IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

LEVELWISE RAG CHATBOT

An RBAC based Chatbot using GEMINI 1.5

[1]Prof. Sachin Charbhe, [2]Pratham Nagvekar, [3]Riyaz Tanwar, [4]Aiman Kulay, [5]Anmol Sharma

Professor, Department of Artificial Intelligence and Data Science,
Rizvi College of Engineering, Mumbai, India

Abstract: This research addresses the challenge of inefficient and often insecure information retrieval in college settings by proposing an intelligent conversational interface. We developed a chatbot system that allows users including Heads of Departments, Professors, and Students to query a MongoDB database containing institutional data using natural language.

The system employs Retrieval-Augmented Generation (RAG) orchestrated by Langchain, using FAISS vector stores and the Gemini 1.5 Flash as the LLM via Google AI API to provide contextually relevant and factually grounded answers, minimizing LLM hallucinations. A key contribution is the tight integration of Role-Based Access Control (RBAC), ensuring data access aligns strictly with user roles through index and metadata filtering. Furthermore, post-retrieval filtering is implemented to enhance the LLM's accuracy when handling specific tasks like counting. Built with Streamlit, the chatbot offers a user-friendly platform for secure, role-appropriate, and efficient access to college information, streamlining data interaction for all user groups.

Keywords - Retrieval-Augmented Generation (RAG), Langchain, FAISS, Gemini 1.5 Flash, LLM, Google AI API, Role-Based Access Control (RBAC), Streamlit

I. INTRODUCTION

Because data is dispersed across multiple systems and conventional search methods are time-consuming, learning environments frequently struggle to provide effective access to information. This may cause delays and inaccurate information since it may be difficult to find specific information about departments, instructors, and students or other data stored by the users. A promising approach to expediting information retrieval is conversational AI, which can comprehend natural language queries. Chatbots can offer rapid and easy access to necessary data by letting users ask questions in their own words. Two important technologies that can further improve the efficiency and security of such systems are Role-Based Access Control (RBAC) and Retrieval-Augmented Generation (RAG). RBAC makes sure that users only have access to the data that is pertinent to their roles, while RAG grounds chatbot responses in particular knowledge bases, increasing their accuracy and dependability**.**

The goal of this research is to create a chatbot system that overcomes the drawbacks of conventional search techniques and offers quick, accurate, and role-appropriate access to college information. The system meets the need for a centralized, user-friendly platform that can securely and contextually deliver pertinent information to various user groups, including professors, students, and heads of departments. A RAG-based chatbot system with integrated RBAC is the suggested remedy. To comprehend user inquiries and produce natural language responses, the system makes use of a Large Language Model (LLM). It uses the FAISS vector search library to retrieve pertinent data from a MongoDB database and makes sure that users only see data that they are permitted to view according to their designated role. To further hone the information that has been retrieved and raise the precision of the LLM's responses, the system also uses post-retrieval filtering.

This is how the rest of the paper is structured. A review of related work is given in Section II. Detalis about the system is described in Section III. The approach used to create the RAG chatbot system is described in detail in Section IV. Results are presented and discussed in Section V. The paper is finally concluded in Section VI. The future work are outlined in Section VII. Finally Acknowledgement and Reference outlined in Section VIII and IX.

II. LITERATYRE REVIEW

2.1 Chatbots and Conversational AI

Chatbots are AI-powered programs designed to simulate human conversation, finding uses across various sectors. In education, research explores their potential for personalized learning, student support, and administrative tasks. The modeling-languages.com link addresses access control in conversational interfaces, a pertinent aspect of chatbot design.

2.2 Retrieval-Augmented Generation (RAG)

By giving them access to outside knowledge sources, the Retrieval-Augmented Generation (RAG) framework improves Large Language Models (LLMs). This method tackles the drawbacks of LLMs, which occasionally produce erroneous or out-of-date information—a condition referred to as "hallucination." RAG guarantees that the generated responses are based on factual information by obtaining pertinent information from a knowledge base and giving it to the LLM as context. Retrieval, in which pertinent data is obtained from an outside source, and Generation, in which the LLM utilizes the information retrieved to create a response, are the two primary steps of the RAG pipeline. (An excellent place to start when implementing RAG is the medium.com link.

2.3 RBAC (Role-Based Access Control)

One security measure that limits system access to authorized users is called Role-Based Access Control (RBAC). RBAC makes sure that people can only access the data and resources required for their job functions by allocating permissions to users according to their roles within an organization. RBAC is essential for maintaining data security and safeguarding sensitive information in a college setting, such as faculty or student records and for maintaining security and privacy.

2.4 LLMs, or large language models

The chatbot system uses Gemini 1.5 Flash as its Large Language Model (LLM), accessed via Google AI API, to decode user inquiries and produce natural language answers. Choosing Gemini 1.5 Flash is taken in context of the overall debate of LLMs, both their pros and cons. The model recognizes that capabilities of the LLM, i.e., capabilities of Gemini 1.5 Flash to be instructed and create summaries of provided context, play an important part in the general accuracy and effectiveness of the system.

2.5 Applicable Technologies

To operate efficiently, the chatbot system makes use of several technologies. The user interface is created with Streamlit, which gives users a way to communicate with the chatbot. The framework for coordinating the RAG pipeline, which links the different parts of the system, is Langchain. The college's data is stored in a MongoDB database.

2.6 Research Focus and Literature Gap

Research Focus: The research centers on the construction of a chatbot system for university settings that offers effective and secure access to information in the form of natural language queries. The system strives to go beyond the weakness of conventional search methods through the implementation of a RAG-based chatbot with the inclusion of RBAC.

Literature Gap: The paper bridges the gap of easily retrieving role-appropriate information in college environments where information is usually dispersed and needs technical skills to query. Conventional search techniques are tedious and do not securely impose access privileges. The new integration of RAG and RBAC in this scenario bridges this gap with a system that offers precise, context-sensitive, and role-suitable information access.

III. PROPOSED SYSTEM

This research addresses the prevalent challenge of accessing specific, role-relevant information efficiently within college environments, where data is often siloed and requires technical expertise to query. Traditional search methods can be time-consuming and may not adequately enforce access permissions crucial in an academic setting. The proposed solution is a **Levelwise RAG Chatbot**, a conversational AI system designed to provide users (Students, Professors, and Heads of Departments - HODs) with intuitive, secure, and accurate access to institutional data stored in a MongoDB database.

The core of the proposed system lies in its integration of **Role-Based Access Control (RBAC)** with a **Retrieval-Augmented Generation (RAG)** architecture. RAG grounds the chatbot's responses in factual data retrieved directly from the knowledge base (MongoDB collections and uploaded files), thereby minimizing the risk of inaccurate or "hallucinated" information often associated with standalone Large Language Models (LLMs). RBAC ensures that users can only query and retrieve information appropriate to their designated role and departmental affiliation, maintaining data confidentiality and security.

The system leverages several key technologies: a **Streamlit** web interface provides a user-friendly conversational front-end; **LangChain** serves as the orchestration framework for the RAG pipeline; **MongoDB** acts as the primary data store, utilizing **GridFS** for efficient storage of uploaded files like departmental marks sheets; **FAISS** vector stores are used for efficient similarity searching of embedded data; and an LLM (initially Gemma 2 via Groq API, potentially switched to **Gemini 1.5 Flash via Google AI API**) handles natural language understanding and response generation.

A key contribution of this system is the implementation of **post-retrieval filtering**, designed specifically to enhance the LLM's accuracy when dealing with quantitative queries (like counting students or professors) by refining the context provided to the LLM based on the query intent. The system aims to create a centralized, context-aware platform that significantly improves information accessibility and usability for all members of the college community.

IV. SYSTEM DESIGN AND IMPLEMENTATION

4.1 System Architecture

The system follows a modular design comprising a presentation layer, an orchestration layer, data processing modules, and data storage layers.

- **Presentation Layer:** A web-based interface built using Streamlit (ui.py) handles user login, displays user information and chat history, provides input fields for queries, and includes conditional components for file/marks uploads based on user role.
- **Orchestration Layer (app.py):** The main Streamlit application script acts as the central orchestrator. It manages user sessions and authentication (st.session_state, access_control.py), initializes core LangChain components (LLM, embedding model, splitter), loads persistent data indexes (mongo_handler.py), processes file uploads by invoking embedding_manager.py, dynamically assembles the appropriate set of retrievers based on user role and available data sources (including loaded marks indexes), orchestrates the multi-step RAG query pipeline using components defined in chat_manager.py, and manages the overall application flow.
- **Data Processing Modules:**
    - mongo_handler.py: Connects to MongoDB, formats data from collections (departments, professors, students) using build_text_map, creates/loads/caches persistent FAISS indexes (mongodb_faiss_index_*) stored locally.
    - embedding_manager.py: Handles uploaded files. It uses LangChain document loaders (PyPDFLoader, UnstructuredExcelLoader, etc.) to read and chunk content, generates embeddings using the provided embedding_model, performs batch embedding for efficiency, and saves resulting FAISS indexes for marks files to local disk (faiss_indexes/marks_index_*.faiss).
    - access_control.py: Manages user authentication against predefined credentials and handles the storage of uploaded marks files into MongoDB GridFS, including relevant metadata.

- chat_manager.py: Defines functions to build prompt components (get_prompt_components) based on user context and refined instructions (for counting, formatting, etc.), and creates the LangChain chain components (create_history_aware_retriever_chain_component, create_qa_chain_component) used in the RAG pipeline.

- **Data Storage Layer:** Utilizes MongoDB for structured college data and GridFS for file storage, and the local filesystem for persistent FAISS indexes. Streamlit session state is used for temporary storage (e.g., FAISS index of a non-marks uploaded file).

4.2 Data Sources and Preparation

The system leverages two primary data sources:

1. **Core College Data (MongoDB):** Collections for departments, professors, and students reside in a MongoDB database. During initialization (and managed by @st.cache_resource), the load_or_create_faiss_indexes function in mongo_handler.py reads these collections. Data is formatted into text using helper functions (build_text_map), embedded using GoogleGenerativeAIEmbeddings, and potentially chunked using RecursiveCharacterTextSplitter (especially for department data). Embeddings are stored in persistent FAISS indexes on local disk, with department metadata added for professors and students collections to enable RBAC filtering.
2. **Uploaded Marks Files (GridFS & FAISS):** Professors or HODs can upload files (e.g., ME_Marks.xlsx) containing marks data. The upload_marks_file_to_gridfs function in access_control.py stores the file content in MongoDB GridFS (marks bucket) along with metadata like department, subject, and uploader. Subsequently, the process_uploaded_file_to_faiss function in embedding_manager.py is triggered. This function reads the uploaded file (potentially from a temporary copy), uses appropriate LangChain document loaders (PyPDFLoader, UnstructuredExcelLoader, etc.)to extract and chunk the text using the provided splitter, generates embeddings for the chunks in batches using the provided embedding_model, and saves the resulting FAISS index to a persistent local directory (./faiss_indexes), named according to the uploader's department (e.g., marks_index_ME.faiss).

4.3 Mechanism of Retrieval

The retrieval process aims to fetch the most relevant context for a given query, respecting RBAC constraints:

1. **History-Aware Query:** The user's input and chat history are processed by create_history_aware_retriever_chain_component (defined in chat_manager.py), which uses the LLM to potentially rephrase the query into a standalone question.
2. **Ensemble Retrieval:** The (potentially rephrased) query is passed to an EnsembleRetriever instance created in app.py. This ensemble combines multiple individual retrievers:
   - Retrievers for the relevant MongoDB FAISS indexes (e.g., professors, students), filtered by the user's department if the role is Professor or Student (search_kwargs={"filter": ...}).
   - A retriever for the department-specific marks FAISS index (loaded from disk via FAISS.load_local with allow_dangerous_deserialization=True), added only if the user is a Student/Professor/HOD of that department.
   - A retriever for a temporarily uploaded document (non-marks), added only if present in st.session_state['uploaded_doc_vs'].
3. **Document Fetching:** Each active retriever fetches its top k relevant documents based on vector similarity (with k configured potentially differently for MongoDB vs. uploaded sources). The EnsembleRetriever aggregates these results.

4.4 Post-Retrieval Filtering

To enhance accuracy, particularly for counting tasks that were initially inconsistent, an explicit filtering step occurs in app.py (Section 11) after documents are retrieved by the EnsembleRetriever but *before* they are sent to the final LLM:

1. The original user query is analyzed for keywords like "student" or "professor".
2. If the query appears specific (e.g., asking only about students), the list of retrieved documents (docs) is filtered to include only those whose page_content starts with the corresponding tag ([Student] or [Professor]).
3. This ensures the final QA chain receives a context (filtered_docs) more precisely tailored to the specific entity type mentioned in the query, reducing potential confusion for the LLM.

4.5 Generation of Answers

The final answer generation involves:

1. **Prompt Construction:** Relevant prompt components (user context string, combined instructions including rules for counting, formatting, and answering based *only* on context) are retrieved using get_prompt_components from chat_manager.py.
2. **QA Chain Invocation:** The final QA chain (create_qa_chain_component using create_stuff_documents_chain) is invoked in app.py. It receives a dictionary containing:
    - The original user_input.
    - The chat_history.
    - The filtered_docs as the context.
3. **LLM Call:** The QA chain formats these inputs according to its prompt template and sends the request to the configured LLM (ChatGoogleGenerativeAI - Gemini 1.5 Flash).
4. **Response:** The LLM generates the answer based solely on the provided filtered context, history, and instructions. This answer string is returned and displayed to the user via the Streamlit interface.

4.6 Implementation of Role-Based Access Control (RBAC)

RBAC is implemented through several mechanisms:

1. **Authentication:** Users log in (login_page), and their role and department are stored in st.session_state.current_user.
2. **UI Control:** Streamlit sections for actions like uploading marks are conditionally rendered based on current_user['role'] (e.g., only for 'Professor', 'HOD').
3. **Data Source Filtering:** The select_indexes_for_user function restricts which *types* of MongoDB data (collections/indexes) are accessible based on role.
4. **Retrieval Filtering:** For non-HOD roles, metadata filters ({"department": user_dept}) are applied directly within the FAISS retriever's search_kwargs for MongoDB professor and student data, ensuring only same-department information is retrieved. Similar filtering could be applied to the marks retriever.
5. **Write Permission Check:** The upload_marks_file_to_gridfs function explicitly checks if current_user['role'] is 'Professor' or 'HOD' before allowing the GridFS write operation.

4.7 Technology Stack

The system utilizes the following core technologies:

- **Programming Language:** Python 3.11.0
- **Web Framework:** Streamlit (for interactive UI)
- **Orchestration Framework:** Langchain (for RAG pipeline components)
- **LLM:** Google Gemini 1.5 Flash (accessed via langchain_google_genai.ChatGoogleGenerativeAI)
- **Embedding Model:** Google models/embedding-001 (via langchain_google_genai.GoogleGenerativeAIEmbeddings)
- **Vector Store:** FAISS (via langchain_community.vectorstores.FAISS) for local storage and similarity search.
- **Database:** MongoDB (for core college data and GridFS file storage) accessed via pymongo. GridFS is used for file storage.
- **Environment Management:** Conda virtual environment.
- **Dependencies:** python-dotenv, langchain-core, langchain-community, document loader libraries (pypdf, python-docx, unstructured, pandas, openpyxl, xlrd), streamlit, pymongo.

4.8 Environment of Development

The development was carried out using Python version 3.11.0. Project dependencies and the execution environment were managed using a Conda virtual environment (conda venv). Key libraries included specific versions of Langchain, Streamlit, Pymongo, FAISS, and Google Generative AI, installed via pip within the Conda environment.

V. RESULT AND DISCUSSION

5.1 Functional Verification

While formal quantitative benchmarks (such as precision, recall, F1-score, or large-scale user satisfaction surveys) were not conducted within the scope of this project, the core functionalities of the Levelwise RAG Chatbot were verified through iterative testing and observation during development. Key capabilities were confirmed as follows:

- **Role-Based Access Control (RBAC):** RBAC effectiveness was verified by logging in using credentials corresponding to each defined role (HOD, Professor, Student). Queries were specifically designed to test access boundaries. For example, Student and Professor accounts were confirmed to retrieve data (e.g., student lists, professor lists) only from their assigned department (ME in testing examples) due to the successful application of metadata filters in the FAISS retrievers. Conversely, the HOD role demonstrated access to data across multiple departments (implicitly, by having access to all MongoDB indexes).
- **Marks Data Workflow (GridFS & FAISS):** The end-to-end process for handling department-specific marks files was validated. This involved:
    - Successfully uploading an Excel file (ME_Marks.xlsx) via the Professor/HOD interface, which triggered storage in MongoDB GridFS using the upload_marks_file_to_gridfs function.
    - Confirming the successful processing of this file by embedding_manager.py to create and save a department-specific FAISS index (marks_index_ME.faiss) to local disk.
    - Verifying that the application correctly loaded this persistent index from disk for users in the relevant department (ME) using FAISS.load_local (with allow_dangerous_deserialization=True).
    - Confirming that the retriever for this marks index was added to the EnsembleRetriever.
- **RAG Pipeline for Marks QnA:** Student users successfully queried the chatbot for specific marks contained within the uploaded and indexed Excel file (e.g., querying for "Thermodynamics" marks yielded the correct value "82". This confirmed that the EnsembleRetriever was searching the marks index and providing the necessary context to the LLM.
- **Contextual Understanding & Formatting:** Through iterative refinement of the prompt instructions within chat_manager.py, the chatbot demonstrated improved handling of conversational context (pronoun resolution) and adherence to specified output formats (e.g., generating lists with items on separate lines, prefixed by hyphens).
- **Counting Accuracy:** Initial inconsistencies in LLM counting were addressed by implementing post-retrieval filtering (based on query keywords like "student" or "professor" within app.py Section 11) and refining prompt instructions, leading to accurate counts based solely on the appropriately filtered context passed to the LLM.

5.2 Qualitative Results

Qualitative analysis involved examining specific interactions across different user roles to assess functional correctness and user experience. Key observations include:

- **Example 1: Login Page for Users**

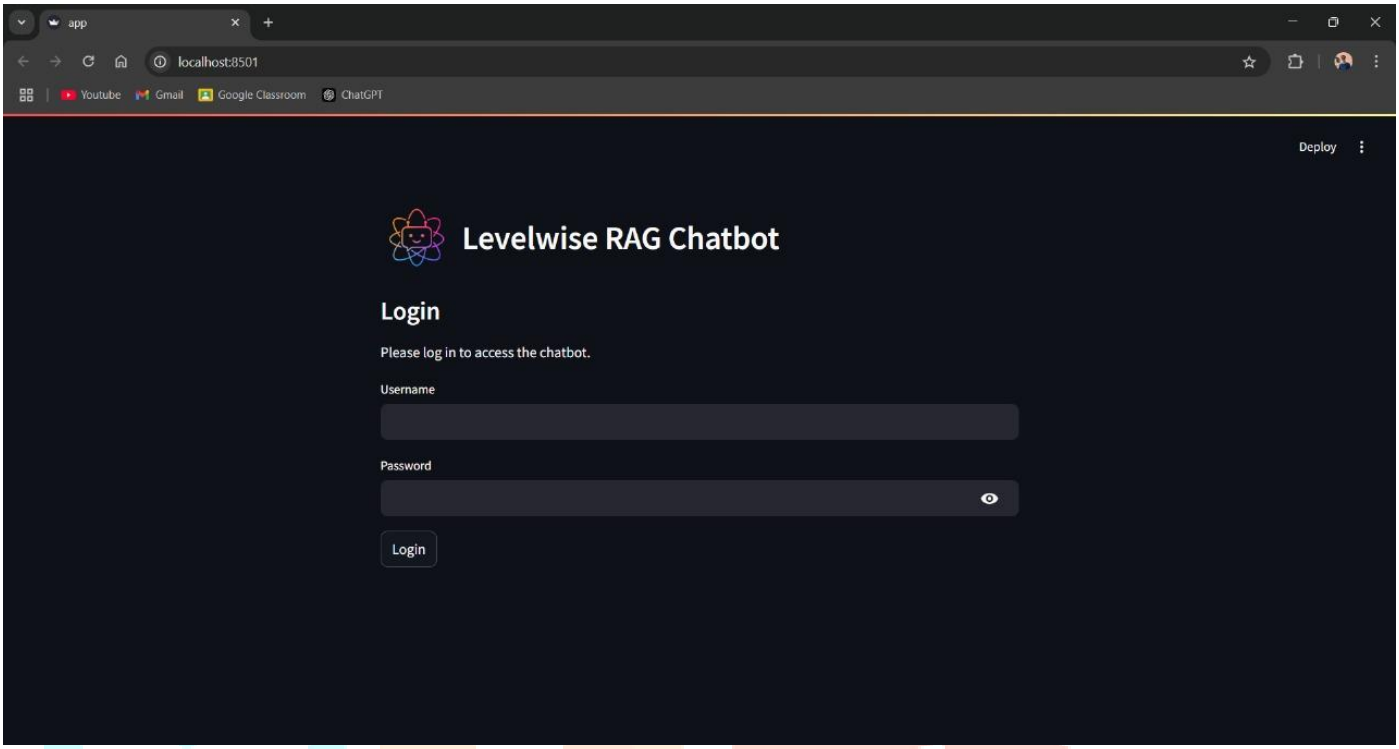


Fig.5.1- Chatbot login page for users

- **Example 2: RBAC in Action (Student)**

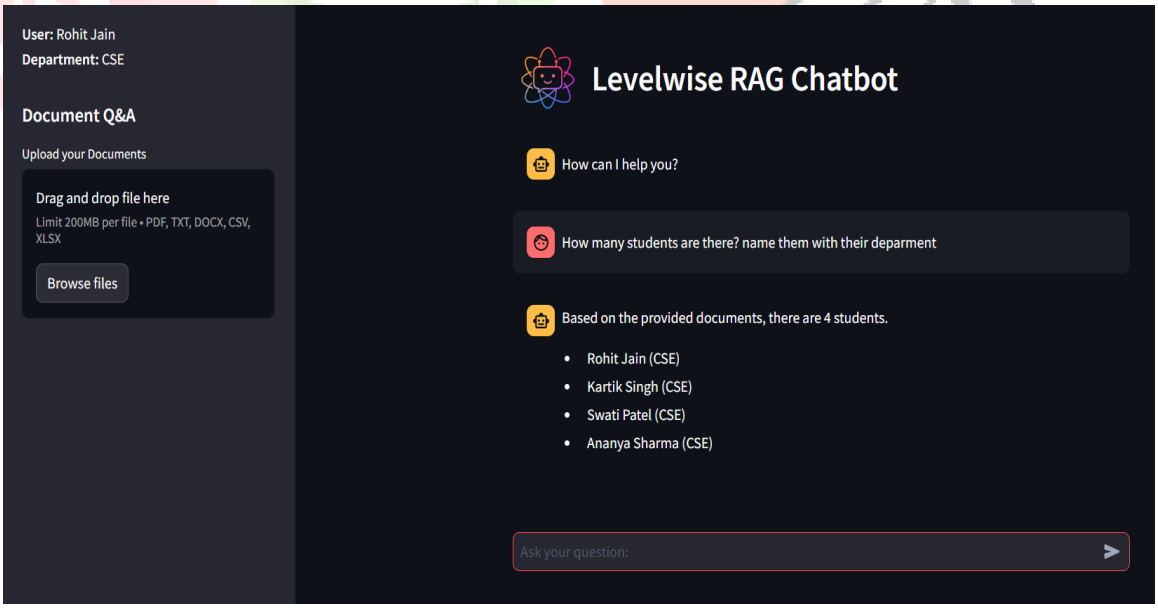
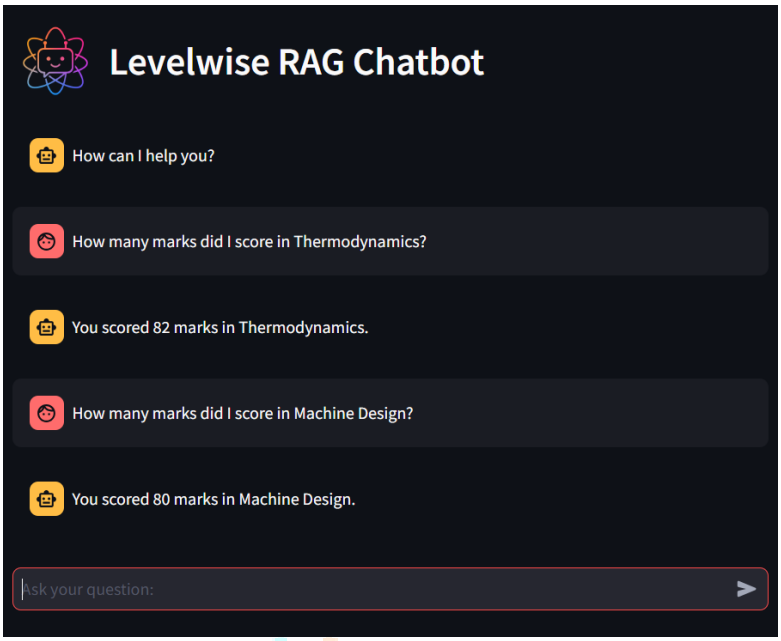


Fig.5.2- Chatbot replies names of students only of CSE branch

- **Example 2: Marks Query (Student)**




```
_id: ObjectId('67f920b264d0e20763ba2250')
filename : "ME_Marks.xlsx"
metadata : Object
chunkSize : 261120
length : 9205
uploadDate : 2025-04-11T14:01:22.427+00:00
```


303	Prateek Mishra	Thermodynamics	82
303	Prateek Mishra	Machine Design	80

Fig.5.3-Chatbot replying from a file uploaded in Database and actual data in the file

- **Example 3: HOD Access**

Fig.5.4-HOD has access to complete Database

- **Example 4:**

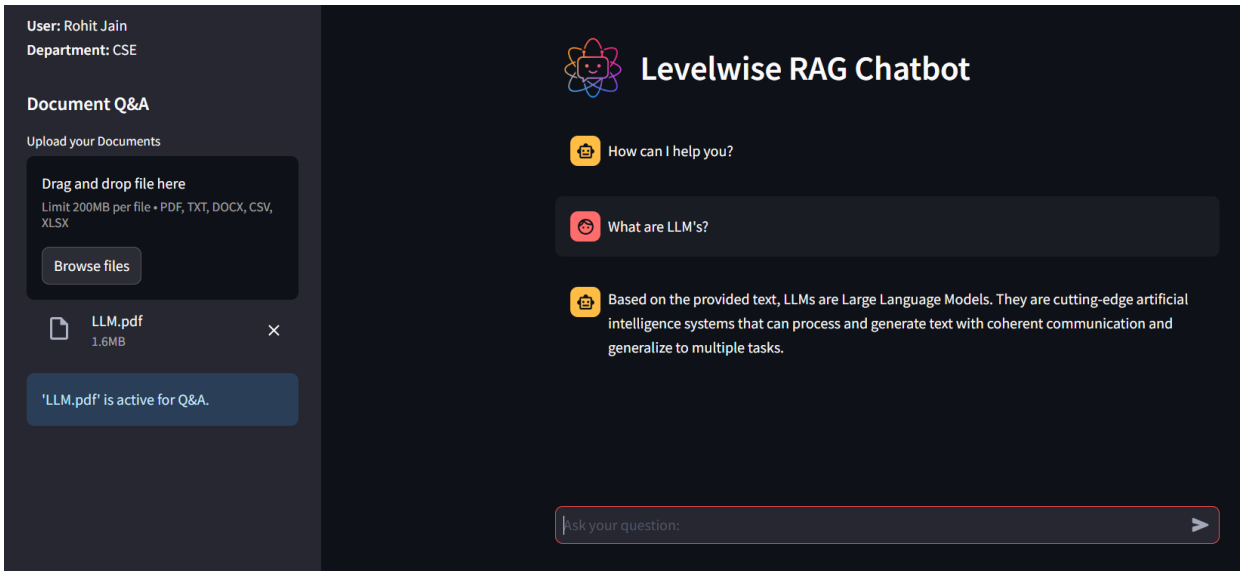


Fig.5.5-Chatbot answering from the document uploaded

5.3 Discussion

The results indicate that the developed Levelwise RAG Chatbot successfully addresses the core objective of providing secure, role-appropriate access to college information through a conversational interface.

- **Effectiveness of RAG:** The RAG architecture, combining retrieval from FAISS vector stores (sourced from MongoDB and uploaded files) with generation by the Gemini 1.5, proved effective in grounding responses in factual data and significantly reducing the likelihood of hallucinations compared to using an LLM alone. The inclusion of department-specific marks data via persistent FAISS indexes demonstrated the system's ability to integrate diverse, dynamically updated knowledge sources.
- **RBAC Implementation:** The multi-level RBAC mechanism, enforced through both selective index access and metadata filtering within FAISS retrievers, successfully restricted data visibility according to predefined roles (HOD, Professor, Student) and department affiliations. This is crucial for maintaining data privacy and security in an educational context.
- **Post-Retrieval Filtering & Prompt Engineering:** Initial challenges with LLM accuracy, particularly in counting tasks within mixed-context results, highlighted the importance of targeted strategies. The implemented post-retrieval filtering logic, which provides the LLM with only the specifically relevant document types (e.g., only student documents when asked to count students), proved highly effective in resolving these inaccuracies. This, combined with iterative prompt refinement focusing on explicit instructions for counting and formatting, was essential for achieving reliable performance.
- **System Strengths:** The system benefits from a modular design (app.py, mongo_handler.py, embedding_manager.py, chat_manager.py), facilitating maintenance and potential future extensions. The use of FAISS allows for efficient retrieval, while Streamlit provides a user-friendly suitable for non-technical users. The integration of GridFS and persistent FAISS indexes allows for the handling of specific datasets like departmental marks.

- System architecture diagram

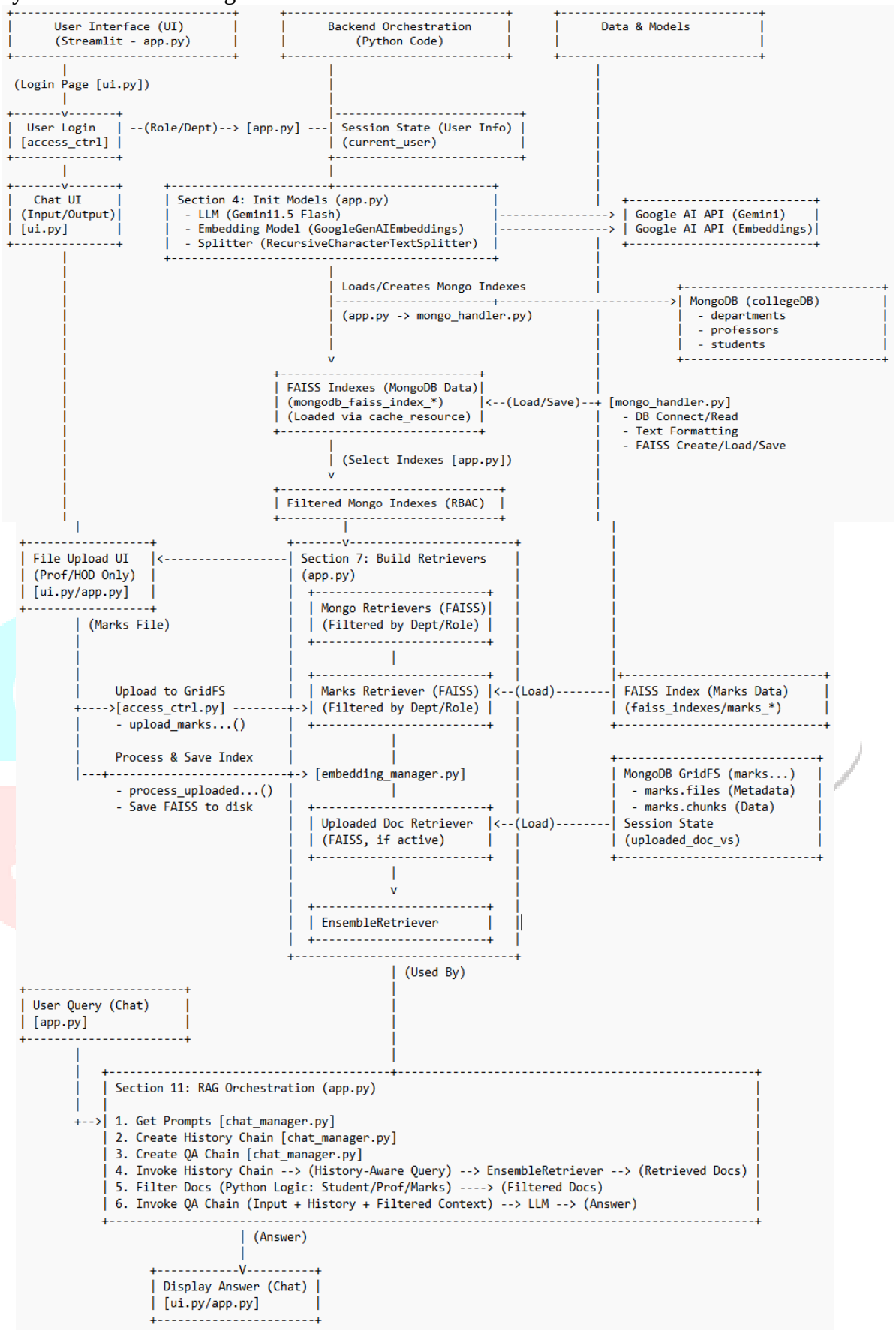


Fig.5.6-Complete architecture of Levelwise RAG Chatbot

VI. Conclusion

This paper presented the design, implementation, and functional verification of the Levelwise RAG Chatbot, a system developed to address the challenges of secure and efficient information retrieval within a college environment. The primary goal was to create an intuitive conversational interface enabling users with varying roles—Students, Professors, and Heads of Departments (HODs)—to access institutional data stored in MongoDB without requiring technical query language expertise.

The proposed system successfully integrates a Retrieval-Augmented Generation (RAG) architecture with Role-Based Access Control (RBAC). By leveraging LangChain for orchestration, FAISS for efficient vector searching across data sourced from MongoDB collections and uploaded files (including marks sheets stored via GridFS), and a capable Large Language Model ([Specify Final LLM, e.g., Gemini 1.5 Flash]), the chatbot provides contextually relevant and factually grounded responses while minimizing hallucinations. The implemented RBAC mechanisms, including index selection and metadata filtering at the retrieval stage, effectively restrict data access based on user roles and departmental affiliations, ensuring data privacy and security.
Qualitative testing confirmed the system's core functionalities. It demonstrated the ability to enforce role-based permissions, accurately retrieve specific information (such as student marks) from department-specific files uploaded by authorized personnel, and handle conversational context across multiple turns. The implementation of post-retrieval filtering and iterative prompt engineering proved crucial in enhancing the LLM's accuracy for specific tasks like counting and adhering to desired output formats.

The modular architecture, separating concerns between UI (streamlit, ui.py), data handling (mongo_handler.py, embedding_manager.py, access_control.py), and RAG logic (chat_manager.py, langchain), provides a solid foundation for maintainability and future extensions. While formal quantitative evaluation was outside the scope of this project, the functional verification indicates the system's potential to significantly improve information accessibility for non-technical users within the college.

Limitations include the current strategy for managing persistent marks indexes (potential overwriting) and the need for more robust parsing of complex file types like spreadsheets. Future work could focus on implementing index versioning or merging, enhancing document parsing capabilities, conducting large-scale usability testing with quantitative metrics, and potentially expanding the chatbot's capabilities to include secure write operations or more complex analytical queries.

In conclusion, the Levelwise RAG Chatbot demonstrates a practical and effective approach to building secure, context-aware, and role-specific conversational AI systems for domain-specific information retrieval, offering a valuable tool for enhancing data democratization within educational institutions.

VII. Future Works

While the current system effectively demonstrates role-based information retrieval using RAG, future work will focus on addressing identified limitations and expanding its functionality.

- **Excel Parsing:** As noted, the current UnstructuredExcelLoader might load entire sheets as single documents, potentially limiting the granularity of retrieval for specific marks. A more robust parser (e.g., using pandas within embedding_manager.py to extract specific rows/cells into meaningful chunks) could improve performance on detailed spreadsheet queries.
- **Marks File Management:** The current implementation saves marks indexes as marks_index_[Dept].faiss, potentially overwriting previous uploads for the same department. A more sophisticated strategy (e.g., using timestamps, subjects, or versioning in filenames/metadata; or merging FAISS indexes) is needed for handling multiple marks files per department robustly.
- **Scalability:** While FAISS is efficient, performance with a very large number of documents or users might require further optimization or transition to enterprise-grade vector databases. Embedding speed for large uploads, while improved with batching, could still be a factor.
- **LLM Dependence:** The system's accuracy is still dependent on the chosen LLM's ability to follow instructions and synthesize information from the provided context. Different LLMs (e.g., Gemini 1.5 Pro vs. Flash) might yield different results.

VIII. Acknowledgment

We take this opportunity to express our sincere thanks to Prof. Sachin Charbhe for his precious guidance, mentorship, and ongoing encouragement during the period of this project. We also appreciate the Department of Artificial Intelligence and Data Science, Rizvi College of Engineering, Mumbai, for affording the appropriate infrastructure and resources to conduct this research.

Special thanks to all the contributors who helped with testing and gave useful feedback during development. Their suggestions were important in helping us improve the system. We also thank the open-source communities of LangChain, FAISS, Streamlit, and Google AI for their efforts that enabled this work.

IX. References

[1] Google AI, "Google AI Studio," [Online]. Available: https://ai.google.dev/studio. (Accessed: Sep. 2, 2024).
[2] LangChain, "LangChain documentation," [Online]. Available: https://python.langchain.com/. (Accessed: Sep. 2, 2024).
[3] FAISS, "Facebook AI similarity search," [Online]. Available: https://github.com/facebookresearch/faiss. (Accessed: Sep. 2, 2024).
[4] Google AI, "Gemini 1.5 flash overview," [Online]. Available: https://ai.google.com/. (Accessed: Sep. 3, 2024).
[5] MongoDB, "GridFS manual," [Online]. Available: https://www.mongodb.com/docs/manual/core/gridfs/. (Accessed: Feb. 10, 2025).
[6] Streamlit, "Streamlit: The fastest way to build data apps," [Online]. Available: https://streamlit.io/. (Accessed: Oct. 2, 2024).
[7] A. Auffarth, *Generative AI with LangChain: Build large language model (LLM) apps with Python, ChatGPT, and other LLMs*. Packt Publishing, 2023.
[8] A. Kate, S. Kamble, A. Bodkhe, and M. Joshi, "Conversion of natural language query to SQL query," in *2018 Second Int. Conf. on Electronics, Communication and Aerospace Technology (ICECA)*, Coimbatore, India, 2018, pp. 488-491.
[9] T. O. Topsakal and T. C. Akinci, "Creating large language model applications utilizing LangChain," *International Conference on Applied Engineering and Natural Sciences*, vol. 1, no. 1, 2023.
[10] M. Khorasani *et al.*, *Web application development with Streamlit: Develop and deploy secure and scalable web applications to the cloud using a pure Python framework*. Apress, 2022.
[11] A. Nicoomanesh, "Creating local LLM-powered research assistant chatbot with Gemma 2 and LangChain," *Medium*, Oct. 23, 2024. [Online]. Available: https://medium.com/@anicomanesh/creating-llm-powered-research-assistant-with-langchain-faiss-and-gemma-2-878d1a15fd3b. (Accessed: Oct. 30, 2024).
[12] R. Akkiraju *et al.*, "FACTS about building retrieval augmented generation-based chatbots," *arXiv preprint arXiv:2407.07858*, Jul. 2024. [Online]. Available: https://arxiv.org/abs/2407.07858. (Accessed: Jul. 30, 2024).
[13] S. Gupta, R. Ranjan, and S. N. Singh, "A comprehensive survey of retrieval-augmented generation (RAG): Evolution, current landscape and future directions," *arXiv preprint arXiv:2410.12837*, Oct. 2024. [Online]. Available: https://arxiv.org/abs/2410.12837. (Accessed: Nov. 15, 2024).
[14] E. Planas, S. Martínez, M. Brambilla, and J. Cabot, "Modeling and enforcing access control policies in conversational user interfaces," *Software and Systems Modeling*, vol. 23, no. 1, pp. 45–60, Jan. 2024. [Online]. Available: https://modeling-languages.com/chatbot-access-control-conversational-user-interfaces/. (Accessed: Aug. 8, 2024).
[15] A. Balakrishnan, "Retrieval Augmented Generation (RAG) and LLMOps for Database Automation," Presentation, American Water, Jul. 2024. Available: https://www.researchgate.net/publication/382490238 (Accessed: Oct. 17, 2024).
[16] V. Bhat, S. D. Cheerla, J. R. Mathew, N. Pathak, G. Liu, and J. Gao, "Retrieval Augmented Generation (RAG) based Restaurant Chatbot with AI Testability," in *Proc. Int. Conf. on AI Applications*, Jul. 2024. Available: https://www.researchgate.net/publication/381461839 (Accessed: Aug. 20, 2024).
[17] R. Akkiraju, S. Srinivasan, S. Garg, A. Srinivas, and A. Rajaram, "FACTS About Building Retrieval Augmented Generation-Based Chatbots," *arXiv preprint arXiv:2407.07858*, Jul. 2024. Available: https://arxiv.org/abs/2407.07858 (Accessed: Jan. 20, 2025).

[18] A. Borucki, "Boosting AI: Build Your Chatbot Over Your Data With MongoDB Atlas Vector Search and LangChain Templates Using the RAG Pattern," *MongoDB Developer Center*, Sep. 18, 2024. Available: https://www.mongodb.com/developer/products/atlas/boosting-ai-build-chatbot-data-mongodb-atlas-vector-search-langchain-templates-using-rag-pattern/ (Accessed: Dec. 28, 2024).
[19] M. Lynn, "Build a RAG-Enabled Helpdesk Chatbot in 10 Minutes with MongoDB," *Dev.to*, Mar. 2025. Available: https://dev.to/michael_lynn_a7c09439545e/build-a-rag-enabled-helpdesk-chatbot-in-10-minutes-with-mongodb-4m52 (Accessed: Dec. 16, 2024).
[20] MongoDB Inc., "Taking RAG to Production with the MongoDB Documentation AI Chatbot," *MongoDB Developer Center*, Aug. 2024. Available: https://www.mongodb.com/developer/products/atlas/taking-rag-to-production-documentation-ai-chatbot/ (Accessed: Jan. 5, 2025).
[21] H. K. Jagannath, P. A. Venkatesh, and K. R. Rani, "Domain-Aware Conversational Agent Using Retrieval Augmented Generation and LangChain," in *2024 IEEE 7th International Conference on Computing, Communication and Security (ICCCS)*, Gandhinagar, India, 2024, pp. 1-6, doi: 10.1109/ICCCS60302.2024.10561020.
[22] Q. Zeng, J. Xu, and H. Li, "Natural Language Query to NoSQL Generation Using Query-Response Model," in *2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, 2019, pp. 5638-5647, doi: 10.1109/BigData47090.2019.8995767.
[23] S. Al-Fedaghi and W. Alghanim, "Intelligent Role-Based Access Control Model and Framework Using Semantic Business Roles in Multi-Domain Environments," in*2019 IEEE International Conference on Big Data (Big Data)*, Los Angeles, CA, USA, 2019, pp. 3814-3821, doi: 10.1109/BigData47090.2019.8954638.