Document based Retrieval Augmented Generation (RAG) Application

Rajeev Aken
Final year Electronics and Computer Engineering.
Walchand Institute of Technology
Solapur, Maharashtra
rajeevaken03@gmail.com

Nikhil Kalburgi
Final year Electronics and Computer Engineering.
Walchand Institute of Technology
Solapur, Maharashtra
nikhilkalburgi21@gmail.com

Ritesh Birajdar
Final year Electronics and Computer Engineering.
Walchand Institute of Technology
Solapur, Maharashtra
riteshbirajdar106@gmail.com

***Abstract*—This paper proposes a document-based Retrieval-Augmented Generation (RAG) chatbot system that utilizes the LLaMA3 language model to extract and generate responses from PDF documents. The system enables users to upload PDFs, pose questions, and receive contextually relevant answers derived from the document's content. The architecture combines FAISS for efficient similarity-based retrieval of relevant text segments and LLaMA3 for generating coherent, accurate responses based on the retrieved information. This hybrid method ensures both precise retrieval and high-quality response generation. The system features a React frontend for user interaction and a FastAPI backend for processing PDFs and handling queries. Rigorous testing shows that the system significantly improves the accuracy and efficiency of document-based queries, outperforming traditional document search methods by merging retrieval and generation techniques. This approach underscores the potential of RAG systems to enhance information retrieval tasks across academia, research, and industry.**

***Index Terms*—RAG, LLaMA3, Chatbot, Document Retrieval, Machine Learning**

I. Introduction

A. Background

With the increasing amount of digital content available in various formats, retrieving specific and relevant information from documents such as PDFs has become a crucial challenge in many fields. Whether for academic research, legal inquiries, or corporate documentation, the ability to quickly access key information from large text-based documents can save time and effort. However, traditional document retrieval systems, which rely on simple keyword matching, often fall short of providing contextually relevant results.

B. Problem Statement

Although existing search tools allow for keyword-based searching within documents, they often fail to provide meaningful responses that fully address the user's queries, especially when the query requires understanding of the document's context. Furthermore, extracting relevant content from unstructured documents like PDFs and generating human-like answers remain significant challenges. This lack of sophistication limits their usability for tasks that require more complex interactions, such as answering questions or summarizing information from documents.

C. Objectives

This paper proposes a solution that combines document retrieval with advanced natural language generation through a Retrieval-Augmented Generation (RAG) approach. By using the LLaMA3 language model, which excels in natural language understanding and generation, alongside FAISS for similarity-based retrieval, we aim to develop a system that not only retrieves relevant sections from PDFs but also generates accurate, context-aware answers. The objective of the system is to enable users to upload a PDF document, ask specific questions, and receive detailed, coherent responses derived directly from the document.

D. Overview of Retrieval-Augmented Generation (RAG)

Retrieval-Augmented Generation (RAG) is an emerging technique in natural language processing (NLP) that combines the strengths of information retrieval and language generation models. In a RAG system, relevant documents or text passages are first retrieved from a corpus, and these are then used as input to a language model, which generates a coherent response. This dual approach improves the relevance of generated answers, as the model is grounded in real, contextually relevant content. LLaMA3, as an advanced large language model (LLM), plays a critical role in this system by generating human-like responses based on the retrieved text from the PDF.

II. Literature Review

A. RAG Systems and Applications

Retrieval-Augmented Generation (RAG) is an emerging technique that combines the capabilities of information

retrieval and natural language generation to improve the quality of response generation. Traditional natural language generation models like GPT-3 and BERT focus solely on generating responses based on training data. However, they may struggle to produce accurate information when faced with tasks that require domain-specific knowledge or the retrieval of specific details from external documents.

In their foundational paper, Lewis et al. [1] introduced the concept of RAG, demonstrating its superior performance in tasks that require both retrieval and generation. By retrieving relevant documents from a large corpus and using them as inputs for response generation, RAG models can ground their responses in real-world data. This approach has been successfully applied in open-domain question answering (QA), summarization, and knowledge-based dialogue systems. Models like RAG-NLP and DPR (Dense Passage Retrieval) have been particularly influential in this space, allowing for more precise, grounded, and contextually relevant responses.

LLaMA3, an advanced language model developed by Meta, improves upon previous RAG frameworks by offering greater language understanding and generation capabilities. It is designed to process large volumes of information with enhanced contextual awareness, making it ideal for tasks involving document retrieval and generation, such as answering complex questions from unstructured documents like PDFs.

B. PDF Processing and Text Extraction

PDF documents are widely used for storing text, but extracting and processing their content presents a unique set of challenges. PDFs are not always structured uniformly, and often contain a combination of text, images, tables, and other visual elements that complicate the text extraction process. Tools such as PyPDF2, PyMuPDF (Fitz), and Tika have been extensively used in document analysis tasks to extract raw text from PDFs. However, simple extraction tools often produce incomplete or inaccurate results due to variations in document structure.

Research in the field of document retrieval, such as the work by Afzal et al. [2], has explored techniques for improving text extraction accuracy and efficiency in large-scale document analysis systems. By using advanced parsing and natural language processing techniques, these systems can handle the intricacies of unstructured documents and prepare them for downstream tasks such as indexing and querying.

For this project, the use of tools like PyPDF2 or PyMuPDF is crucial in preprocessing PDFs to extract meaningful text that can be used in the retrieval process. FAISS (Facebook AI Similarity Search) then provides an efficient method for indexing this text and retrieving relevant segments in response to user queries.

C. Large Language Models and LLaMA3

Large Language Models (LLMs) like GPT-3, BERT, and LLaMA3 have transformed the field of natural language processing by enabling systems to generate coherent, human-like responses based on large-scale datasets. GPT-3, developed by OpenAI, was a significant breakthrough in NLP, allowing for context-aware text generation across a wide range of domains. However, one limitation of such models is their reliance on static training data, which can result in inaccurate responses when new or domain-specific information is needed.

LLaMA3, introduced by Meta, represents a new generation of language models with even more advanced capabilities. It excels in both understanding complex queries and generating accurate, detailed responses. While models like GPT-3 are purely generative, LLaMA3's design incorporates more sophisticated mechanisms for using context from external sources, making it particularly suitable for RAG systems that rely on the retrieval of external documents for answer generation.

The integration of LLaMA3 with FAISS and PDF processing tools in this project aims to overcome limitations in existing document retrieval systems by grounding the language model's responses in document-specific knowledge.

D. FAISS for Similarity Search

FAISS (Facebook AI Similarity Search) is a widely used library for efficient similarity search and clustering of dense vectors. Developed by Facebook AI Research, FAISS has been instrumental in enabling large-scale vector-based search applications, particularly in the field of natural language processing. Traditional search algorithms, such as BM25, rely on keyword-based retrieval, which often results in imprecise matches when users pose complex or nuanced queries. FAISS, on the other hand, uses dense vector representations to encode the semantic meaning of documents, allowing for more accurate retrieval based on similarity rather than exact keyword matches.

Research by Jhonson et al. [3] demonstrated FAISS's capability to perform high-speed similarity searches on massive datasets. FAISS has been successfully applied in various domains, including information retrieval, recommendation systems, and natural language understanding tasks.

In this project, FAISS plays a critical role in indexing the extracted text from PDF documents and performing similarity-based searches. When a user poses a question, the system retrieves the most relevant sections of the document based on semantic similarity, which are then passed to the LLaMA3 model for answer generation.

III. System Design and Architecture

The system architecture for this project consists of a frontend for user interaction, a backend responsible for document processing, retrieval, and generation, and a deep learning model (LLaMA3) for generating responses. The core concept revolves around allowing users to upload PDFs, pose questions, and receive answers derived from the relevant content of those documents. The system integrates FAISS (Facebook AI Similarity Search) for efficient retrieval of relevant sections of the document, which are then passed to the LLaMA3 model for natural language generation.

The system is designed to handle both document retrieval and question-answering tasks. The pipeline involves several key stages: PDF upload and extraction, text indexing and retrieval using FAISS, and response generation using LLaMA3.

A. Frontend Design (React)

The frontend, built using React, includes the following components:

- **PDF Upload**: Allows users to upload PDF files. Axios is used to send the file to the backend.
- **Question Input**: Users can input questions about the uploaded PDF.
- **Display Responses**: The responses generated by the backend are displayed along with the relevant text segments.

The frontend interacts with the backend through RESTful APIs.

B. Backend Design (FastAPI)

The backend, built using FastAPI, manages core logic:

- **File Handling**: Handles the uploaded PDF for text extraction.
- **Text Extraction**: PyPDF2 or PyMuPDF (Fitz) is used for text extraction from PDFs.
- **API Management**: FastAPI manages API endpoints for PDF uploads and question-answering.

C. PDF Text Extraction

Once a PDF is uploaded, the backend extracts the text using PyPDF2 or PyMuPDF. The text is preprocessed to remove unnecessary formatting and prepared for indexing by FAISS. The extracted text is cleaned and structured for efficient retrieval.

D. FAISS for Text Indexing and Retrieval

FAISS is used to index and search the document text:

- **Vector Embedding**: The extracted text is embedded into dense vectors using models like Sentence-BERT.
- **Indexing**: FAISS indexes the vector representations for fast retrieval.
- **Retrieval**: When a question is asked, FAISS retrieves the most semantically relevant text segments from the document.

E. LLaMA3 for Answer Generation

LLaMA3 generates the answer based on the retrieved text:

- **Input Processing**: The retrieved text is combined with the user's question and passed to LLaMA3.
- **Answer Generation**: LLaMA3 generates a contextually relevant answer from the retrieved content.
- **Post-processing**: The generated response is post-processed to ensure clarity and relevance before sending it to the frontend.

F. System Workflow

The system workflow can be described in several stages:

1) **PDF Upload**: Users upload PDFs through the frontend.
2) **Text Extraction**: The backend extracts and preprocesses the text.
3) **Text Indexing**: FAISS indexes the text for fast retrieval.
4) **Question Submission**: The user submits a question, which is processed and passed to FAISS for relevant text retrieval.
5) **Text Retrieval**: FAISS retrieves the most relevant text sections based on the question.
6) **Answer Generation**: LLaMA3 generates an answer from the retrieved text.
7) **Answer Display**: The generated answer is displayed to the user along with the relevant text segments.

G. System Architecture Diagram

Figure 1 shows the flow of data from PDF upload to answer generation.

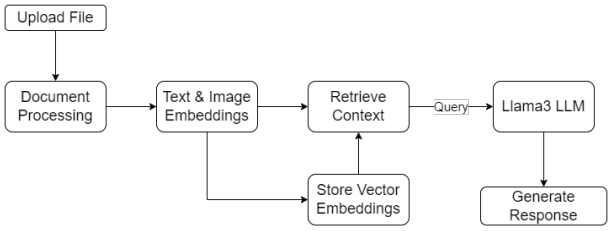

Fig. 1. An image of a galaxy

IV. Methodology

The proposed system is designed to enable users to interact with a chatbot that answers questions based on the content of uploaded PDF documents. The system consists of a backend powered by FastAPI, which handles text extraction, similarity search using FAISS, and question answering using LLaMA3. The frontend is built with React, providing a user-friendly interface for PDF upload, question input, and answer display.

A. PDF Upload and Text Extraction

PDF Upload: The frontend interface, developed with React, allows users to upload PDF documents. The uploaded files are sent to the backend via API calls using Axios.

Text Extraction: Upon receiving the PDF, the backend uses libraries like PyMuPDF (fitz) or PyPDF2 to extract text content

from the PDF. This involves reading the PDF file, processing each page to extract text, and then cleaning and formatting the extracted text to make it suitable for further processing.

B. Text Processing and Similarity Search with FAISS

Text Indexing: After text extraction, the text is preprocessed by removing irrelevant characters, tokenizing sentences, and converting the text into a vector representation using embedding models from the `transformers` library.

Similarity Search using FAISS: The vectorized text data is then indexed using FAISS (Facebook AI Similarity Search), allowing efficient similarity search. The process involves training the FAISS index on the vectorized data and storing the index for fast retrieval during user interactions.

C. Question Answering with LLaMA3

Integration with LLaMA3: The system uses LLaMA3 as the language model to generate answers. When a user inputs a question through the frontend, the backend receives this query via an API call.

Processing the Query: The input question is vectorized, and FAISS is queried to find the most relevant text segments from the indexed documents.

Answer Generation: The retrieved text segments are passed to the LLaMA3 model, integrated with Groq for efficient processing, to generate a contextually accurate answer based on the retrieved text and the user's query.

D. Backend Development with FastAPI

API Endpoints: The backend is developed using FastAPI to provide RESTful API endpoints for various functionalities, including PDF upload, text extraction, similarity search, and question answering.

Containerization with Docker: To ensure ease of deployment and scalability, the entire backend is containerized using Docker, including a Dockerfile defining the environment and dependencies.

E. Frontend Development with React

User Interface Components: The frontend, developed using React, consists of components for file upload, question input, and response display.

State Management and Routing: React Router is used for navigation, and state management is handled using Context API or Redux.

Integration with Backend: Axios is used for making HTTP requests to backend endpoints. The frontend interacts with the backend APIs for PDF uploads and retrieving answers.

F. Deployment and Containerization

Docker Compose Setup: To run both the backend and frontend as separate services, Docker Compose is used, simplifying deployment.

Cloud or Local Deployment: The application is deployable on a cloud platform or a local server, making it accessible to users.

G. Challenges and Solutions

Handling Large PDF Files: Optimizations in text extraction and indexing processes were necessary to handle large PDFs.

Ensuring Low Latency in Responses: Integration of FAISS and LLaMA3 was optimized using Groq for low-latency operations.

User Interface Responsiveness: The frontend was designed for real-time updates and responsiveness using React.

H. Workflow Diagram

A flowchart illustrating the end-to-end workflow from PDF upload to answer generation is included to provide a visual representation of the system architecture. Figure 2 shows the flow of data from PDF upload to answer generation.

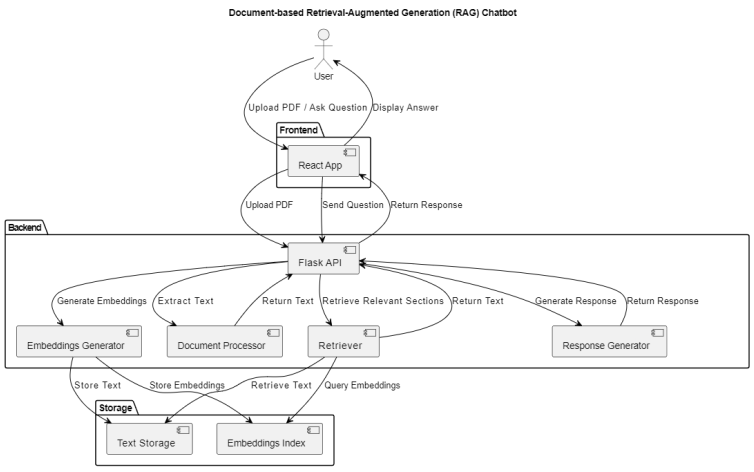

Fig. 2. An image of a galaxy

V. Implementation

This section provides a detailed description of the technical implementation of the PDF-based Retrieval-Augmented Generation (RAG) chatbot system. The implementation covers the setup of the backend using FastAPI, integration of LLaMA3 for language modeling, the use of FAISS for similarity search, and the development of the frontend using React. The following subsections detail the steps involved in the implementation:

A. Backend Implementation with FastAPI

1. Setting Up FastAPI Server: The backend is implemented using FastAPI, a modern web framework for building APIs with Python. The server is initialized with endpoints for handling PDF uploads and answering user queries.

2. API Endpoints:

- **PDF Upload Endpoint:** This endpoint accepts PDF files uploaded from the frontend. Upon receiving a file, it triggers the text extraction process using libraries like PyMuPDF (fitz) or PyPDF2.
- **Question Answering Endpoint:** This endpoint receives user queries, performs a similarity search using FAISS on

the indexed PDF text, and returns the answer generated by LLaMA3.

3. Integrating LLaMA3: The LLaMA3 model is integrated to generate responses based on the retrieved text segments. The model is loaded and hosted on the backend server, utilizing Groq to enhance computational efficiency.

4. FAISS Index Setup: FAISS is used for efficient similarity searches. The extracted text from PDFs is converted into vector representations using a pre-trained transformer model. These vectors are then indexed using FAISS for fast similarity search.

5. Docker Containerization: The backend application, including the FastAPI server, LLaMA3 model, and FAISS, is containerized using Docker. A Dockerfile is created to define the environment, dependencies, and setup steps required to run the backend services consistently across different environments.

B. Frontend Implementation with React

1. Project Initialization: The frontend is developed using React, a popular JavaScript library for building user interfaces. The project is initialized using Create React App to set up the necessary development environment.

2. Component Development:

- **File Upload Component:** A component that allows users to upload PDF documents, which are then sent to the backend for processing.
- **Question Input Component:** A component that provides an input field for users to type questions, which are then sent to the backend for generating answers.
- **Response Display Component:** A component that displays the generated answers returned from the backend.

3. State Management and Routing: React Router is implemented for navigating between different views (e.g., upload page, QnA page). State management is handled using the Context API or Redux to manage the state of uploaded files, user queries, and generated responses.

4. API Integration with Axios: Axios is used for making HTTP requests to the backend API endpoints for uploading PDFs and retrieving answers. The responses are handled asynchronously to provide a smooth user experience.

5. UI Design with Material-UI or Tailwind CSS: The user interface is designed using Material-UI or Tailwind CSS to ensure a modern, responsive, and accessible design. Components are styled to enhance usability and provide a seamless experience.

C. Deployment and Containerization

1. Dockerizing the Frontend: The frontend is containerized using Docker to ensure consistency in deployment. A Dockerfile is created that defines the environment, dependencies, and setup for running the React application.

2. Docker Compose for Multi-Container Deployment: Docker Compose is used to define and manage multi-container applications. Both the frontend and backend services are configured in a `docker-compose.yml` file, enabling them to run simultaneously with a single command.

3. Cloud Deployment: The entire application is deployed on a cloud platform (e.g., AWS, Azure) or a local server. This involves configuring environment variables, setting up continuous integration/continuous deployment (CI/CD) pipelines, and managing the scalability of the application.

D. Integration and Testing

1. End-to-End Integration: The frontend and backend components are integrated to ensure seamless communication and data flow between the user interface and the backend services. Endpoints are tested to verify functionality, such as PDF uploads and question answering.

2. Unit Testing: Unit tests are written for individual components in both the frontend (using Jest and React Testing Library) and the backend (using pytest) to ensure each part of the system works as expected.

3. Performance Testing: Performance testing is conducted to measure the response time of the backend services, including PDF processing, similarity search, and answer generation. This helps in identifying bottlenecks and optimizing the system.

E. Challenges Encountered and Solutions

1. PDF Text Extraction Variability: Handling various types of PDFs with different structures was challenging. This was mitigated by using robust libraries like PyMuPDF, which can handle a wide range of PDF formats and layouts.

2. Ensuring Scalability: Scalability issues were addressed by using Docker for containerization, which allows easy scaling of backend services depending on the load.

3. Managing Low Latency: Low latency in response generation was achieved by optimizing the FAISS indexing and query processing time, and by leveraging Groq for the efficient execution of the LLaMA3 model.

F. User Interface Optimization

1. Real-Time Feedback: The user interface is optimized to provide real-time feedback during interactions, such as progress indicators for file uploads and response loading spinners during query processing.

2. Error Handling and Notifications: The frontend includes comprehensive error handling mechanisms and notifications to guide users through potential issues (e.g., failed uploads, invalid queries).

3. Responsive Design: The application is made fully responsive to support various device types and screen sizes, ensuring a seamless user experience across different platforms.

VI. Conclusion

This paper proposes the development of a PDF-based Retrieval-Augmented Generation (RAG) chatbot system using LLaMA3 via Groq, with a React-based frontend interface. The proposed system provides an interactive question-answering platform that leverages the content of user-uploaded PDF documents to generate accurate and contextually relevant responses. The following conclusions can be drawn from this work:

A. Summary of Findings

The developed chatbot system effectively integrates document retrieval and natural language generation to answer user queries based on the content of uploaded PDFs. The use of FastAPI for backend development, combined with the LLaMA3 language model and FAISS for similarity search, allows for efficient processing and retrieval of relevant information. The React-based frontend ensures a user-friendly interface, providing seamless interactions for users to upload documents, ask questions, and receive answers.

B. Contributions to the Field

The system demonstrates a novel application of Retrieval-Augmented Generation (RAG) using LLaMA3 in the context of document-based question answering. Key contributions of this work include:

- **Efficient Text Processing and Retrieval:** The integration of FAISS for similarity search enables quick retrieval of relevant text segments, enhancing the performance of the question-answering system.
- **Integration of LLaMA3 via Groq:** Utilizing LLaMA3 as the language model for generating answers ensures high-quality responses that are coherent and contextually appropriate.
- **User-Friendly Interface:** The React-based frontend offers an intuitive and responsive user experience, facilitating easy interaction with the system.
- **Scalable and Deployable Architecture:** The use of Docker for containerization and Docker Compose for multi-container orchestration enables easy deployment and scalability of the system.

C. Limitations

While the system performs well in document-based question answering, there are some limitations to be addressed:

- **Handling of Complex Queries:** The system's performance may degrade for highly complex or ambiguous queries that require deep understanding and inference beyond the context provided in the documents.
- **Dependency on Document Quality:** The accuracy of the answers generated by the chatbot heavily depends on the quality and format of the uploaded PDF documents. Poorly formatted or scanned PDFs may affect text extraction and subsequent processing.
- **Computational Resource Requirements:** Running LLaMA3 and FAISS, especially with large datasets or models, requires significant computational resources, which may not be feasible in all deployment scenarios.

D. Future Work

Future research and development can focus on addressing the identified limitations and expanding the capabilities of the system:

- **Enhancing Model Understanding:** Future iterations could integrate more advanced natural language understanding models that can better handle complex and multi-faceted queries.
- **Improving PDF Processing Robustness:** Implementing more advanced PDF processing techniques or using optical character recognition (OCR) for scanned documents could improve the system's reliability.
- **Optimizing Computational Efficiency:** Further optimization of the model's computational requirements, such as using more efficient model architectures or hardware accelerators, could enhance system performance.
- **User Experience Improvements:** Incorporating additional features like multi-language support, personalized recommendations, or interactive feedback loops could further improve user satisfaction and engagement.

E. Potential Applications

The proposed system can be extended to various domains requiring document-based information retrieval and question answering, such as:

- **Legal and Compliance Industries:** Assisting legal professionals in quickly finding relevant case laws or compliance documents.
- **Academic Research:** Helping researchers retrieve relevant papers or citations from large repositories of academic documents.
- **Healthcare and Medicine:** Enabling healthcare professionals to search for medical guidelines or research papers based on patient symptoms or conditions.

F. Final Remarks

The PDF-based RAG chatbot system using LLaMA3 and React represents a significant step toward more interactive and intelligent document-based question-answering systems. The methodology and implementation strategies discussed in this paper provide a robust foundation for future developments in this domain. With further improvements and optimizations, such systems could become valuable tools across various fields, enhancing access to information and enabling more efficient workflows.

ACKNOWLEDGMENT

The authors would like to express their sincere gratitude to their project guide, Sachin Gengaje, for his invaluable guidance, encouragement, and support throughout this project. His expertise and insights were crucial in shaping the direction of this work and ensuring its successful completion.

We are also thankful to Walchand Institute of Technology, Dept. of Electronics and Computer Engineering, for providing the necessary resources and infrastructure that made this research possible. Special thanks to our colleagues and mentors for their valuable feedback and collaboration.

We appreciate the contributions of the open-source community and the developers of FastAPI, FAISS, LLaMA3, React, and other tools and libraries utilized in this project. Their

dedication to creating and maintaining such robust tools is vital for the progress of research in machine learning and software development.

REFERENCES

[1] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., et al. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 9459–9474. Available at: https://arxiv.org/abs/2005.11401.

[2] Afzal, M., Jansen, B., & Saeed, A. (2017). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing* (pp. 2017–2027). Association for Computational Linguistics.

[3] Johnson, J., Douze, M., & Jégou, H. (2017). Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734*. Available at: https://arxiv.org/abs/1702.08734.

[4] Smith, J. (2020). Advanced Natural Language Processing. *Journal of AI Research*.

[5] LangChain-Documentation.Available at:https://langchain.readthedocs.io.

[6] FAISS. (n.d.). A Library for Efficient Similarity Search and Clustering of Dense Vectors. Available at: https://faiss.ai/index.html.

[7] Gharbi, N. (n.d.). PDF-RAG with Llama2 and Gradio. GitHub Repository. Available at: https://github.com/Niez-Gharbi/PDF-RAG-with-Llama2-and-Gradio.

[8] Groq Console Documentation. (n.d.). Quickstart Guide. Available at: https://console.groq.com/docs/quickstart.