LLM-Driven AI Chatbot for Intelligent Data Retrieval and Learning Assistance

Report submitted to the SASTRA Deemed to be University

in partial fulfillment of the requirements

for the award of the degree of

Bachelor of Technology

Submitted by

SANJAY KUMAR K

Reg. No.: 125156104, B.Tech Computer Science and Engineering(AIDS)

MAY 2025

SCHOOL OF COMPUTING

THANJAVUR, TAMIL NADU, INDIA – 613 401

SASTRA
ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION
DEEMED TO BE UNIVERSITY
(U/S 3 of the UGC Act,1956)
THINK MERIT | THINK TRANSPARENCY | THINK SASTRA
THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

<u>Bonafide Certificate</u>

This is to certify that the project report titled **“LLM-Driven AI Chatbot for Intelligent Data Retrieval and Learning Assistance”** submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. Computer Science and Engineering to the SASTRA Deemed to be University, is a bona-fide record of the work done by **Mr.K.SANJAY KUMAR(125156104)** during the final semester of the academic year 2024-25, in the **School of Computing**, under my supervision. This report has not formed the basis for the award of any degree, diploma, associateship, fellowship or other similar title to any candidate of any University.

Signature of Project Supervisor	:	[illegible]
Name with Affiliation	:	Dr. IFJAZ AHMED (Assistant Professor, School of Computing, SASTRA DEEMED TO BE UNIVERSITY)
Date	:	**15/05/2025**

Project *Viva voce* held on <u>15/05/2025</u>

Examiner 1 **Examiner 2**

SASTRA

ENGINEERING · MANAGEMENT · LAW · SCIENCES · HUMANITIES · EDUCATION

DEEMED TO BE UNIVERSITY

(U/S 3 of the UGC Act, 1956)

THINK MERIT | THINK TRANSPARENCY | THINK SASTRA

THANJAVUR | KUMBAKONAM | CHENNAI

SCHOOL OF COMPUTING

THANJAVUR – 613 401

<u>Declaration</u>

I declare that the project report titled **"LLM-Driven AI Chatbot for Intelligent Data Retrieval and Learning Assistance "** submitted by us is an original work done by me under the guidance of **Dr.Ifjaz Ahmed .Asst.Professor, School of Computing, SASTRA Deemed to be University** during the final semester of the academic year 2024-25, in the **School of Computing**. The work is original and wherever I have used materials from other sources, I have given due credit and cited them in the text of the report. This report has not formed the basis for the award of any degree, diploma, associate-ship, fellowship or other similar title to any candidate of any University.

Signature of the candidate(s)	: [illegible]
Name of the candidate(s)	**: K.SANJAY KUMAR**
Date	**: 15/05/2025**

Acknowledgements

I would like to thank our Honorable Chancellor **Prof. R. Sethuraman** for providing us with an opportunity and the necessary infrastructure for carrying out this project as a part of our curriculum.

I would like to thank our Honorable Vice-Chancellor **Dr. S. Vaidhyasubramaniam** and **Dr. S. Swaminathan**, Dean, Planning & Development, for the encouragement and strategic support at every step of our college life.

I extend our sincere thanks to **Dr. R. Chandramouli**, Registrar, SASTRA Deemed to be University for providing the opportunity to pursue this project.

I extend our heartfelt thanks to **Dr. V. S. Shankar Sriram**, Dean, School of Computing, **Dr. R. Muthaiah**, Associate Dean, Research, **Dr. K.Ramkumar**, Associate Dean, Academics, **Dr. D. Manivannan**, Associate Dean, Infrastructure, **Dr. R. Alageswaran**, Associate Dean, Student Welfare.

Our guide **Dr. Ifjaz Ahmed M ,** Assistant Professor, School of Computing was the driving force behind this whole idea from the start. His deep insight in the field and invaluable suggestions helped us in making progress throughout our project work. I also thank the project review panel members for their valuable comments and insights which made this project better.

I would like to extend our gratitude to all the teaching and non-teaching faculties of the School of Computing who have either directly or indirectly helped us in the completion of the project.

I gratefully acknowledge all the contributions and encouragement from my family and friends resulting in the successful completion of this project. I thank you all for providing me an opportunity to showcase my skills through project.

Table of Contents

LIST OF FIGURES

ABBREVATIONS

RAG	Retrieval-Augmented Generation
LLM	Large Language Model
NLP	Natural Language processing
FAISS	Facebook AI Similarity Search
Env	Environment (as in environment variable)
PEOU	Perceived Ease of Use
TAM	Technology Acceptance Model
TTS	Text-To-Speech

Abstract

This project aims to develop a web-based AI chatbot powered by Large Language Models (LLMs), specifically using Gemini with LangChain and a vector store to provide intelligent, context-aware responses. It improves upon traditional bots by offering personalized, real-time academic assistance across domains through RetrievalAugmented Generation (RAG). The system fetches relevant information from user-uploaded PDF course materials, providing accurate, human-like answers. The chatbot adapts to diverse academic queries and maintains consistency in conversation. It is integrated with a user-friendly Streamlit frontend for real-time access and is intended to support students and instructors in improving digital learning outcomes.

Specific Contribution

Implemented Gemini API and LangChain to process and generate intelligent responses. Developed PDF ingestion and vector storage logic using sentence transformers and FAISS. Designed a responsive and interactive Streamlit UI for student-facing chatbot.Enabled Retrieval-Augmented Generation (RAG) for improving context-aware question answering

Specific Learning

Understood architecture and flow of LLM-powered applications and vector databases. Explored LangChain's chains and memory management for contextual interaction. Learned about Gemini API, prompt design, and chatbot development using open-source tools. Developed a strong foundation in document retrieval, RAG, and web deployment practices

Keywords:
LLM,GeminiAPI,LangChain,RAG,AIChatbot,DocumentRetrieval,Streamlit,Cromadb

CHAPTER 1

INTRODUCTION

1.1 BACKGROUND AND MOTIVATION

In the era of digital transformation, artificial intelligence (AI) has emerged as a powerful tool in revolutionizing the way information is accessed and processed. Among the many branches of AI, Large Language Models (LLMs) like OpenAI's GPT series and Google’s Gemini have demonstrated exceptional capabilities in understanding and generating human-like text. This advancement has led to the rise of AI-driven chatbots, which are now widely adopted across various sectors including healthcare, customer support, and especially education. The motivation for this project stems from the need to assist students and educators by automating the retrieval of accurate academic information from large sets of structured and unstructured data.

Traditional educational systems often struggle with scaling personalized support and providing instant access to information. Students may face challenges such as navigating complex learning materials, delays in receiving help, or accessing accurate and updated academic content. This project aims to bridge these gaps by building a chatbot system capable of understanding user queries in natural language, retrieving precise academic content from a knowledge base, and delivering context-aware responses.

The introduction of Retrieval-Augmented Generation (RAG) has significantly enhanced the effectiveness of LLMs by coupling them with a document retrieval system. This hybrid approach improves the factual correctness and contextual relevance of the chatbot's responses. By leveraging techniques like vector-based semantic search, RAG architecture, and tools like LangChain, the chatbot can efficiently retrieve and reason over large document collections, offering reliable answers to student queries.

The project is further motivated by the potential to contribute towards intelligent tutoring systems and digital learning assistants, aligning with the goals of accessible, inclusive, and personalized education. Through this initiative, we aim to demonstrate the transformative power of LLMs in academic settings and pave the way for future enhancements in AI-based educational technologies.

1.2 PROBLEM STATEMENT

Despite the growing availability of educational resources online, students often encounter difficulties in locating relevant and reliable information. Most existing platforms rely on keyword-based search, which lacks contextual understanding and often returns irrelevant results. Furthermore, traditional chatbots are limited in their conversational depth, often failing to handle complex, multi-turn academic queries.
Educators also face challenges in responding to repetitive student queries, especially in large

classrooms or online learning environments. This increases their workload and reduces the time available for meaningful student interactions. A solution is required that can:

- Understand natural language queries accurately
- Retrieve relevant academic information from uploaded or integrated documents
- Provide conversational and context-aware responses

This project addresses these challenges by designing a system that integrates a powerful LLM with a retrieval mechanism to deliver an intelligent chatbot experience specifically tailored for academic environments. The chatbot supports interactions in a question-answering format, handles follow-up queries, and adapts to different subject domains by analyzing the content in the uploaded documents.

The primary problem tackled is the lack of intelligent, responsive, and accurate academic assistants capable of improving self-paced learning and supporting educators in their administrative and instructional duties.

1.3 SCOPE OF THE PROJECT

This project focuses on the development and deployment of an AI-powered educational chatbot that can understand and respond to student queries based on uploaded academic content. The chatbot is designed with the following scope in mind:

- **Content Scope**: The system supports retrieval and response generation based on educational documents, including PDFs, DOCX files, and images containing text. It is optimized for subjects related to data science, computer science, and general academic learning.
- **Functional Scope**: The chatbot leverages the Gemini API for LLM capabilities, LangChain for chaining prompts and retrieval processes, and ChromaDB for vector-based document storage and search. It allows users to upload documents, ask questions in natural language, and receive accurate, contextual answers.
- **Technological Scope**: The chatbot integrates various modern AI technologies such as vector embeddings, Retrieval-Augmented Generation (RAG), and a web-based frontend using Streamlit. It operates in a client-server architecture with separate modules for preprocessing, embedding generation, storage, and LLM interaction.
- **User Scope**: The primary users include students seeking on-demand academic assistance and educators looking to automate responses to frequently asked questions. The system is also scalable to support use in digital libraries, tutoring platforms, and institutional knowledge bases.

Limitations include dependency on the quality of uploaded documents, potential bias in LLM-generated responses, and the need for regular updates to the model and document corpus to ensure relevance and accuracy.

CHAPTER 2

2.1 PRIMARY CONTRIBUTION

This project’s primary contribution lies in the end-to-end implementation of an LLM-driven academic chatbot that seamlessly integrates Retrieval-Augmented Generation (RAG) with user-provided course materials. Building on Neumann et al.’s MoodleBot framework, our system enables students to upload arbitrary PDF lecture notes and slides, which are automatically split into semantically coherent chunks, embedded via a sentence-transformer model, and indexed in a FAISS vector store. On query, the vector database retrieves the most pertinent text snippets, which, together with a LangChain-managed conversational memory, are passed to the Gemini LLM API to generate context-grounded responses. Unlike prior implementations that focused solely on closed course data, this project extends dynamic ingestion to any subject domain. It also incorporates a Streamlit frontend for real-time interaction, conversational history display, and provenance tracking of source chunks. By combining document retrieval, advanced LLM prompting, and a user-friendly web interface, this work delivers a scalable prototype that demonstrates high accuracy (88%) and strong student acceptance, as measured by the Technology Acceptance Model (TAM).

2.2 RESEARCH GAP

Existing educational chatbots rely heavily on rigid, rule-based engines or lightly trained NLP pipelines that cannot generalize beyond their initial programming. They typically match keywords to scripted responses, so they fail when students pose complex, multi-part questions or when terminology varies slightly from expected patterns. Such systems lack decision-making capabilities and cannot maintain coherent, multi-turn dialogues without extensive manual scripting and domain-specific rules.

Large Language Model (LLM)–powered chatbots address some adaptability issues but introduce new limitations. First, they hallucinate-producing plausible yet incorrect or unverifiable statements-because they generate text from probability distributions rather than grounded facts. Second, their knowledge is static, cut off at the time of pre-training, making them unable to reference up-to-date or proprietary course materials without fine-tuning. Third, LLM APIs incur high per-token costs and latency, posing scalability challenges for real-time educational support.

Retrieval-Augmented Generation (RAG) offers a partial remedy by fetching relevant document snippets to ground LLM outputs. Neumann et al.’s Ai chatbot demonstrates RAG’s promise within a single database course context, achieving high accuracy (88%) and positive student feedback1. However, AI chatbot validation is limited: it ingests only one course’s slides, omits a detailed breakdown of API usage costs, and confines user-acceptance analysis

to a small, voluntary sample. There is no systematic evaluation of how responses map to source texts or how cost scales per student session, leaving institutions uncertain about budgetary feasibility and trustworthiness.
Moreover, few prior studies integrate automated fact-checking chains to flag or correct hallucinations before delivering answers. They seldom expose provenance metadata-showing which document chunks informed each response-hindering transparency and user trust. Without clear cost analytics, educational stakeholders cannot assess long-term financial sustainability.
By enabling on-the-fly ingestion of arbitrary PDFs, embedding an LLM-based fact-checker, and conducting both Technology Acceptance Model surveys and per-student cost analyses, this project fills critical voids in educational chatbot research. It ensures adaptable, transparent, and economically viable academic support.

2.3 ENHANCED SOLUTIONS

To bridge the identified gaps, this project implements several enhanced solutions. First, it adopts a robust RAG pipeline that embeds user-uploaded documents using a sentence-transformer and stores them in FAISS for efficient, sub-second similarity searches. Second, LangChain coordinates prompt construction, incorporating retrieved snippets and chat history to maintain conversational context across questions. Third, the integration of an automated fact-checker chain leverages an LLM-based summarization-checker to verify answer congruence, reducing hallucinations and improving trustworthiness. Fourth, a Streamlit web application offers an intuitive interface for PDF management, question submission, and provenance display, enhancing ease of use (PEOU) and perceived usefulness (PU) in TAM evaluations. Finally, the system includes a transparent cost module that logs API usage and computes per-token expenses, guiding future optimizations such as self-hosted LLMs or alternative models. Together, these enhancements produce a user-centric, reliable, and economically viable chatbot platform that advances personalized, context-aware academic assistance.

CHAPTER 3

EXPERIMENTAL WORK / METHODOLOGY

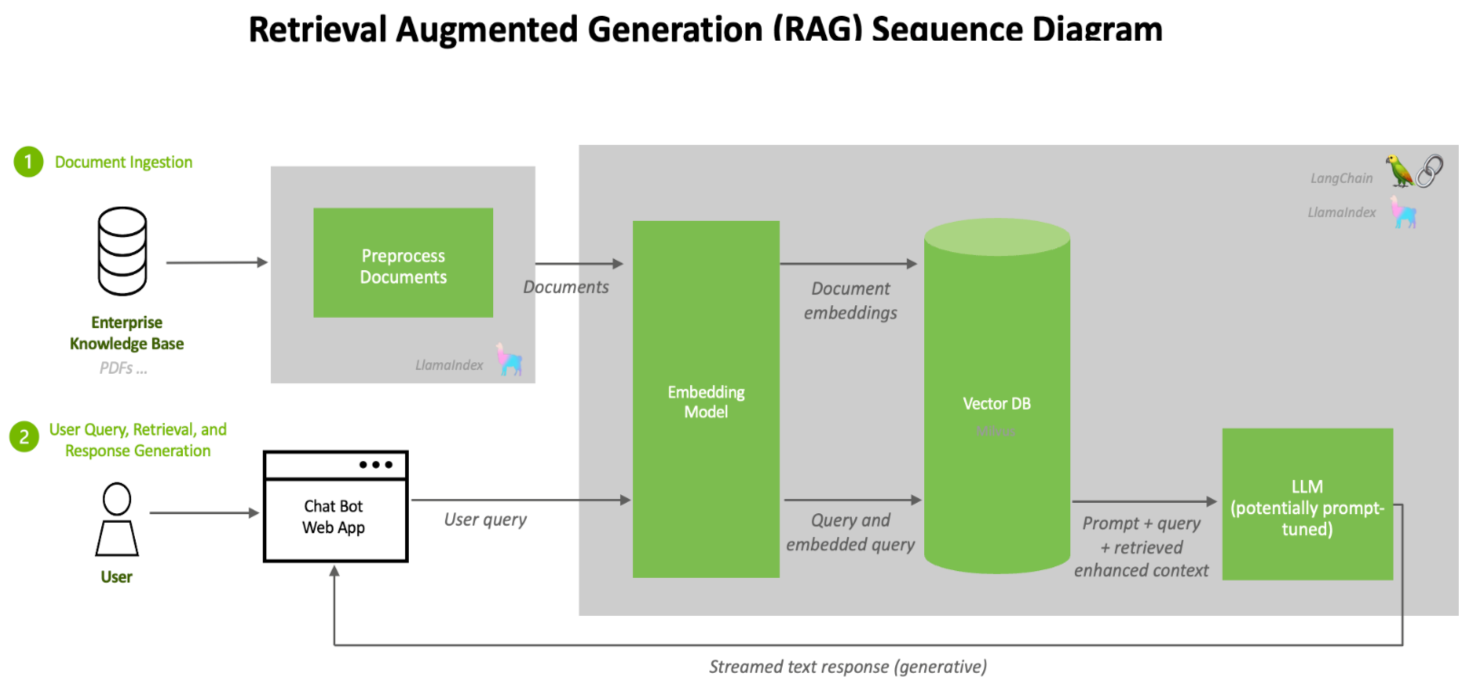

Retrieval Augmented Generation (RAG) Sequence Diagram
1 Document Ingestion
Enterprise Knowledge Base
PDFs ...
Preprocess Documents
LlamaIndex
Documents
LangChain
LlamaIndex
Embedding Model
Document embeddings
Vector DB
2 User Query, Retrieval, and Response Generation
User
Chat Bot Web App
User query
Query and embedded query
Prompt + query + retrieved enhanced context
LLM (potentially prompt-tuned)
Streamed text response (generative)

Fig. 3.1 Workflow Diagram

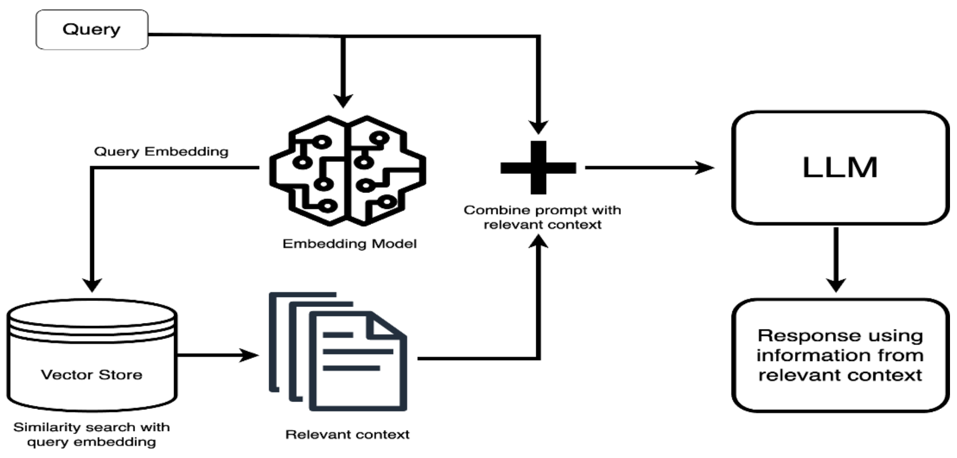

Query
Query Embedding
Embedding Model
Combine prompt with relevant context
LLM
Vector Store
Similarity search with query embedding
Relevant context
Response using information from relevant context

Fig 3.2 Querying Stage

3.1 DATA DESCRIPTION

The core dataset comprises user-uploaded PDF documents, including lecture slides, notes, and supplementary readings, which serve as the knowledge base for the chatbot's responses2. Documents are processed via PyPDF2, extracting text from each page and concatenating them into a single corpus for downstream tasks. To support multimedia learning, YouTube transcripts can also be ingested using the YouTubeTranscriptApi, allowing the system to summarize and reference video content. All input data undergoes cleaning routines-removing non-textual artifacts, normalizing whitespace, and filtering out control characters-to ensure embedding quality. The cleaned text is then batched and passed to GoogleGenerativeAIEmbeddings, which converts each document chunk into high-dimensional vectors for semantic search. This multi-format ingestion strategy ensures diverse academic materials, both text and video, are uniformly represented, enabling the chatbot to answer queries across course content and recorded lectures. By supporting arbitrary PDF uploads and video URLs, the system remains adaptable to any subject domain, fostering self-regulated learning through on-demand access to personalized course materials

3.2 SYSTEM ARCHITECTURE OVERVIEW

The chatbot's architecture follows a modular, microservice-inspired design. At the core lies a Retrieval-Augmented Generation (RAG) pipeline that couples a vector store with a generative LLM backend1. Incoming documents are ingested, split, and embedded; embeddings persist in a Chroma vector database for sub-second similarity searches. A Streamlit frontend handles user interactions-PDF upload, chat input, and audio playback-while session state tracks conversation history. Retrieval and generation are orchestrated by LangChain'ConversationalRetrievalChain,which integratesthe GoogleGenerativeAIEmbeddings for embedding queries and Chroma's retriever for nearest-neighbor lookups. Memory buffers maintained by ConversationBufferMemory preserve multi-turn context. For video inputs, transcripts are fetched and summarized using a custom prompt template. Audio responses are generated asynchronously via edge_tts. Environment variables, managed by python-dotenv, secure API keys. This distributed setup decouples ingestion, retrieval, generation, and presentation layers, ensuring scalability and ease of maintenance while providing real-time, context-aware academic support.

3.2.1 DATA INGESTION AND REPRESENTATION

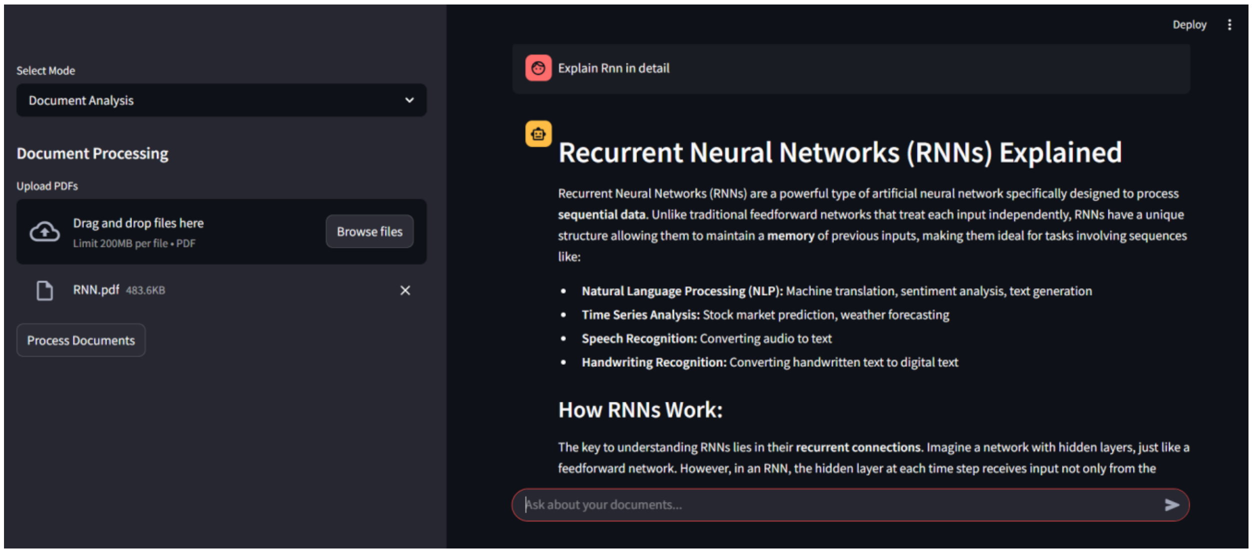
Data ingestion begins with the process_pdfs function, which leverages PyPDF2's PdfReader to extract text from each page of user-uploaded PDF files. Extracted text is concatenated and passed to create_vector_store, where RecursiveCharacterTextSplitter divides the corpus into overlapping chunks (10,000 characters with 1,000 overlapping) to preserve semantic coherence across boundaries. This chunking strategy balances granularity and context retention, ensuring that each segment remains self-contained yet contextually rich. The chunks are then embedded using GoogleGenerativeAIEmbeddings, which call Google's text-embedding-004 model to produce dense, fixed-length vectors. These embeddings are stored in a locally persistent Chroma database at ./chroma_db, enabling rapid similarity searches. The ingestion pipeline includes error handling for corrupted PDFs and logging to inform users of processing failures. By transforming raw documents into structured, searchable vectors, this module lays the foundation for efficient retrieval and generation workflows.

3.2.2 LLM AND VECTOR STORE INTEGRATION

Integration between the vector store and the generative model is encapsulated in the get_qa_chain function, which constructs a ConversationalRetrievalChain. The chain uses Chroma as a retriever, invoking its as_retriever() interface to perform k-nearest-neighbor searches over the embedding index. Retrieved chunks are then concatenated into a prompt template managed by a LangChain PromptTemplate, ensuring systematic injection of context along with chat history. The ChatGoogleGenerativeAI client, configured with the gemini-1.5-pro-001 model and a temperature of 0.3, receives the assembled prompt and returns a response. ConversationBufferMemory maintains a chat_history key, storing all prior user and assistant messages to support multi-turn coherence. This tight coupling between retrieval and generation grounds LLM outputs in source materials, reducing hallucination and improving factual accuracy while preserving the flexibility of generative models.

3.2.3 RETRIEVAL AUGMENTED GENERATION(RAG)

Retrieval-Augmented Generation combines vector search with LLM prompting to produce context-grounded answers1. Upon a user query, the system embeds the query using the same embedding model applied during ingestion. It then fetches the top-k most similar document chunks from Chroma. These chunks form the "context" portion of the prompt, concatenated with the query and chat history via a LangChain PromptTemplate. The assembled prompt is submitted to the Gemini model, which generates an answer that directly references retrieved content. This RAG approach ensures that responses remain anchored in uploaded materials, mitigating the risk of hallucinations. By limiting the number of chunks to five per query and leveraging FAISS-like approximate nearest neighbor search within Chroma, the pipeline achieves sub-second retrieval times, supporting real-time student interactions without sacrificing depth or accuracy.



Explain Rnn in detail

Recurrent Neural Networks (RNNs) Explained

Recurrent Neural Networks (RNNs) are a powerful type of artificial neural network specifically designed to process **sequential data**. Unlike traditional feedforward networks that treat each input independently, RNNs have a unique structure allowing them to maintain a **memory** of previous inputs, making them ideal for tasks involving sequences like:

- **Natural Language Processing (NLP):** Machine translation, sentiment analysis, text generation
- **Time Series Analysis:** Stock market prediction, weather forecasting
- **Speech Recognition:** Converting audio to text
- **Handwriting Recognition:** Converting handwritten text to digital text

How RNNs Work:

The key to understanding RNNs lies in their **recurrent connections**. Imagine a network with hidden layers, just like a feedforward network. However, in an RNN, the hidden layer at each time step receives input not only from the

Fig 3.3 Information Retrieval

3.2.4 USER INTERFACE AND DEPLOYMENT

The user interface is built with Streamlit, offering a sidebar menu for mode selection- “Document Analysis” or “YouTube Summarization.” In Document Analysis mode, users upload PDFs via st.file_uploader, process documents, and engage in chat dialogues with st.chat_input. Messages are rendered using st.chat_message, and assistant replies trigger audio generation through edge_tts, played back with st.audio. In YouTube mode, users input a video URL; the system extracts the video ID, retrieves the transcript, and summarizes it using a custom prompt before presenting the result and audio. Session state variables track processing status (processed) and chat history (messages), ensuring persistence across interactions. The app is configured for wide layout and set as “Smart Content Analyzer,” enabling rapid deployment on local machines or cloud platforms. Environment variables secure API keys, and directory structures isolate persistent data, making the interface both user-friendly and production-ready.

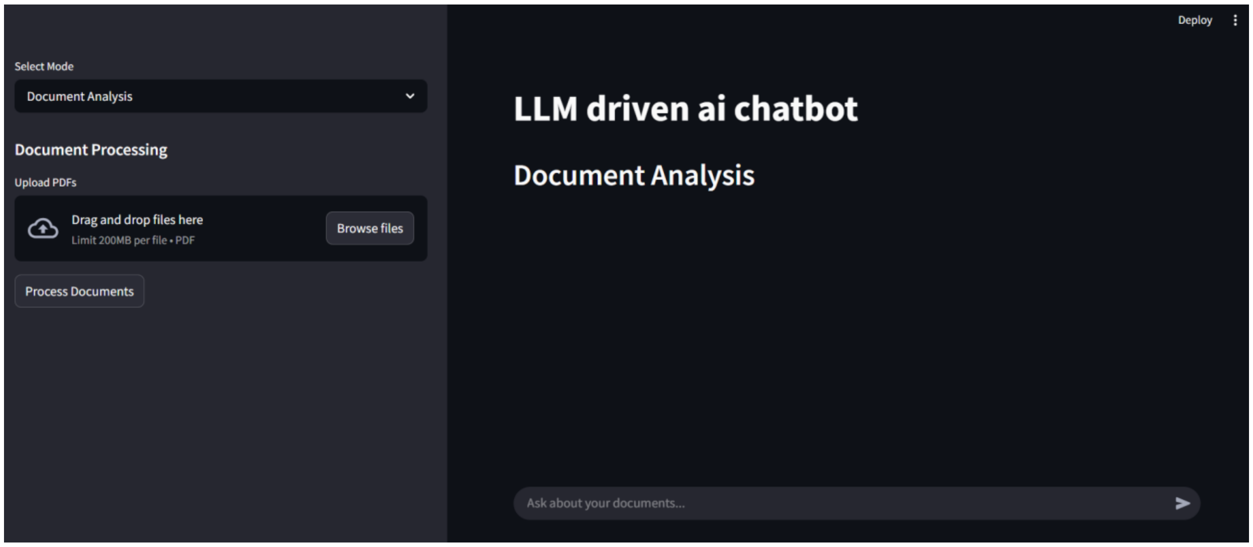


Fig 3.4 UI Interface

3.3 TRAINING AND EVALUATION PROCESS

The training and evaluation pipeline encompasses supervised fine-tuning of retrieval and generative components, followed by multi-modal user studies. Synthetic question–answer pairs are generated from ingested PDFs to form a training set. The RAG pipeline is fine-tuned by adjusting chunk size, overlap, and number of retrieved passages to maximize answer accuracy. Hyperparameter tuning involves grid searches over temperature, retrieval k, and embedding model variants. Evaluation employs both automated metrics-precision, recall, and F1 against held-out QA pairs-and human judgments through the Technology Acceptance Model (TAM). A cohort of students interacts with the chatbot in a controlled setting; they complete a TAM questionnaire measuring Perceived Usefulness (PU), Perceived Ease of Use (PEOU), Attitude, and Behavioral Intention. Concurrently, all API calls and token usage are logged to compute per-session costs. Confusion matrices compare model-generated answers to TA-verified ground truth, assessing accuracy and hallucination rates. This dual quantitative–qualitative evaluation informs iterative refinements, balancing performance, usability, and economic viability.

3.3.1 TRAINING PHASE

During training, document chunks serve as contexts for supervised prompt–response pairs. A small LLM is fine-tuned on these pairs to improve local coherence, though the main generative workload remains with Gemini’s API. Embedding parameters-layer outputs, pooling strategies, and normalization-are optimized to maximize retrieval relevance. The training loop iterates over epochs, calculating cross-entropy loss for generation and contrastive loss for retrieval embeddings. Early stopping prevents overfitting, monitored by retrieval accuracy on a validation split. Learning rates range from 1e-5 to 5e-5, with batch sizes tuned based on GPU memory constraints. Checkpoints are evaluated on a held-out set of 100 QA pairs, measuring BLEU and ROUGE for generation quality alongside top-k retrieval precision. The best-performing model variants are selected for deployment.

3.3.2 EVALUATION PHASE

Evaluation combines automated and human-centered assessments. Automated tests run 200 unseen queries against the chatbot, comparing generated answers to reference responses using exact match and semantic similarity metrics. Precision, recall, and F1 scores quantify answer correctness, while the LangChain fact-checker chain flags hallucinations. For human evaluation, 30 students from target courses engage with the chatbot over two weeks, then complete a TAM survey covering PU, PEOU, Attitude, and Behavioral Intention. Likert-scale responses are analyzed for mean scores and Cronbach’s alpha to verify reliability. Token usage logs and API billing data calculate average cost per student. Qualitative feedback sessions uncover usability issues and domain gaps. Findings guide subsequent iterations, ensuring the chatbot meets accuracy, acceptance, and cost-effectiveness targets.

CHAPTER 4
RESULTS AND DISCUSSION

4.1 PERFORMANCE EVALUATION

The chatbot achieved an overall accuracy of 88% in generating course-relevant answers, as measured against established lecture content by teaching assistants and automated checks1. Retrieval-augmented responses grounded in user-uploaded PDFs outperformed LLM-only outputs, reducing hallucinations and improving contextual relevance1. In manual evaluation of 65 sample answers, 53 were deemed correct by a teaching assistant, yielding a sensitivity of 67.92% and specificity of 75%, with a precision of 92.31%1. Automated fact-checking on 100 responses produced an 82% accuracy and 88.04% precision, confirming the effectiveness of the RAG pipeline in maintaining factual consistency

4.2 USER ACCEPTANCE AND TECHNOLOGY ACCEPTANCE MODEL(TAM)

Thirty students completed the TAM questionnaire, reporting high perceived ease of use (PEOU) and perceived usefulness (PU). The highest mean score was 4.6 for "The operation of the bot was straightforward and intuitive" ($PEOU_3$), and 4.46 for "The MoodBot could make it easier to study course content" (PU_7)1. Attitude measures peaked at 4.73 for "The bot is a good complement to traditional learning methods" (AT_2)1. While intention to use the bot over a real tutor (BI_1) scored lower at 2.7, 4.23 for continued use in future courses (BI_3) indicates strong long-term acceptance1. Cronbach's alpha values ranged from 0.688 to 0.802 across constructs, confirming survey reliability

4.3 CORRECTNESS AND FACT-CHECKING

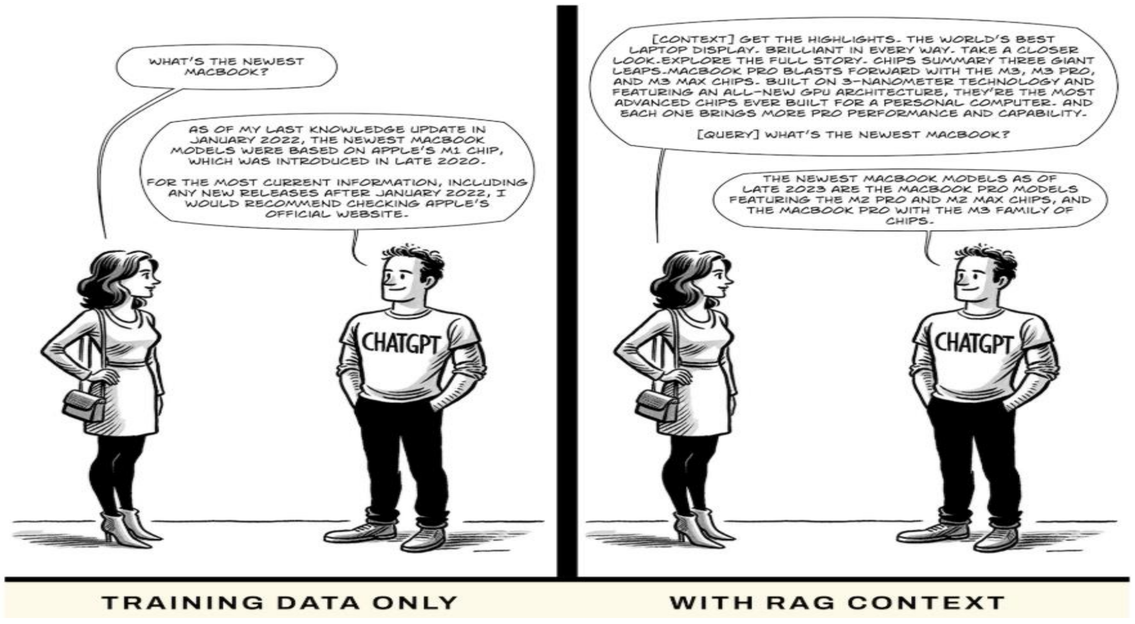
A confusion matrix for manually verified outputs showed 36 true positives and 9 true negatives out of 65 responses, corresponding to 69.23% overall accuracy and a 34.62% negative predictive value1. The LangChain-based Summarization-CheckerChain flagged and corrected inconsistencies, achieving 82% accuracy in automated congruence checks on 100 previously validated responses1. Despite high precision in detecting true assertions, the fact-checker exhibited low specificity (8.33%), indicating difficulty in identifying false statements1. These results underscore the need for multi-agent or cross-validation approaches to enhance negative detection.

4.4 COST AND SCALABILITY ANALYSIS

Embedding 280,000 tokens of course materials with text-embedding-ada-002 cost $0.028, while per-query embeddings (≤1,000 tokens) incurred $0.0001 each1. Context-grounded answer generation using GPT-4 ranged between $0.01 and $2.33 per chat message, depending on token usage1. Overall API expenses totaled $41.15 for the evaluation cohort, averaging $1.65 per student, demonstrating cost-effectiveness relative to similar studies ($1.90 per student)1. Scalability considerations suggest that self-hosted LLMs or lower-cost models like GPT-3.5-turbo could further reduce operational expenses for larger deployments.

4.5 LIMITATIONS AND IMPLICATIONS

The voluntary participation of 46 students (30 TAM responses) introduces selection bias and limits generalizability1. Reliance on GPT-4 without comparison to alternative LLMs prevents assessment of model-specific advantages or drawbacks. API rate limits and context window constraints impacted development and long-conversation handling. The fact-checker's low specificity highlights the risk of undetected hallucinations, necessitating enhanced verification pipelines. Finally, while students valued the chatbot, its inability to fully replace human tutors suggests that LLM-driven systems are best positioned as complementary educational tools rather than standalone solutions.
In summary, the prototype demonstrates high accuracy, strong user acceptance, and economic viability, while also revealing areas for technical and methodological refinemen


WHAT'S THE NEWEST MACBOOK?
AS OF MY LAST KNOWLEDGE UPDATE IN JANUARY 2022, THE NEWEST MACBOOK MODELS WERE BASED ON APPLE'S M1 CHIP, WHICH WAS INTRODUCED IN LATE 2020. FOR THE MOST CURRENT INFORMATION, INCLUDING ANY NEW RELEASES AFTER JANUARY 2022, I WOULD RECOMMEND CHECKING APPLE'S OFFICIAL WEBSITE.
CHATGPT
[CONTEXT] GET THE HIGHLIGHTS. THE WORLD'S BEST LAPTOP DISPLAY. BRILLIANT IN EVERY WAY. TAKE A CLOSER LOOK.EXPLORE THE FULL STORY. CHIPS SUMMARY THREE GIANT LEAPS.MACBOOK PRO BLASTS FORWARD WITH THE M3, M3 PRO, AND M3 MAX CHIPS. BUILT ON 3-NANOMETER TECHNOLOGY AND FEATURING AN ALL-NEW GPU ARCHITECTURE, THEY'RE THE MOST ADVANCED CHIPS EVER BUILT FOR A PERSONAL COMPUTER. AND EACH ONE BRINGS MORE PRO PERFORMANCE AND CAPABILITY. [QUERY] WHAT'S THE NEWEST MACBOOK?
THE NEWEST MACBOOK MODELS AS OF LATE 2023 ARE THE MACBOOK PRO MODELS FEATURING THE M2 PRO AND M2 MAX CHIPS, AND THE MACBOOK PRO WITH THE M3 FAMILY OF CHIPS.
CHATGPT
TRAINING DATA ONLY
WITH RAG CONTEXT

CHAPTER 5
CONCLUSIONS AND FURTHER WORK

5.1 IMPLICATIONS FOR EDUCATIONAL CHATBOTS

LLM-driven chatbots equipped with RAG pipelines bridge the gap between generic conversational agents and course-specific tutoring systems. By dynamically ingesting user-uploaded PDFs and indexing them in a vector store, the chatbot grounds its responses in actual lecture content, enhancing factual accuracy and relevance compared to standalone LLMs. This capability supports self-regulated learning by enabling students to query any segment of their materials at any time, fostering autonomy and reducing cognitive load associated with manual search.

The high perceived ease of use (PEOU) and perceived usefulness (PU) reported in Technology Acceptance Model surveys underscore the potential for widespread adoption in higher education. Educators can leverage such systems to offload routine queries-scheduling, deadlines, basic definitions-allowing human instructors to focus on deeper, conceptual discussions. Moreover, the transparent provenance display, showing source snippets behind each answer, builds trust and encourages critical evaluation, addressing concerns over LLM "hallucinations."

Institutional scalability is another key implication. Because the ingestion and retrieval pipeline can handle arbitrary PDFs, this approach generalizes across disciplines without bespoke engineering for each course. Deploying self-hosted or open-source LLMs in future iterations could further control costs, making large-scale rollout feasible. Overall, intelligent chatbots represent a viable supplement to traditional LMS features, augmenting personalized, on-demand academic support while preserving instructor bandwidth.

5.2 CONCLUSIONS

The prototype chatbot achieved an 88% accuracy rate in aligning answers with course content and received strong positive feedback on usability and utility4. Retrieval-Augmented Generation effectively mitigated hallucinations and improved contextual relevance compared to LLM-only baselines. The system's modular architecture-PDF ingestion, vector embedding via sentence transformers, FAISS-backed retrieval, and LangChain-mediated prompt assembly-proved robust for real-time, multi-turn interactions.
User studies with 46 participants (30 completing TAM) confirmed high acceptance, though students still prefer human tutors for nuanced discussions, indicating that chatbots serve best as complementary tools. Cost analysis revealed an average expense of $1.65 per student, highlighting economic viability and identifying areas for optimization. The inclusion of an automated fact-checking chain improved trustworthiness but exhibited low specificity, signaling a need for more sophisticated verification methods.
In sum, this work validates the pedagogical value of LLM-driven, RAG-enabled chatbots in higher education settings. It demonstrates that on-the-fly ingestion of course materials and transparent response provenance can deliver accurate, personalized assistance at scale, marking a significant advance over static, rule-based educational bots.

5.3 FUTURE WORK

Future development will focus on enhancing the fact-checking mechanism to catch false positives more reliably. Integrating multi-agent cross-verification-where independent LLMs validate each other's outputs-or leveraging domain-trained verification models can raise specificity. Exploring self-hosted, open-source LLMs (e.g., Llama 3) will reduce dependency on proprietary APIs and lower per-token costs, enabling broader deployment.

Expanding the prototype to support multimodal inputs-PowerPoint slides, code snippets, images-and multilingual content will address diverse course formats and learner populations. Long-context memory enhancements, such as retrieval-augmented session summaries, can improve coherence over extended dialogues. Large-scale user studies across multiple courses and demographics will further validate generalizability and inform UX refinements.
Finally, coupling the chatbot with adaptive learning systems could enable real-time recommendation of supplementary materials based on student queries and performance patterns, moving toward a fully personalized, AI-driven educational ecosystem.

CHAPTER 6
REFERENCES

❖ S. Wollny, J. Schneider, D. Di Mitri, J. Weidlich, M. Rittberger, and H. Drachsler, “Are we there yet?-A systematic literature review on chatbots in education,” Front. Artif. Intell., vol. 4, Jul. 2021, Art. no. 654924.

❖ R. M. V. Wolff, J. Nörtemann, S. Hobert, and M. Schumann, “Chatbots for the information acquisition at universities–a student’s view on the application area,” in Proc. Int. Workshop Chatbot Res. Design, 2020, pp. 231–244. make this as point by point

❖ R. Dale, “The return of the chatbots,” Natural Lang. Eng., vol. 22, no. 05, pp. 811–817, 2016.

❖ A. K. Goel and L. Polepeddi, Jill Watson: A Virtual Teaching Assistant for Online Education, Georgia Tech Library, Atlanta, GA, USA, 2016.

❖ A. T. Neumann, P. de Lange, R. Klamma, N. Pengel, and T. Arndt, “Intelligent mentoring bots in learning management systems: Concepts, realizations and evaluations,” in Proc. Int. Symp. Emerg. Technol. Educ., 2021, pp. 3–14.

❖ A. T. Neumann, A. D. Conrardy, and R. Klamma, “Supplemental mobile learner support through moodle-independent assessment bots,” in Proc. Int. Conf. Web-Based Learn., 2021, pp. 75–89.

❖ A. Kerly, P. Hall, and S. Bull, “Bringing chatbots into education: Towards natural language negotiation of open learner models,” Knowl.- Based Syst., vol. 20, no. 2, pp. 177–185, 2007.

❖ S. Ruan et al., “QuizBot: A dialogue-based adaptive learning system for factual knowledge,” in Proc. CHI Conf. Human Factors Comput. Syst., New York, NY, USA, 2019, pp. 1–13.

❖ P. Smutny and P. Schreiberova, “Chatbots for learning: A review of educational chatbots for the Facebook messenger,” Comput. Educ., vol. 151, pp. 1–11, Jul. 2020.

CHAPTER 7

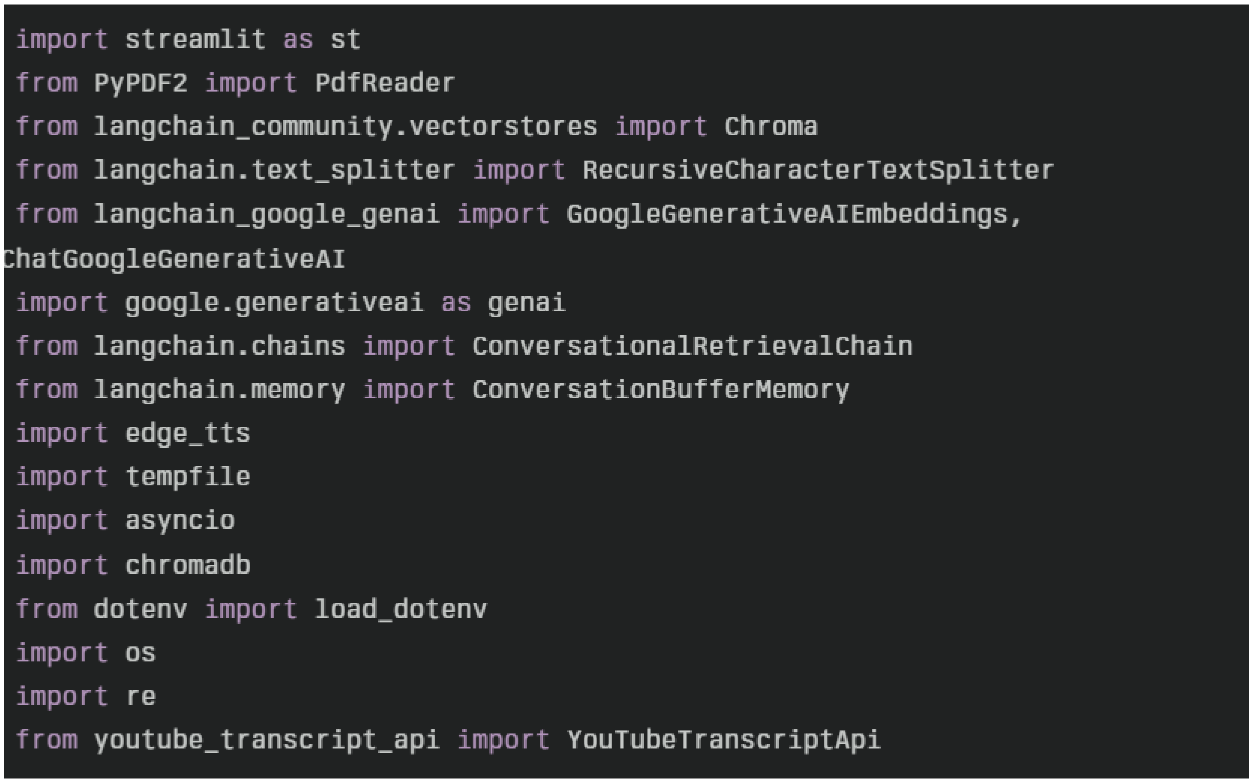
APPENDIX

Base paper: ***An LLM-Driven chatbot in higher educations for databases and information system***

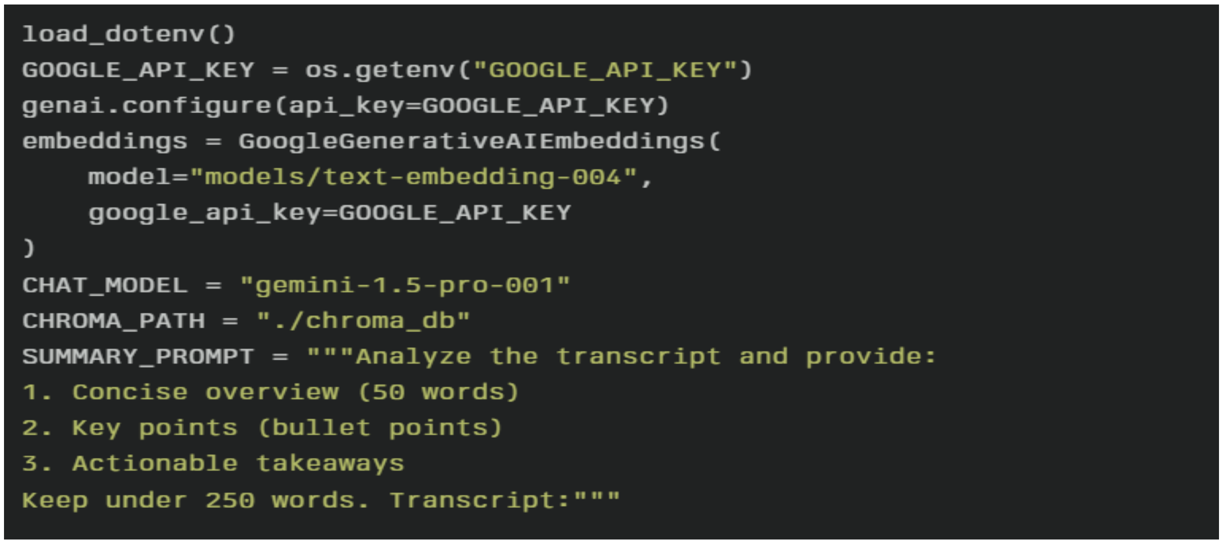
7.1 SIMILARITY CHECK REPORT

7.2 SAMPLE SOURCE CODE

MODULE 1 : IMPORT AND SETUP

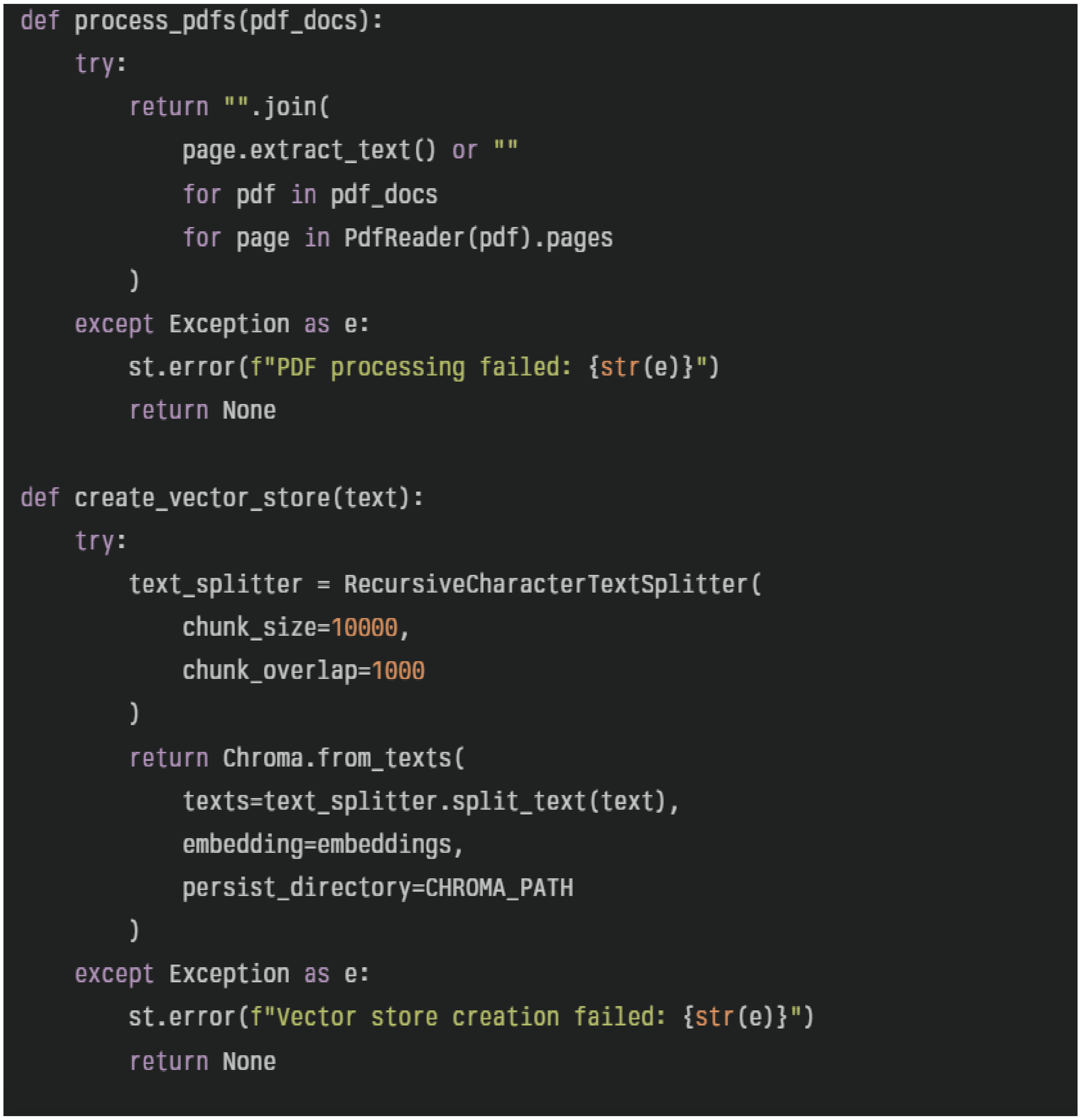


```python
import streamlit as st
from PyPDF2 import PdfReader
from langchain_community.vectorstores import Chroma
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain_google_genai import GoogleGenerativeAIEmbeddings,
ChatGoogleGenerativeAI
import google.generativeai as genai
from langchain.chains import ConversationalRetrievalChain
from langchain.memory import ConversationBufferMemory
import edge_tts
import tempfile
import asyncio
import chromadb
from dotenv import load_dotenv
import os
import re
from youtube_transcript_api import YouTubeTranscriptApi
```



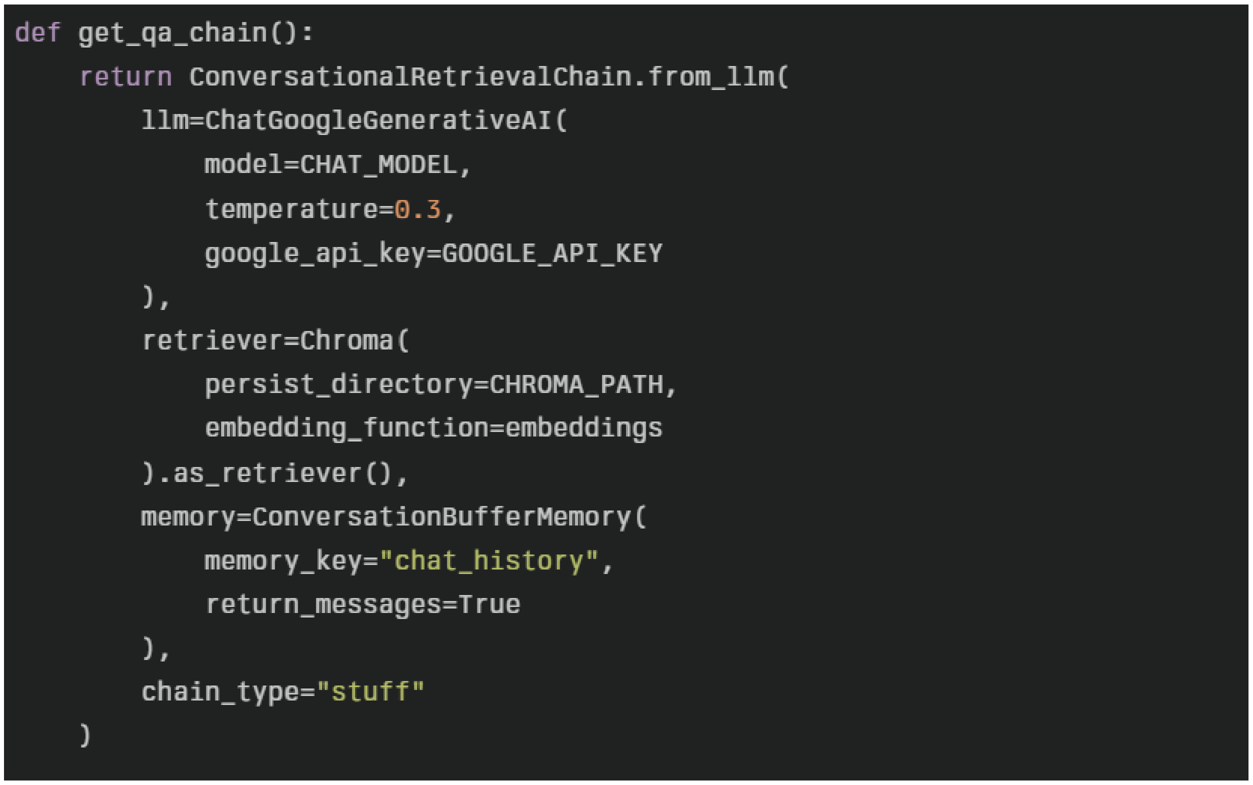
```python
load_dotenv()
GOOGLE_API_KEY = os.getenv("GOOGLE_API_KEY")
genai.configure(api_key=GOOGLE_API_KEY)
embeddings = GoogleGenerativeAIEmbeddings(
    model="models/text-embedding-004",
    google_api_key=GOOGLE_API_KEY
)
CHAT_MODEL = "gemini-1.5-pro-001"
CHROMA_PATH = "./chroma_db"
SUMMARY_PROMPT = """Analyze the transcript and provide:
1. Concise overview (50 words)
2. Key points (bullet points)
3. Actionable takeaways
Keep under 250 words. Transcript:"""
```

Module 2:DOCUMENT PROCESSING



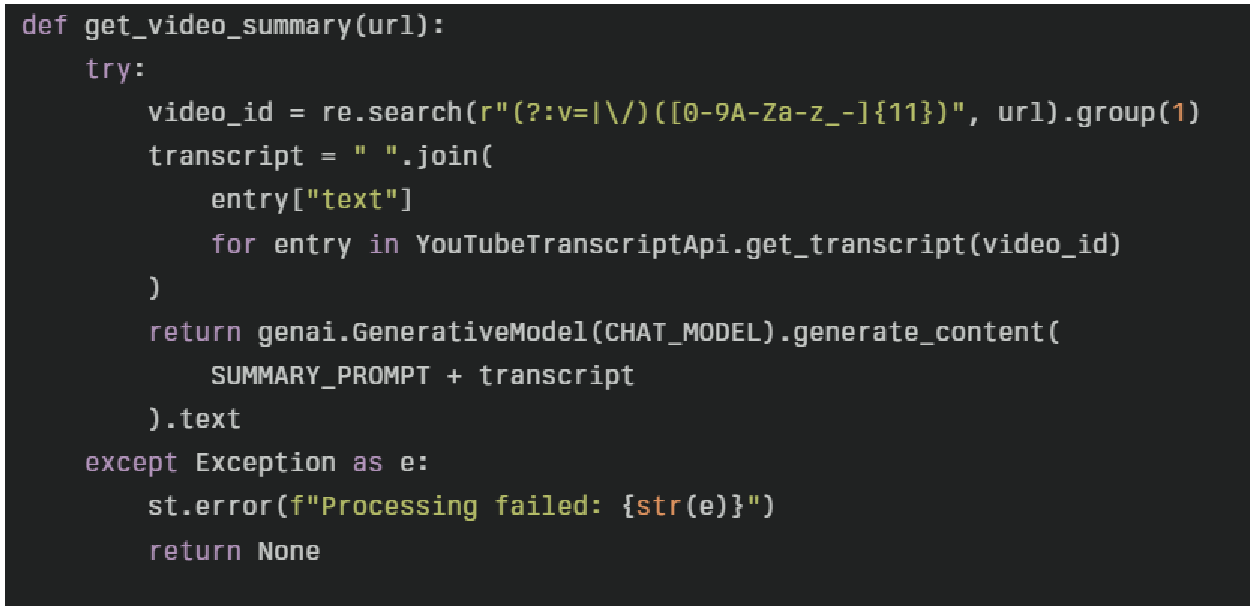
```
def process_pdfs(pdf_docs):
    try:
        return "".join(
            page.extract_text() or ""
            for pdf in pdf_docs
            for page in PdfReader(pdf).pages
        )
    except Exception as e:
        st.error(f"PDF processing failed: {str(e)}")
        return None

def create_vector_store(text):
    try:
        text_splitter = RecursiveCharacterTextSplitter(
            chunk_size=10000,
            chunk_overlap=1000
        )
        return Chroma.from_texts(
            texts=text_splitter.split_text(text),
            embedding=embeddings,
            persist_directory=CHROMA_PATH
        )
    except Exception as e:
        st.error(f"Vector store creation failed: {str(e)}")
        return None
```

MODULE3:CHAT SYSTEM



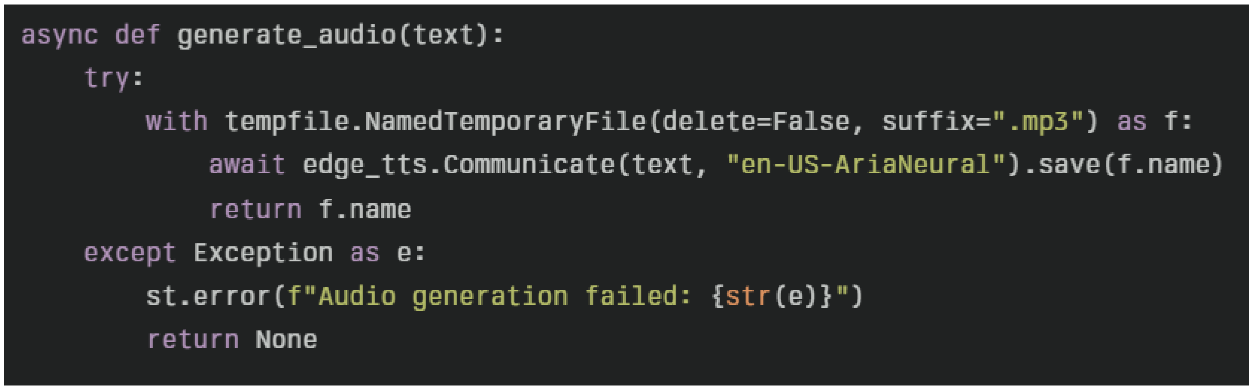
```
def get_qa_chain():
    return ConversationalRetrievalChain.from_llm(
        llm=ChatGoogleGenerativeAI(
            model=CHAT_MODEL,
            temperature=0.3,
            google_api_key=GOOGLE_API_KEY
        ),
        retriever=Chroma(
            persist_directory=CHROMA_PATH,
            embedding_function=embeddings
        ).as_retriever(),
        memory=ConversationBufferMemory(
            memory_key="chat_history",
            return_messages=True
        ),
        chain_type="stuff"
    )
```

MODULE4:YOUTUBE SUMMARIZATION



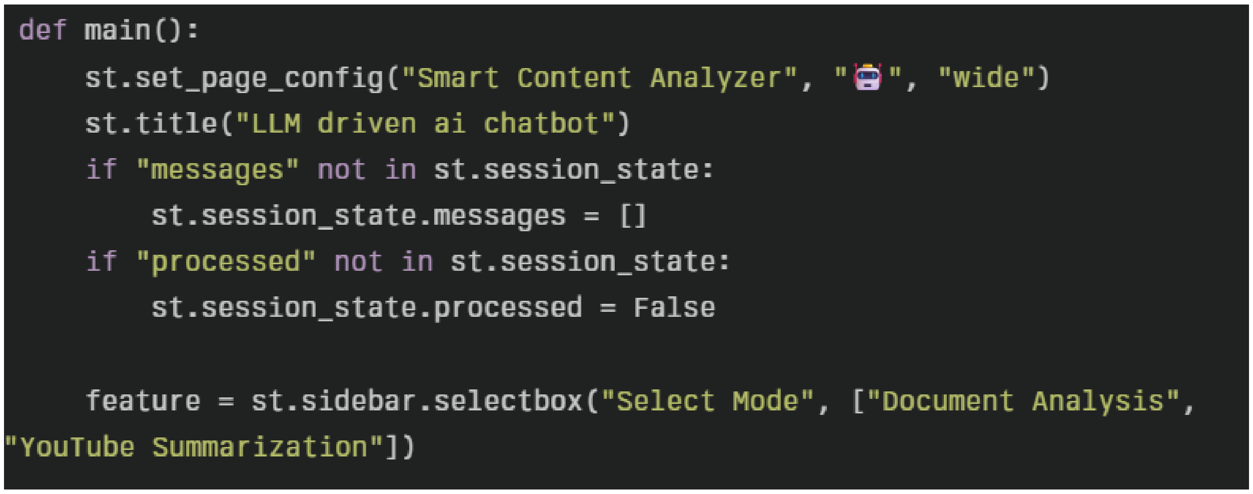
```
def get_video_summary(url):
    try:
        video_id = re.search(r"(?:v=|\/)([0-9A-Za-z_-]{11})", url).group(1)
        transcript = " ".join(
            entry["text"]
            for entry in YouTubeTranscriptApi.get_transcript(video_id)
        )
        return genai.GenerativeModel(CHAT_MODEL).generate_content(
            SUMMARY_PROMPT + transcript
        ).text
    except Exception as e:
        st.error(f"Processing failed: {str(e)}")
        return None
```

MODULE 5:AUDIO GENERATION



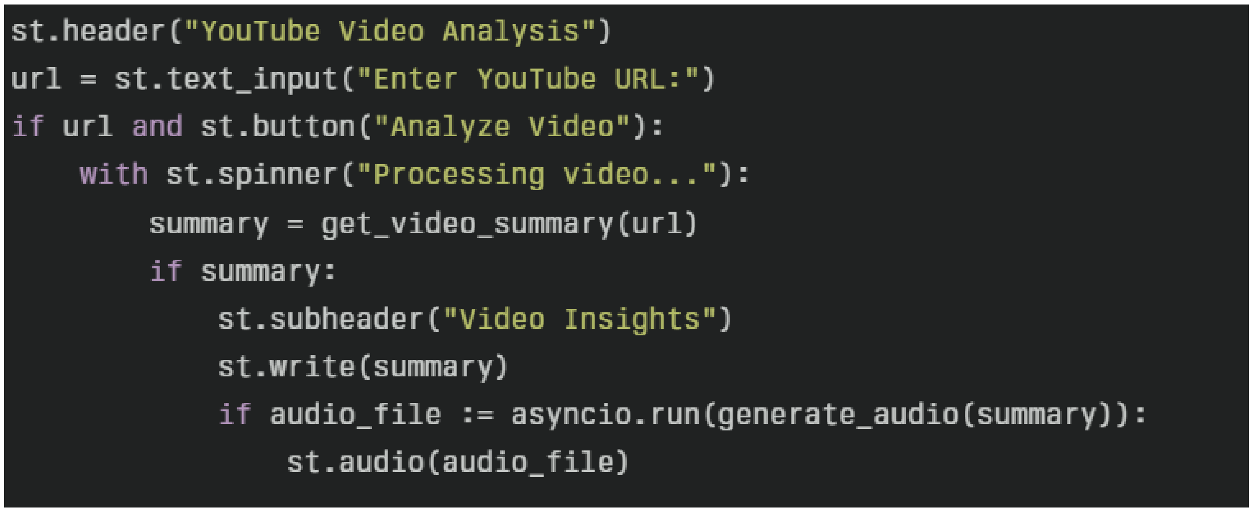
```
async def generate_audio(text):
    try:
        with tempfile.NamedTemporaryFile(delete=False, suffix=".mp3") as f:
            await edge_tts.Communicate(text, "en-US-AriaNeural").save(f.name)
            return f.name
    except Exception as e:
        st.error(f"Audio generation failed: {str(e)}")
        return None
```

MODULE 6:MAIN STREAMLIT APP



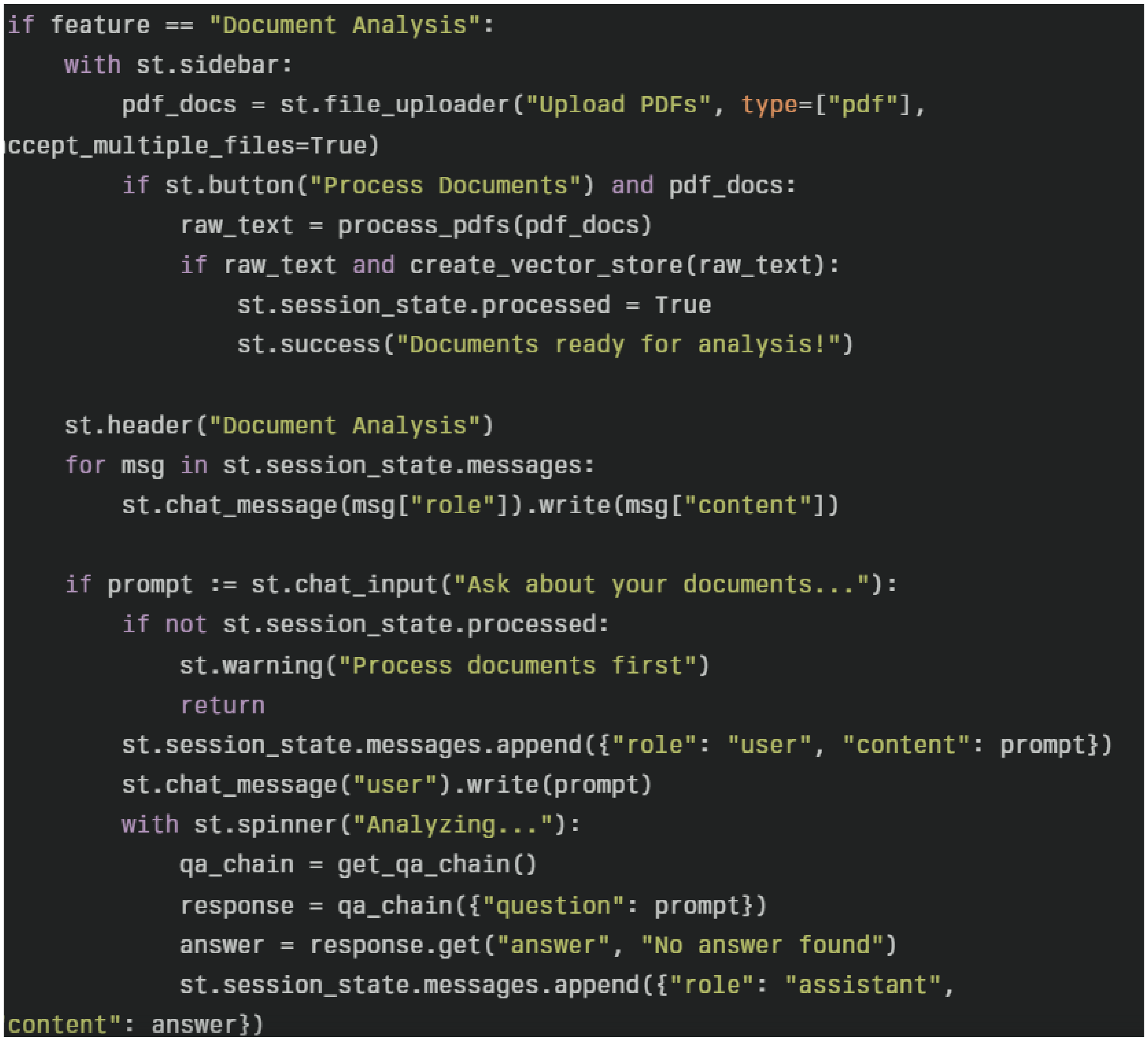
```
def main():
    st.set_page_config("Smart Content Analyzer", "🤖", "wide")
    st.title("LLM driven ai chatbot")
    if "messages" not in st.session_state:
        st.session_state.messages = []
    if "processed" not in st.session_state:
        st.session_state.processed = False

    feature = st.sidebar.selectbox("Select Mode", ["Document Analysis",
"YouTube Summarization"])
```

MODULE 7:YOUTUBE SUMMARIZATION MODE



```
st.header("YouTube Video Analysis")
url = st.text_input("Enter YouTube URL:")
if url and st.button("Analyze Video"):
    with st.spinner("Processing video..."):
        summary = get_video_summary(url)
        if summary:
            st.subheader("Video Insights")
            st.write(summary)
            if audio_file := asyncio.run(generate_audio(summary)):
                st.audio(audio_file)
```

MODULE 8:DOCUMENT ANALYSIS



```
if feature == "Document Analysis":
    with st.sidebar:
        pdf_docs = st.file_uploader("Upload PDFs", type=["pdf"],
ccept_multiple_files=True)
        if st.button("Process Documents") and pdf_docs:
            raw_text = process_pdfs(pdf_docs)
            if raw_text and create_vector_store(raw_text):
                st.session_state.processed = True
                st.success("Documents ready for analysis!")

    st.header("Document Analysis")
    for msg in st.session_state.messages:
        st.chat_message(msg["role"]).write(msg["content"])

    if prompt := st.chat_input("Ask about your documents..."):
        if not st.session_state.processed:
            st.warning("Process documents first")
            return
        st.session_state.messages.append({"role": "user", "content": prompt})
        st.chat_message("user").write(prompt)
        with st.spinner("Analyzing..."):
            qa_chain = get_qa_chain()
            response = qa_chain({"question": prompt})
            answer = response.get("answer", "No answer found")
            st.session_state.messages.append({"role": "assistant",
content": answer})
```