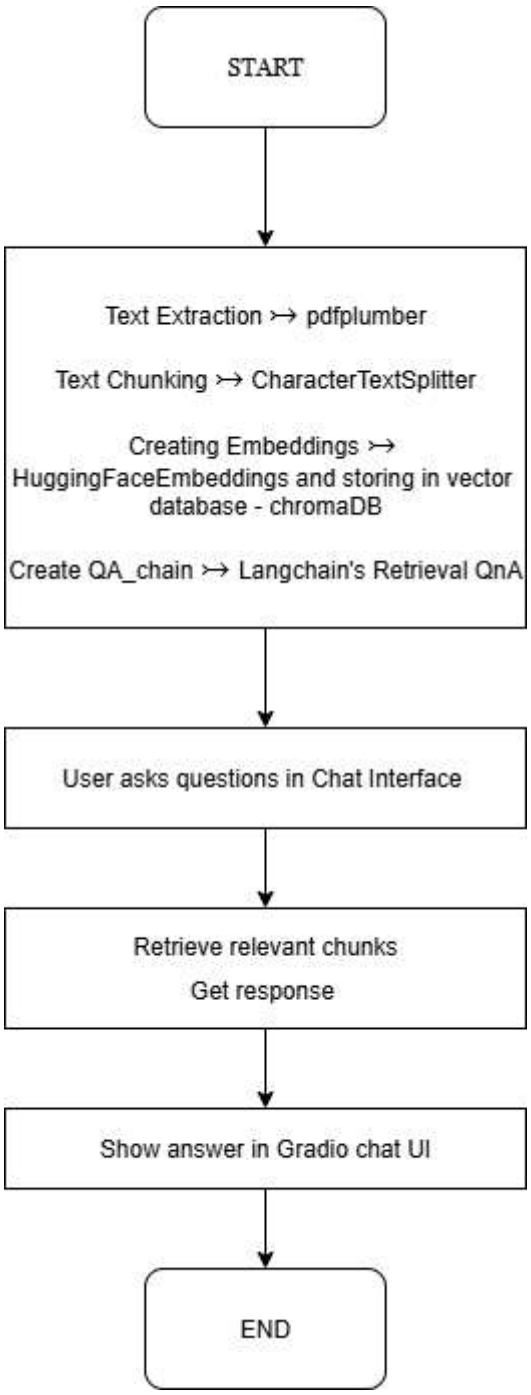
PDF-BASED CHATBOT

Problem Statement

PDFs are widely used in academic settings and professional workplaces as research papers, reports, manual, and legal papers. Manually extracting relevant information from these documents to find specific information can be time-consuming and inefficient. This project uses large language models (LLMs) and retrieval systems to automate that process. It allows users to upload any PDF and interact with it through a chatbot. The chatbot provides instant, accurate answers using AI tools like Google Gemini and LangChain.

Flowchart



Methodology

Our project uses the following skillset:

1. Programming Language: Python is used for building, training and deploying the model
2. Libraries Used:
   ↣ Gradio: The trained model is integrated into a Gradio Interface
   ↣ pdfplumer: For extracting text from PDF files
   ↣ Langchain: For creating the retrieval-based QA chain
   ↣ chromaDB: Vector database used to store and retrieve text embeddings
3. Large Language Model(LLM) : Google Generative AI (**gemini-2.5-pro**)
4. Embeddings: HuggingFace Embedding - To convert text chunks into numerical vectors

Steps:

1. PDF Upload and Text Extraction:
   User uploads a PDF file via a Gradio interface. The text is extracted using pdfplumber.

2. Text Preprocessing:
   The text is split into manageable chunks using LangChain's CharacterTextSplitter.

3. Embeddings & Vector Store:
   Each chunk is embedded using HuggingFaceEmbeddings. Chunks are stored in Chroma, a vector database. This allows for efficient semantic similarity searches when a user asks a question.

4. Question Answering Chain:
   When a user inputs a question, the chatbot retrieves the most relevant text chunks from ChromaDB and sends them to the Gemini model(gemini-2.5-pro). LangChain's RetrievalQA chain is used to handle this process.

Algorithm

PDF Chatbot Algorithm:

1. Receive PDF file from user.
2. Use pdfplumber to extract text from each page.
3. Split the extracted text into overlapping chunks.
4. Convert chunks into vector embeddings using HuggingFace.
5. Store the vectorized chunks in ChromaDB.
6. Set up a RetrievalQA chain with Gemini LLM and Chroma retriever.
7. When a user asks a question:
   a. Retrieve top relevant chunks from Chroma.
   b. Provide retrieved chunks to Gemini for context-aware response.
   c. Return answer to the user in the chat interface.

Code

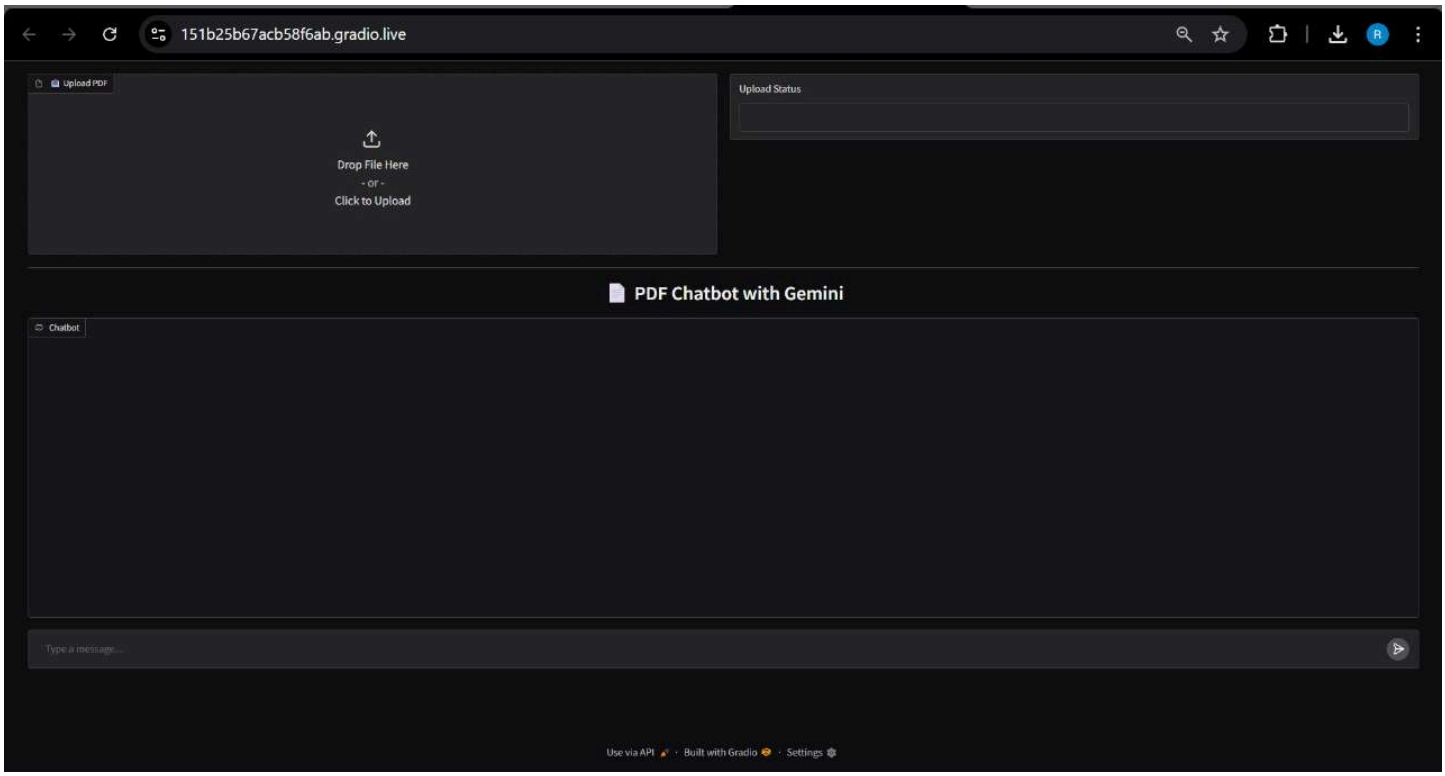
```
!pip install langchain chromadb gradio google-generativeai pdfplumber transformers
langchain-google-genai langchain-community
import os
os.environ['GOOGLE_API_KEY'] = ''

from langchain_google_genai import ChatGoogleGenerativeAI
llm = ChatGoogleGenerativeAI(model="gemini-2.5-pro", temperature=0.2)

import pdfplumber
import tempfile

#Function to extract content from pdf using pdfplumber
def read_pdf(file_obj):
    try:
        with pdfplumber.open(file_obj) as pdf:
            text = ""
            for page in pdf.pages:
```

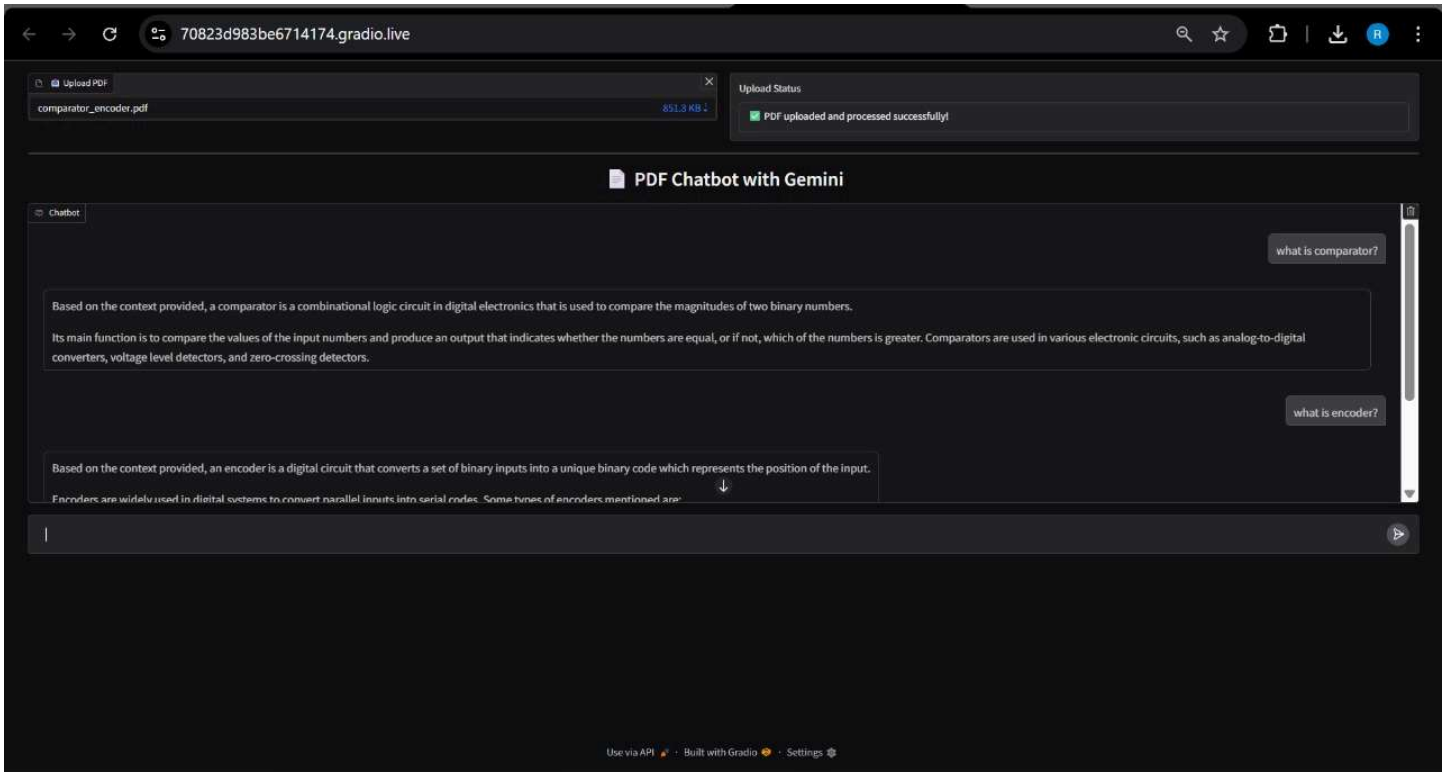
```
                page_text = page.extract_text()
                if page_text:
                    text += page_text + "\n"
        return text
    except Exception as e:
        return f"PDF Read Error: {e}"

from langchain.text_splitter import CharacterTextSplitter

#Splitting extracted text into overlapping chunks
def chunk_text(text, chunk_size=1000, chunk_overlap=200):
    splitter = CharacterTextSplitter(separator="\n", chunk_size=chunk_size,
chunk_overlap=chunk_overlap)
    return splitter.split_text(text)

from langchain.embeddings import HuggingFaceEmbeddings

embedding = HuggingFaceEmbeddings()

from langchain.vectorstores import Chroma
def create_vector_store(chunks):
    return Chroma.from_texts(texts=chunks, embedding=embedding)

from langchain.chains import RetrievalQA

#Setting up a RetrievalQA_Chain
def create_qa_chain(vectorstore):
    retriever = vectorstore.as_retriever()
    qa_chain = RetrievalQA.from_chain_type(
        llm=llm,
        chain_type="stuff",
        retriever=retriever,
        return_source_documents=True     )
    return qa_chain

#Creating Interface
import gradio as gr

qa_chain = None

def process_pdf(file):
    global qa_chain
    try:
        if not file:
            return "❌ No file uploaded."

        text = read_pdf(file)
        if text.startswith("PDF Read Error"):
```

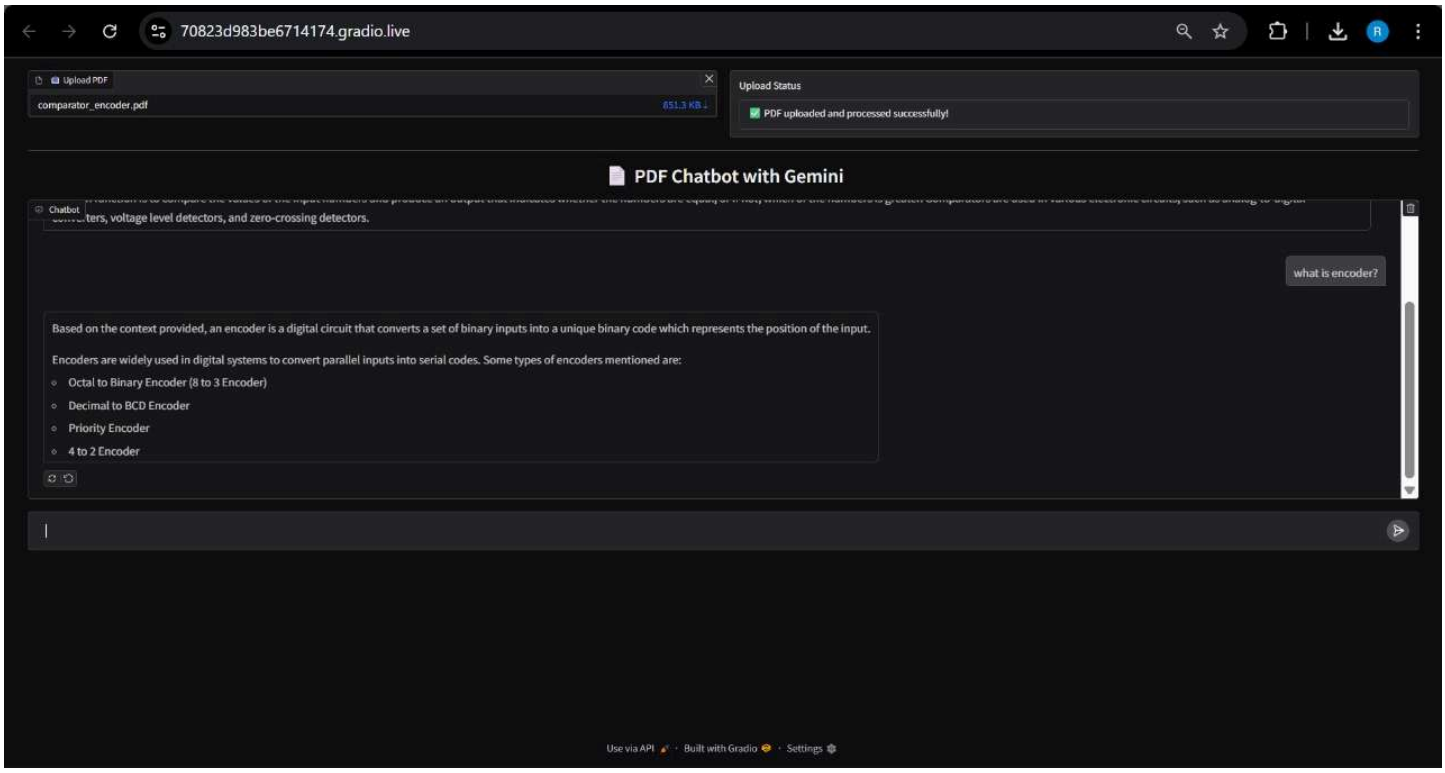
```
            return text

        chunks = chunk_text(text)
        if not chunks:
            return "❌ No text content found in PDF."

        vectorstore = create_vector_store(chunks)
        qa_chain = create_qa_chain(vectorstore)
        return "✅ PDF uploaded and processed successfully!"
    except Exception as e:
        return f"❌ Error during processing: {str(e)}"


def answer_question(question, history):
    if not qa_chain:
        history.append({"role": "assistant", "content": "❌ Please upload a PDF
first."})
        return history

    try:
        print(f"🔍 User asked: {question}")

        docs = qa_chain.retriever.get_relevant_documents(question)
        print(f"✅ Retrieved {len(docs)} relevant chunks.")
        for i, doc in enumerate(docs[:2]):
            print(f"Chunk 1:\n{doc.page_content[:300]}...\n")

        response = qa_chain.invoke({"query": question})
        answer = response.get("result", "No answer found.")

        history.append({"role": "user", "content": question})
        history.append({"role": "assistant", "content": answer})

        return history

    except Exception as e:
        print("❌ Exception during question answering:", e)
        history.append({"role": "assistant", "content": f"❌ Error: {e}"})
        return history


upload = gr.File(label="Upload PDF")

chatbot = gr.ChatInterface(fn=answer_question, type='messages', title="📄 PDF Chatbot
with Gemini", fill_height=True)

app = gr.Blocks()
with app:
```

```
    with gr.Row():
        upload_button = gr.File(label="📤 Upload PDF", file_types=['.pdf'])
        output = gr.Textbox(label="Upload Status")
    upload_button.change(fn=process_pdf, inputs=upload_button, outputs=output)
    chatbot.render()

app.launch(share=True)
```

Output







Conclusion

This project presents an efficient and user-friendly solution for interacting with PDF documents. By combining text extraction, chunking, vector-based search, and a responsive interface, it enables users to ask questions and receive relevant answers from their uploaded PDFs.

The system simplifies the process of finding information in large documents, making it especially useful for students, educators, and professionals. Overall, it enhances productivity by reducing the time spent manually searching through PDFs and offers a practical tool for document-based learning and reference.