The frontend is implemented in Streamlit, offering a lightweight and intuitive user interface accessible via web browsers. LangChain serves as the orchestration framework for chaining document loaders, retrievers, and response generation tools, simplifying complex workflows and enabling scalable development [2] [3] [4].

In addition to real-time query processing, the system includes persistent conversation history storage using SQLite, allowing users to revisit prior interactions. This feature makes it particularly useful for researchers, students, and professionals working with technical documents. The hybrid approach is further enhanced by query expansion techniques and tunable weights, which ensure a balanced contribution from both keyword and semantic retrieval components [8].

The entire system is designed to run locally with minimal resource requirements, making it accessible even without high-end hardware or cloud infrastructure. At the same time, it remains flexible for future deployment on cloud platforms to support multi-user access and scalability [5] [6].

By integrating RAG principles, LangChain workflows, and a modular, user-friendly interface, this system provides a powerful tool for accurate, context-aware knowledge retrieval from unstructured textual data. It not only addresses the challenges of retrieval precision and contextual comprehension but also lays the foundation for future enhancements like OCR integration, reranking, and multilingual support [3][7].

## II. LITERATURE REVIEW

Recent research has established the effectiveness of Retrieval-Augmented Generation (RAG) in enhancing the performance of large language models (LLMs), particularly for question answering and chatbot applications. RAG improves the accuracy of responses by mitigating hallucinations and outdated knowledge, though challenges such as handling long-tail data persist [1][9]. RAG techniques in natural language processing have been explored with emphasis on retriever architectures and fusion strategies, though computational overhead remains a significant constraint [10].

LangChain has been widely adopted for building scalable and modular LLM-based applications. It has been used to create efficient query systems, demonstrating rapid retrieval from unstructured datasets [2]. Other implementations have proposed frameworks for fast development of LLM apps across various domains [4]. Additional research introduced a LangChain-powered news research tool, improving retrieval relevance through similarity search and token limit optimization [7].

Several studies have combined LangChain and RAG to create custom conversational agents. These systems demonstrated improved responsiveness and modularity, though they required careful design to maintain performance across domains [3][6]. One implementation automated Snowflake database query using LangChain and RAG, effectively replacing dashboard tools like Power BI, but faced limitations when scaling to large datasets [5].

Overall, these studies confirm the potential of combining RAG and LangChain to improve LLM responsiveness and accuracy. However, unresolved issues such as computational load, scalability, and real-time integration continue to present opportunities for improvement [8][9][10].

## III. DESIGN

The system follows a modular architecture as shown in Fig. 1 that supports efficient, accurate, and scalable information retrieval. It integrates both keyword and semantic search mechanisms, delivering results that balance precision and contextual understanding.

### *A. System Workflow*

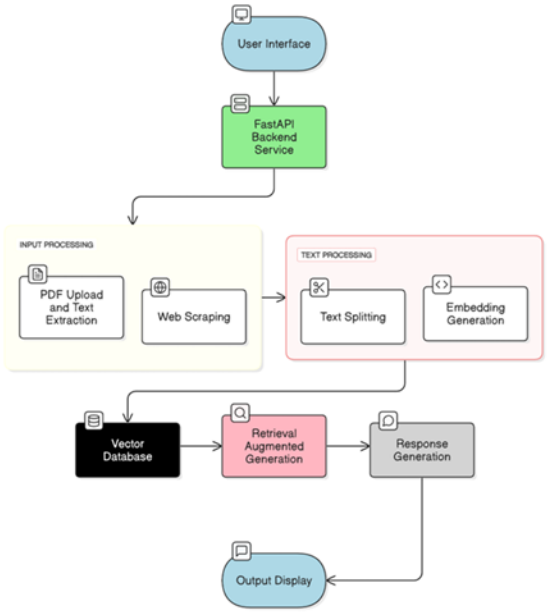


Fig. 1. System Workflow using various components

Data Processing: PDFs and URLs are ingested and segmented into smaller textual chunks to facilitate fine-grained retrieval.
Query Handling: User inputs are expanded with related terms and processed through the hybrid retrieval mechanism.
Retrieval and Ranking: BM25 and FAISS are applied in parallel, and results are ranked based on their combined relevance scores.
Response Generation: The top-ranked segments are returned to the user along with metadata, enabling transparency and contextual clarity.

### *B. Interface Components*

The web interface, built with Streamlit, emphasizes simplicity and responsiveness. Users can:

- Submit search queries
- Upload PDF files or input URLs
- View the top-5 relevant segments along with source metadata
- Read brief summaries and identify keyword highlights

This design ensures minimal learning curve and seamless user interaction across devices and browsers.

*C. User Roles*

• General Users: Seek concise information for study or research.
• Professionals: Require detailed, reliable references for analysis and decision-making.

*D. Design Goals*

• Modularity: Separate components for retrieval and ranking enable easy updates and testing.
• Scalability: Supports growth to handle larger datasets or additional search methods in the future.
• Accessibility: Works on any browser, requires no installation, and is intuitive for all users.

## IV. IMPLEMENTATION

The system was developed using Python and delivered through a web-based interface. The implementation involved setting up data processing, retrieval mechanisms, and a simple web tool for user interaction with the search system.

*A. Tools and Technologies*

Programming Language: Python 3.9
Development Environment: VS Code
Libraries:
- Data Handling: LangChain, FAISS
- Retrieval: sentence-transformers(for embeddings), rank_bm25 (BM25)

Backend: FastAPI
Web Interface: Streamlit
Utilities: NumPy, logging

*B. Retrieval System Setup*

• BM25: Configured for keyword-based retrieval with tuned parameters (k1=1.2, b=0.3) to prioritize exact matches.
• FAISS: Used the all-mpnet-base-v2 model for generating dense vector embeddings. Search scope was expanded using
*n*probe=20, enhancing recall.
• Hybrid Retrieval: Combined BM25 and FAISS with weights of 0.3 and 0.7, respectively, and applied query expansion to enhance search terms.
The retrieval logic was implemented as an API endpoint for efficient processing.

C. FastAPI and Streamlit Integration

FastAPI was used to create a backend server that handles query processing and retrieval, while Streamlit provided the user interface with:
• Query input field for user searches
• Display of the top-5 relevant text segments
• Metadata alongside results, such as segment source
• Highlighted keywords in results for clarity
The system runs locally as shown in Fig. 2

D. Execution Requirements

• System Specs: Min. 2GB RAM, i3 processor, stable internet
• Software Dependencies: Captured in a requirements list.

The system achieved strong retrieval performance, with a Precision@5 of 0.68, Recall@5 of 0.8285, and MRR of 1.0. The hybrid strategy consistently outperformed single-method approaches, with FAISS contributing to contextual recall and BM25 improving exact-match precision.

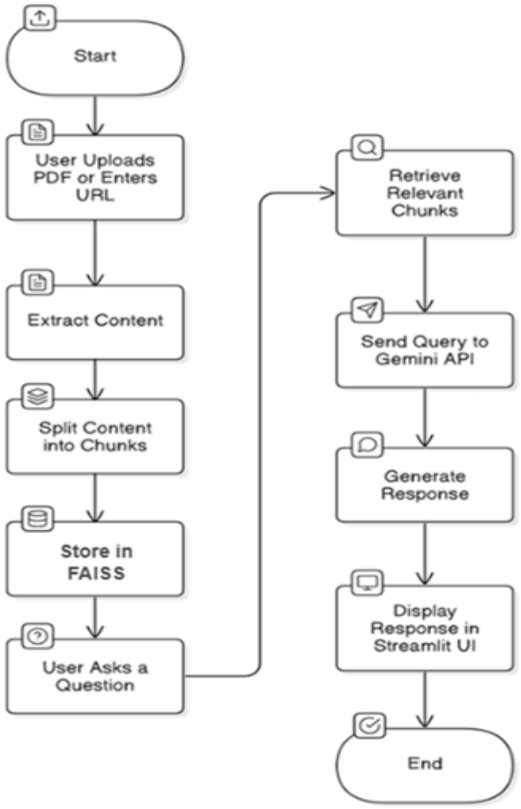


Fig. 2. Process flow diagram showing retrieval steps, from query input to result display.

## V. RESULTS AND DISCUSSIONS

The system was evaluated both quantitatively, using established retrieval metrics, and qualitatively, based on interface usability and responsiveness. The findings highlight the system's effectiveness and point to areas for enhancement.

### *A. Retrieval Performance*

Evaluation was conducted using 10 representative queries related to computer networking. The system achieved:

Precision@5: 0.68 — 68% of the top five retrieved segments were relevant (34 out of 50).

Recall@5: 0.8285 — 82.85% of all relevant segments (29 out of 35) were retrieved in the top five.

MRR: 1.0 — In all cases, the first returned segment was relevant.

These results as shown in Fig. 3 affirm the hybrid model's ability to rank relevant content accurately, with FAISS contributing significantly to recall and BM25 ensuring keyword-level precision.

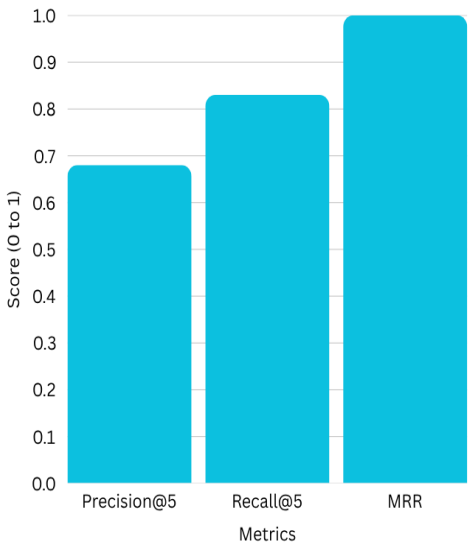


Fig. 3. Graph depicting Precision, Recall and MRR on a scale of 0 to 1.0

### *B. Result Presentation*

The Streamlit-based interface displays the top-5 retrieved segments along with metadata, such as source document name or section. Keywords are highlighted to aid quick identification. The interface supports real-time interaction, updating results dynamically as users submit new queries.

### *C. Backend Efficiency*

The FastAPI backend processes queries efficiently, delivering rapid response times. The hybrid retrieval strategy—weighted 0.3 (BM25) and 0.7 (FAISS)—was optimized with tuned parameters and query expansion, achieving a consistent balance of speed and relevance.

### *D. User Interface Experience*

The Streamlit interface as shown in Fig. 4 offers an intuitive experience, with features like:

- A text field for entering search queries
- A side-bar to upload PDFs or enter a URL with a process button
- Display of the answered query along with the top-5 retrieved segments with source details
- Highlighted keywords within results for quick scanning
- Fast updates as users submit new queries

The interface was tested on multiple browsers (Chrome, Safari, Edge) and devices, ensuring consistent functionality and accessibility for all users, regardless of technical background.

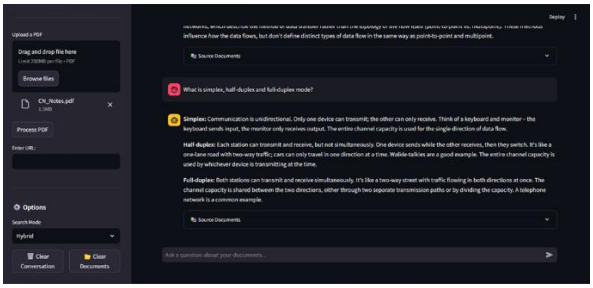

Fig. 4. User-interface with sidebar and textbox

### *E. Limitations*

Despite strong performance, the current version has some limitations:

Limited Dataset: Only 166 segments were used; this may constrain generalizability.

Ambiguous Queries: Vague or broad queries occasionally yield semantically similar but irrelevant results.

Scalability: Scaling to larger corpora or concurrent users would require cloud infrastructure and memory optimization.

*F. Observations and Future Improvements*

Observations indicate that the system performs well for specific, well-defined queries. However, precision may degrade for abstract or multi-faceted queries due to FAISS overgeneralization. To address this:

• A reranking mechanism (e.g., cross-encoder) could improve result ordering.
• Expanding the dataset would increase topic diversity and improve recall.
• Support for multi-term and natural language queries could increase search flexibility.
•A cloud-based deployment would enhance scalability and reliability.

## VI. CONCLUSION AND FUTUREWORK

*A. Conclusion*

This project presents a robust and user-friendly conversational AI system for retrieving information from unstructured text sources such as PDFs and web URLs. By integrating a hybrid retrieval mechanism—BM25 for keyword-based precision and FAISS for semantic relevance—the system delivers accurate, context-aware responses. A modular architecture supported by FastAPI and Streamlit ensures efficient processing, persistent conversation history, and a responsive user interface. The solution is particularly well-suited for academic and professional domains where fast and reliable knowledge access is essential.

*B. Future Work*

To enhance system capabilities and address current limitations, the following improvements are proposed:

1. Reranking Mechanism: Implementing a cross-encoder model can refine the ranking of retrieved results, potentially improving Precision@5.
2. Expanded Dataset: Incorporating a more diverse and comprehensive document set will improve the system's ability to generalize across domains.
3. Advanced Query Handling: Supporting natural language and multi-term queries will allow for more nuanced search behavior and better relevance matching.
4. Cloud Deployment: Hosting the system on cloud platforms (e.g., AWS, Google Cloud) will enhance scalability and enable support for multiple concurrent users.
5. OCR Integration: Adding optical character recognition will extend functionality to scanned documents and image-based PDFs, broadening document compatibility.

## REFERENCES

[1] P. Zhao et al., "Retrieval-Augmented Generation for AI-Generated Content: A Survey," arXiv, 2024.
[2] A. S. Sreeram and P. J. Sai, "An Effective Query System Using LLMs and LangChain," Int. J. Eng. Res. Technol., vol. 12, pp. 789–796, 2023.
[3] S. Vidivelli, V. Ramachandran, and N. Dharunbalaji, "Efficiency-Driven Custom Chatbot Development: Unleashing LangChain, RAG, and Performance-Optimized LLM Fusion," Comput. Mater. Contin., vol. 78, pp. 145–162, 2023.
[4] O. Topsakal and T. C. Akinci, "Creating Large Language Model Applications Utilizing LangChain: A Primer on Developing LLM Apps Fast," Int. Conf. Appl. Eng. Nat. Sci., pp. 210–218, 2023.
[5] A. Bansal, "Building a RAG System using LLMs And Langchain to automate Snowflake Database Queries: A Chatbot Solution for replacing Power BI Dashboards," Int. J. Core Eng. Manag., vol. 11, pp. 56–67, 2024.
[6] K. Pandya and M. Holia, "Automating Customer Service using LangChain: Building custom open-source GPT Chatbot for Organizations," arXiv, 2023.
[7] R. Kumar et al., "LLM Based News Research Tool Using LangChain with Enhancing Similarity Search and Token Limit," Int. J. Res. Publ. Rev., vol. 5, pp. 342–350, 2024.
[8] K. Pichai, "A Retrieval-Augmented Generation Based Large Language Model Benchmarked on a Novel Dataset," J. Stud. Res., vol. 12, pp. 89–102, 2023.
[9] Y. Gao et al., "Retrieval-Augmented Generation for Large Language Models: A Survey," arXiv, 2024.
[10] S. Wu et al., "Retrieval-Augmented Generation for Natural Language Processing: A Survey," arXiv, 2024.