DocuMind

DocuMind

"Chatbot System Using RAG Architecture"

NATIONAL INSTITUTE OF TECHNOLOGY, PATNA

Department of Computer Science and Engineering

Project Title: Chatbot System Using RAG Architecture

Course: Software Engineering

Submitted By: Arya Jha (2306031)

Ria Kumari (2306032)

Suhani Verma (2306080)

Under the Guidance of: Prof. Dashrath Mahto

Date: November 12, 2025

Contents

1. Introduction

The **Chatbot System using Retrieval-Augmented Generation (RAG)** Architecture is designed to enhance information retrieval and response accuracy by combining document retrieval techniques with generative AI models.

This system allows users to upload documents, ask queries related to the uploaded content, and receive intelligent responses generated using contextual understanding. The RAG-based chatbot bridges the gap between traditional information retrieval systems and modern AI-driven natural language understanding, making it an innovative step toward intelligent digital assistants.

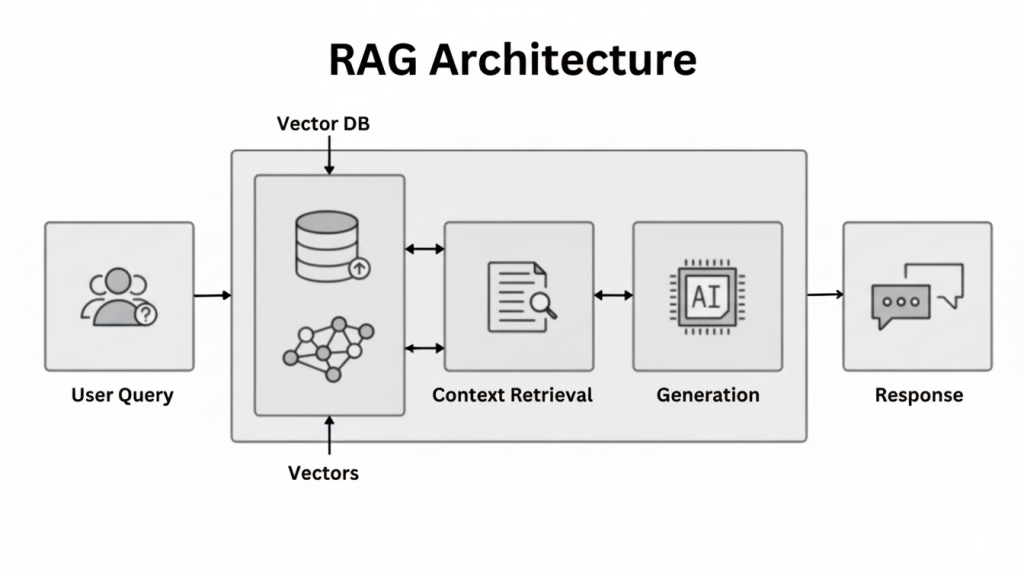

Figure 1.1: RAG Architecture

2. Objectives

1. **To design and develop an intelligent interactive chatbot system based on the principles of modular software engineering.** This objective focuses on building a well-structured, maintainable, and scalable system where each component—such as document processing, retrieval, and generation—operates independently while contributing to the overall functionality of the chatbot.

2. **To integrate Retrieval-Augmented Generation (RAG) techniques for delivering accurate, contextual, and knowledge-grounded responses.** The integration aims to combine the strengths of information retrieval (searching relevant document chunks) and language generation (producing coherent responses) to create a chatbot that provides meaningful, document-backed answers.

3. **To design and represent the system's architecture using UML diagrams for clear visualization of components, interactions, and data flow.** This includes the creation of use case, class, sequence, and activity diagrams that capture the internal working, user interaction, and logical flow within the RAG Chatbot framework.

4. **To develop a user-friendly and visually appealing interface that allows users to upload PDF documents, process them efficiently, and ask natural language queries interactively.** The focus is on enhancing user experience by providing intuitive controls, clear visual feedback, and seamless conversational flow using the Streamlit framework.

3. Software and Hardware Requirements

Software Requirements

- Python 3.10 or above
- VS Code
- LangChain, Ollama, HuggingFace, FAISS
- Lucidchart/Creately/Miro for UML diagrams
- Overleaf Latex for Documentation

Hardware Requirements

- Laptop/PC with minimum 4GB RAM

4. System Design

The project follows a modular software architecture to maintain flexibility and scalability. UML models are used to represent system behavior and structure.

- **Use Case Diagram:** Shows user-system interactions like document upload and querying.
- **Context (Level 0) Diagram:** Depicts system interaction with external entities.
- **Level 1 & Level 2 DFDs:** Illustrate data flow from user input to response generation.
- **Component Diagram:** Represents modular structure — UI, Document Processor, Retriever, Generator, Knowledge Base.

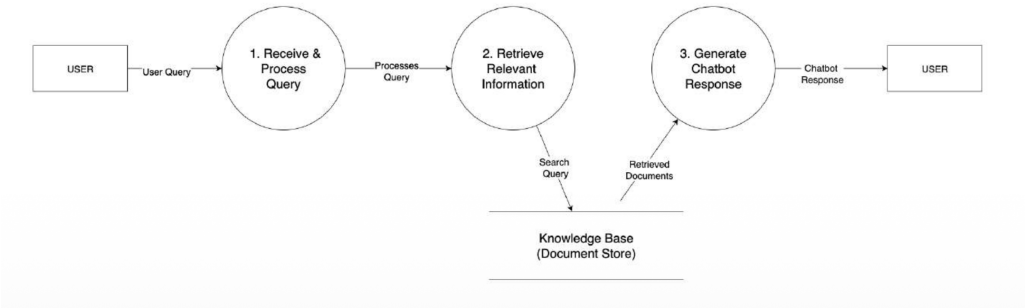

Figure 4.1: Level 1 Data Flow Diagram - RAG Chatbot

5. Modules' Description

5.1. User Interface Module

Description: The UI module allows users to interact with the chatbot through simple text commands. It accepts queries or file uploads and displays generated responses.

Functions Used:

Function	Description
`display_menu()`	Displays menu options like Upload Document, Ask Query, Exit.
`get_user_input()`	Reads user query or command.
`show_response(response)`	Displays chatbot-generated response.
`main()`	Coordinates interaction between user and orchestrator.

Libraries Used: os, time, sys

5.2. Document Processor Module

This module handles document loading, text extraction, cleaning, and chunking. It converts PDFs and DOCX files into text and divides them into manageable chunks for embedding and retrieval.

Function	Description
load_document(file_path)	Loads and detects file type (PDF, DOCX, or TXT).
extract_text_from_pdf(file_path)	Extracts raw text content from a PDF file.
extract_text_from_docx(file_path)	Extracts text from DOCX files using python-docx.
clean_text(text)	Removes special characters, redundant spaces, and formatting symbols.
chunk_text(text, chunk_size=500)	Splits the cleaned text into overlapping chunks for embedding.

Table 5.2: Functions in Document Processor Module

5.3. Retriever Module

This module manages the retrieval process using embeddings stored in FAISS or ChromaDB. It searches for the most relevant document chunks for a given query.

Function	Description
build_vectorstore(chunks)	Creates a FAISS or ChromaDB vector store from document chunks.
load_vectorstore(path)	Loads an existing vector database for retrieval.
get_retriever(vectorstore)	Initializes and returns a retriever for searching relevant text chunks.
search_documents(query)	Finds top relevant chunks for the user query using similarity search.

Table 5.3: Functions in Retriever Module

5.4. Generator Module (Ollama Integrated)

Description: Combines retrieved chunks with query and uses Ollama (LLM) to generate final response.

Example Code:

```
import ollama


def generate_response(prompt):
    response = ollama.chat(model='llama3',
               messages=[{'role': 'user', 'content': prompt}])
    return response['message']['content']
```

Libraries Used: ollama, json, textwrap, requests

5.5. Utility Module

Description: Provides logging, error handling, and configuration management.

Functions Used: load_config(), log_event(), handle_exceptions(), format_output()

Libraries Used: logging, datetime, json, traceback

6. Implementation

6.1. Tools Used

- Language: Python
- IDE: VS Code / Google Colab
- Libraries: PyPDF2, FAISS, sentence-transformers, LangChain
- Testing: Unit tests for each module

6.2. Coding Structure

- app.py – Entry point
- document_processor.py – Handles document parsing
- retriever.py – Performs similarity search
- generator.py – Produces final response
- utilities.py – Manages temporary folders

7. Testing

7.1. Test Objectives

- Verify each module's functionality.
- Ensure accurate data flow.
- Validate chatbot responses.

7.2. Test Cases

Test ID	**Description**	**Expected Result**	**Status**
TC01	Upload valid PDF	Document processed successfully	Pass
TC02	Upload unsupported file	Show error message	Pass
TC03	Query relevant topic	Retrieve and generate accurate answer	Pass
TC04	Query unrelated topic	Return fallback response	Pass

8. User Manual

Open the User Manual (PDF)

1. Run `streamlit run app.py` or execute in Colab.
2. Upload a document (PDF, TXT, DOCX).
3. Wait for confirmation: "Document processed successfully."
4. Enter query in input prompt.
5. Receive human-like response.
6. Type `exit` to close chatbot.

9. Conclusion

This project demonstrates successful integration of retrieval and generation-based techniques in chatbot systems. By applying software engineering design principles, the chatbot achieves modularity, reusability, and scalability. It exemplifies how RAG-based architectures enhance contextual understanding and user interaction in AI systems.

10. References

1. LangChain Documentation – `https://www.langchain.com`
2. FAISS Library – Facebook AI Similarity Search
3. Python Official Documentation – `https://docs.python.org`
4. Research Papers on RAG Architecture (Meta AI, 2020)