PROJECT REPORT

ON

”**AskPDF: A Smart Q&A Bot for Document Search**”

Submitted to

Sant Gadge Baba Amravati University, Amravati.

In partial fulfillment of the requirements of

M.Sc. (Computer Software) Final Year Examination

Submitted by

Jagruti R. Zade

Under the guidance of

Mr. S. S. Gawande

Assistant Professor

(Department of Computer Science)

Shri Shivaji Education Society Amravati’s

SHRI SHIVAJI SCIENCE COLLEGE

Amravati.

2024-2025

CERTIFICATE

This is to certify that the Project Report entitled **AskPDF: A Smart Q&A Bot for Document Search** being submitted by **Jagruti R. Zade** in partial fulfillment for the award of Master of Science in Computer Software (Final Year) **Sant Gadge Baba Amravati University, Amravati** is a record of work carried out for the session 2024-25.

To the best of my knowledge, the matter presented in this project has not been presented earlier for a similar degree/diploma.

Place : Amravati **Project Guide**
Date : **Mr. S. S. Gawande**

Internal Examiner **External Examiner**

Head
Dept. of Computer Science

DECLARATION

To,
The Principal,
Shri Shivaji Science College,
Amravati.

Respected Sir,

I the undersigned, hereby declare that the Project work entitled **AskPDF: A Smart Q&A Bot for Document Search** submitted to **Sant Gadge Baba Amravati University, Amravati** is my independent work. This is my original work and has not been submitted anywhere for any degree/diploma. The system presented herein has not been duplicated from any other source.

I understand that any such copying is liable to be punished in any way the University authority may deem fit.

Thanking You.

Place : Amravati
Date :

Yours Sincerely,

Jagruti R. Zade
M.Sc. IInd (Sem-IV)

ACKNOWLEDGMENT

We wish to express our sincere thanks to the many persons who helped us to develop our original project work.

First, we express our sincere thanks to Principal **Dr. G. V. Korpe**, Shri Shivaji Science College, Amravati for providing the infrastructure and facilities without which it would have been impossible to complete this hard task.

Foremost this is to**, Dr. M. M. Bhonde**, Head of the Department of Computer Science who has aided us in completing this project report. and I am thankful to **Mr. S. S. Gawande,** Assistant Professor, Department of Computer Science, Shri Shivaji Science College, Amravati for their constant inspiration and guidance throughout this project work.

Our foremost thanks are to Other Staff, who have guided us in completing this project report, We take the opportunity to express our deep sense of gratitude and wholehearted thanks for his inspiration and guidance throughout this project.

I express my gratitude to all members of the teaching and non-teaching staff of the Department of Computer Science for their cooperation during the course.
Finally, we thank our friends and especially those who helped us in our endeavors.

Place: Amravati
Date:

INTRODUCTION

ABSTRACT

In the era of information overload, extracting relevant insights from large documents, especially PDFs like research papers, manuals, or reports, can be time-consuming and inefficient. This project aims to solve that problem by developing a local AI-powered chatbot that allows users to upload a PDF file and ask questions about its content in a conversational manner. The system processes the uploaded PDF, extracts its content, and intelligently answers questions by identifying the most relevant parts of the document.

The chatbot is built using Flask for the backend web framework, PyMuPDF (fitz) for extracting text from PDF files, and Sentence Transformers for semantic understanding and similarity-based question answering. Once a PDF is uploaded, its content is broken down into meaningful sentence-level chunks. Each chunk is then converted into a mathematical representation, called an embedding, using the pre-trained "multi-qa-mpnet-base-dot-v1" SentenceTransformer model. These embeddings are stored in memory and later compared to the embedding of the user's question using cosine similarity.

When a user submits a question through the chatbot interface, the system computes its embedding and finds the most relevant document chunk(s) by measuring similarity scores with the previously generated embeddings. The most semantically relevant chunk is returned as the answer. This approach allows the chatbot to answer questions in a context-aware manner, even if the exact words of the question are not present in the document.

The chatbot runs locally, meaning that no external APIs are called, ensuring privacy, speed, and cost-effectiveness. It features a simple and interactive web interface, where users can upload PDFs, view the file name, and ask multiple questions, receiving real-time answers. The chatbot is particularly useful in academic, research, and business scenarios where quick document summarization and Q&A can enhance productivity.

This thesis documents the design, development, and implementation of the AI PDF chatbot, with an emphasis on natural language understanding, information retrieval, and user-friendly web deployment. It also explores the strengths, limitations, and future opportunities for improvement, such as adding multi-document support, streaming responses, or more advanced summarization capabilities.

INTRODUCTION

In today's digital age, vast amounts of information are stored in electronic documents, especially in PDF format. Whether it's academic research papers, technical manuals, corporate reports, or legal documents, accessing relevant information quickly from a large PDF can be a daunting task. Traditionally, users need to manually read through pages of content to find specific answers, which is both time-consuming and inefficient.

To address this challenge, this project introduces a smart AI chatbot system capable of reading and understanding PDF documents, allowing users to ask questions and get meaningful answers directly from the document content. The chatbot mimics the behavior of a human assistant who has read the entire document and can answer questions related to it.

The project is built using Flask, a lightweight web framework in Python, and incorporates natural language processing (NLP) techniques through Sentence Transformers to understand the meaning behind user queries and compare them with the content of the uploaded document. The system uses PyMuPDF (fitz) to extract raw text from PDFs and processes the content by breaking it down into small, meaningful chunks. These chunks are then converted into embeddings – mathematical vectors that capture the semantic meaning of the text.

When a user uploads a PDF, the system prepares the document by chunking and embedding its contents. Later, when a user submits a question, the chatbot generates a similar embedding for the question and compares it with the document chunks using cosine similarity to identify the most relevant answer. The response is then presented back to the user through an interactive web interface.

This project highlights how Natural Language Understanding (NLU), combined with effective backend architecture and machine learning models, can be used to build a helpful and intuitive document assistant. It is designed to be local-first, meaning the entire process runs on the user's system without depending on external APIs or internet connectivity. This ensures faster performance and higher privacy, especially useful in domains where document confidentiality is critical.

The following thesis explores the complete design, implementation, and working of this intelligent chatbot system. It also discusses the technologies used, the benefits of semantic search over keyword-based methods, and potential improvements that can make the system even more powerful in the future.

OBJECTIVES

The objectives of the Chatbot for helping college students are as follows:

The primary objective of this project is to develop a web-based **AI PDF Chatbot** that allows users to interact with the contents of a PDF document through natural language queries and receive relevant answers in real-time. The system aims to offer a user-friendly, intelligent alternative to manual document search by leveraging modern NLP and machine learning techniques.

Main Objective:

- To design and implement a chatbot system that can accurately **retrieve contextually relevant answers** from uploaded PDF documents based on user-input questions in natural language.
- **To enable PDF document upload and text extraction:**
  - Use PyPDF2 to parse and extract text from PDF files.
  - Support handling of multiple pages and varying document structures.
- **To preprocess and segment extracted text:**
  - Break down the extracted text into smaller, meaningful chunks for better indexing and retrieval.
  - Ensure uniform chunk sizes to maintain embedding quality.
- **To convert text into semantic embeddings:**
  - Use the pre-trained SentenceTransformer model (*multi-qa-mpnet-base-dot-v1*) to generate high-quality vector representations of both questions and document chunks.
- **To create a fast and efficient similarity search system:**
  - Utilize **FAISS** to build a vector index for the document chunks.
  - Implement fast nearest-neighbor search to retrieve the most relevant answers based on semantic similarity.
- **To provide a real-time, chatbot-style user interface:**
  - Build a web interface using HTML, CSS, and JavaScript that allows:
    - PDF upload
    - Input of questions
    - Display of bot-like answers in a chat format
- **To ensure seamless integration between frontend and backend:**
  - Design RESTful API endpoints (/upload, /ask) using Flask to handle data exchange between the client and server.
- **To enhance user experience with streaming and responsiveness:**
  - Stream the bot's answer word-by-word to simulate natural conversation.
  - Provide real-time feedback during file upload and question processing.
- **To test and validate the chatbot's performance:**
  - Evaluate the system's ability to retrieve correct and relevant answers.
  - Handle various types of questions, such as factual, contextual, and summary-based queries.

SCOPE OF STUDY

The scope of Chatbot for helping college students encompasses the following key functionalities:
The scope of this study focuses on the development and implementation of an intelligent chatbot system capable of answering questions from PDF documents using natural language processing (NLP) and semantic search techniques. This system is specifically designed to improve the way users interact with large volumes of textual information by enabling intelligent, context-aware document querying.

- **In-Scope Features:**
  1. **PDF Document Processing:**
     - The system supports uploading and processing of PDF files.
     - Text is extracted from the uploaded document using PyPDF2 and split into smaller chunks to improve manageability and relevance during search.
  2. **Semantic Search Capability:**
     - By using a state-of-the-art Sentence Transformer model (multi-qa-mpnet-base-dot-v1), the system goes beyond keyword matching and instead performs semantic understanding of both user queries and document content.
     - FAISS is used for building an efficient similarity index to retrieve the top relevant results.
  3. **Real-Time Chatbot Interface:**
     - The chatbot allows users to ask free-form questions through a simple and interactive web interface.
     - Responses are generated in real-time, simulating a human-like conversation experience.
  4. **Web-Based Deployment:**
     - The entire system is accessible via a browser, thanks to Flask (backend) and standard frontend technologies (HTML, CSS, JavaScript).
     - Communication between frontend and backend is handled via HTTP requests.
  5. **Streaming Response Output:**
     - Answers are streamed word-by-word to create a more natural conversational flow and enhance user engagement.

Out-of-Scope Elements:

1. **Multilingual Support:**
   - This version of the system only supports English-language documents and questions. Other languages are not currently supported.
2. **Complex Document Structures:**
   - PDFs containing complex layouts like scanned images, tables, and handwritten text may not be processed effectively, as the system relies on basic text extraction.
3. **Multiple Document Handling:**
   - At present, the chatbot works with a single uploaded document at a time. It does not support querying across multiple documents or maintaining document history.
4. **Authentication and User Management:**
   - The project does not include user login, role-based access, or data persistence beyond the active session.

SIGNIFICANCE OF STUDY

The **AI PDF Chatbot** project holds significant relevance in today's digital world where information is predominantly available in document formats such as PDFs. Reading and extracting relevant content from large documents is often time-consuming and inefficient. This study offers a practical solution by developing a system that allows users to interact with such documents using natural language, thereby enhancing accessibility and productivity.

🔹 **1. Reducing Information Overload**

In academic, corporate, and legal environments, users are often faced with voluminous documents. This chatbot streamlines access to relevant content without requiring the user to read the entire document. By retrieving specific answers based on queries, it minimizes information overload and supports faster decision-making.

🔹 **2. Enhancing User Experience with AI**

The integration of **Natural Language Processing (NLP)** and **semantic search** provides a human-like interaction. This significantly improves the user experience compared to traditional keyword-based searches, allowing users to ask questions as they would in a conversation.

🔹 **3. Real-World Applications**

This study has broad applications in education, research, customer support, and knowledge management. For instance:

- **Students** can ask questions about course material.
- **Researchers** can quickly extract key insights from academic papers.
- **Professionals** can review business contracts or legal documents more efficiently.

🔹 **4. Demonstrating the Power of Modern NLP Models**

The use of pre-trained transformer models like multi-qa-mpnet-base-dot-v1 demonstrates how state-of-the-art NLP tools can be practically applied to solve real-world problems. This contributes to the growing field of applied machine learning and helps bridge the gap between theoretical research and practical implementation.

🔹 **5. Promoting Intelligent Document Interaction**

This project promotes the idea of **"interactive documents"** — where users do not passively read, but actively engage with the content through intelligent systems. This paradigm shift could lead to the development of smarter knowledge systems and more personalized learning tools.

🔹 **6. Encouraging Innovation in Low-Code AI Solutions**

With a simple architecture combining **Flask**, **JavaScript**, **FAISS**, and **Sentence Transformers**, this project shows how powerful AI systems can be built without complex infrastructure. It serves as a learning model for students and developers entering the AI and NLP domains.

REQUIREMENT AND ANALYSIS

PURPOSE

The primary purpose of this study is to design and develop an intelligent, AI-powered chatbot system capable of reading and understanding PDF documents to provide relevant answers to user queries in real-time. This project aims to bridge the gap between human curiosity and machine-readable document content by utilizing modern natural language processing (NLP) techniques.

- **Key Purposes of the Study:**
  1. **To Simplify Document Understanding**
     The chatbot enables users to interact with large and complex PDF documents in a conversational way, eliminating the need to manually search for specific information.
  2. **To Integrate NLP and Semantic Search in Real-world Use Cases**
     This project demonstrates the practical application of sentence embedding models and similarity search using FAISS to retrieve the most contextually accurate answers from document content.
  3. **To Enhance User Interaction with Static Data**
     Static documents like PDFs often require sequential reading. The chatbot transforms them into dynamic, queryable content, improving user efficiency and engagement.
  4. **To Create a Web-based Tool with Educational and Professional Value**
     By deploying the chatbot via a Flask web application with an intuitive frontend, the project serves as an accessible tool for students, educators, researchers, and professionals alike.
  5. **To Explore the Capabilities of Pre-trained Language Models**
     Through this study, we explore the effectiveness of transformer-based models like multi-qa-mpnet-base-dot-v1 for semantic similarity and information retrieval in custom user-uploaded documents.
  6. **To Promote AI Accessibility through Lightweight Applications**
     The system is designed with minimal dependencies and simple architecture to encourage adoption and learning among students, developers, and researchers.

PROJECT SCOPE

The scope of the Chatbot for helping college students project encompasses the following key aspects:

The scope of this project revolves around the design, development, and deployment of an AI-powered chatbot system that can extract, understand, and respond to user queries based on the contents of uploaded PDF documents. This system integrates modern Natural Language Processing (NLP), document parsing, semantic similarity search, and a web-based user interface to create a complete and functional application.

- **1. Functional Scope**
  - **PDF Uploading**
    Users can upload PDF documents through the web interface. The system handles file storage and content extraction from these documents.
  - **Text Extraction and Preprocessing**
    The system uses PyPDF2 to read text from PDFs and then preprocesses the data by splitting it into smaller text chunks for efficient analysis.
  - **Semantic Embedding**
    Each chunk of text is converted into numerical embeddings using the SentenceTransformer model (multi-qa-mpnet-base-dot-v1), enabling the system to understand semantic meaning.
  - **FAISS Indexing for Search**
    The project utilizes Facebook AI Similarity Search (FAISS) to store and query embeddings efficiently, supporting fast retrieval of the most relevant chunks during user interactions.
  - **Natural Language Q&A Interface**
    The chatbot allows users to type questions in natural language. Based on semantic similarity, the system returns the best-matching response from the document content.
  - **Real-Time Interaction via Web Interface**
    Built using Flask, HTML, CSS, and JavaScript, the frontend supports file upload, question submission, and dynamic display of answers.

- **2. Technical Scope**
  - **Technologies Used**
    - Backend: Python, Flask
    - NLP & AI: Sentence Transformers, FAISS, NumPy
    - File Handling: PyPDF2
    - Frontend: HTML, CSS, JavaScript
  - **Model Scope**
    - The model used (multi-qa-mpnet-base-dot-v1) is trained for question-answering and semantic similarity tasks, ideal for matching user queries with relevant document text.
  - **Deployment Scope**
    - The project is designed to run locally or on any server that supports Flask applications. It does not require cloud-based APIs, ensuring faster processing and data privacy.

EXISTING SYSTEM

Before the development of this AI PDF Chatbot, several systems and methods existed to help users retrieve information from documents. These traditional or semi-automated systems rely on manual search techniques or limited keyword-based searching. While useful to a certain extent, these systems have several limitations in terms of speed, relevance, and user experience.

🔹 1. Manual Reading and Searching

In many organizations and academic settings, users are required to open PDF documents and manually read through pages to find specific information. This approach is:

- **Time-consuming**, especially for lengthy documents.
- **Error-prone**, as users may overlook important details.
- **Inefficient**, particularly when dealing with multiple documents or repeated queries.

🔹 2. Keyword-Based Search Tools

Some PDF readers or content management systems provide keyword search functionality, where users can input a term and locate its occurrences within the document.

Limitations:

- Only returns exact or near-exact keyword matches.
- Cannot understand the *context* or *semantic meaning* of the query.
- Fails when the user asks a question instead of a keyword.
- Cannot handle paraphrased or conceptually similar questions.

🔹 3. Traditional Chatbots or Rule-Based Systems

Some systems attempt to simulate intelligent document querying using predefined rules or scripted responses.

Drawbacks:

- Not adaptive to new or unknown documents.
- Responses are pre-programmed and limited to static logic.
- Poor at handling complex or open-ended queries.
- Cannot process file uploads or dynamically index content.

🔹 4. AI-based Search Engines (e.g., ElasticSearch with NLP plugins)

A few advanced enterprise tools use search engines combined with NLP for better indexing and querying.

Challenges:

- High setup and resource requirements.
- Often requires domain-specific customization.
- Limited accessibility for students and small-scale developers.

PROPOSED SYSTEM

The proposed system is an **AI-powered PDF Chatbot** that leverages Natural Language Processing (NLP), semantic search, and modern web technologies to enable users to interact with uploaded PDF documents using natural language queries. Unlike traditional keyword-based tools, this system understands the **semantic meaning** behind the user's question and returns the most relevant response from the document.

🔹 1. System Overview

The system allows users to:

- Upload a PDF file via a web interface.
- Ask questions in natural language about the content of the uploaded file.
- Receive precise answers based on the document's content using AI-powered semantic search.

This is achieved using:

- **Sentence Transformer models** for embedding both questions and document chunks.
- **FAISS** (Facebook AI Similarity Search) for fast and accurate nearest-neighbor search.
- **Flask** for building the backend server.
- **HTML, CSS, and JavaScript** for creating a user-friendly frontend.

🔹 2. Key Components and Features

- **Natural Language Understanding (NLU)**

The chatbot uses a pre-trained transformer model (multi-qa-mpnet-base-dot-v1) from **Sentence Transformers** to convert both the document chunks and user questions into high-dimensional embeddings. These embeddings represent the **semantic meaning** of text, allowing the system to understand and match similar concepts even if worded differently.

- **PDF Parsing and Chunking**

The uploaded PDF is processed using PyPDF2, which extracts raw text. The text is split into small, manageable chunks (e.g., 300 words each) to ensure better performance and relevance during semantic search.

- **Semantic Search with FAISS**

FAISS is used to build an **index of document embeddings**, which allows extremely fast search operations. When a user asks a question, the chatbot:

1. Converts the question into an embedding.
2. Searches for the most similar chunks in the FAISS index.
3. Returns the most relevant answer(s).

💬 Real-Time Web Chat Interface

The user interacts with the system through a web page where they can:

- Upload a PDF file.
- Ask questions.
- View real-time responses from the AI chatbot, rendered dynamically via JavaScript.

SYSTEM OVERVIEW

The **AI PDF Chatbot** is an intelligent question-answering system designed to help users interact with the contents of a PDF document through a conversational interface. The system combines **natural language processing**, **semantic search**, and a **web-based user interface** to deliver context-aware answers extracted from PDF files.
This section provides an overview of how the components of the system work together to achieve this goal.

🔹 1. Workflow Summary

The system operates through the following key steps:

1. **PDF Upload**
   The user selects a PDF file and uploads it through the web interface.
2. **Text Extraction and Processing**
   The backend extracts text from the PDF using PyPDF2, then splits the content into manageable chunks for processing.
3. **Embedding and Indexing**
   Each chunk is converted into a dense vector (embedding) using a Sentence Transformer model, and these embeddings are indexed using FAISS for fast similarity search.
4. **User Question Input**
   The user enters a question in natural language via the chat interface.
5. **Semantic Search for Answer Retrieval**
   The system converts the question into an embedding, searches for similar content in the FAISS index, and selects the most relevant chunk(s).
6. **Answer Display**
   The most appropriate answer is streamed back to the user in a conversational format.

IMPLEMENTATION ISSUE

IMPLEMENTATION ISSUE

During the development and deployment of the **AI PDF Chatbot** system, several technical and practical challenges were encountered. These issues span across the backend, frontend, AI model integration, and overall system performance. This section outlines the key problems, their causes, and how they were addressed or can be mitigated in future improvements.

🔹 1. PDF Text Extraction Inconsistencies

Issue:
Some PDFs, especially scanned documents or those with complex formatting, do not yield clean text when parsed using PyPDF2.

Cause:

- PyPDF2 cannot handle image-based or scanned PDFs (these require OCR).
- Text may be fragmented or encoded in a way that's difficult to extract.

Solution / Workaround:

- Only use machine-generated PDFs for testing and demo.
- For production, integrate OCR libraries like **Tesseract** or **pdfplumber** for better handling.

🔹 2. Long or Noisy Document Content

Issue:
When PDFs are very large or have repeated/irrelevant content (like footers, headers), the system sometimes retrieves less accurate answers.

Cause:

- Chunking is based on a fixed word count (e.g., 300), which can split sentences and paragraphs awkwardly.
- Repeated content across pages affects FAISS search quality.

Solution / Workaround:

- Improve text cleaning and preprocessing.
- Use NLP techniques like sentence boundary detection or key phrase extraction for better chunking.

🔹 3. FAISS Index Resetting

Issue:
The FAISS index is recreated every time a new PDF is uploaded. This means only one document is searchable at a time.

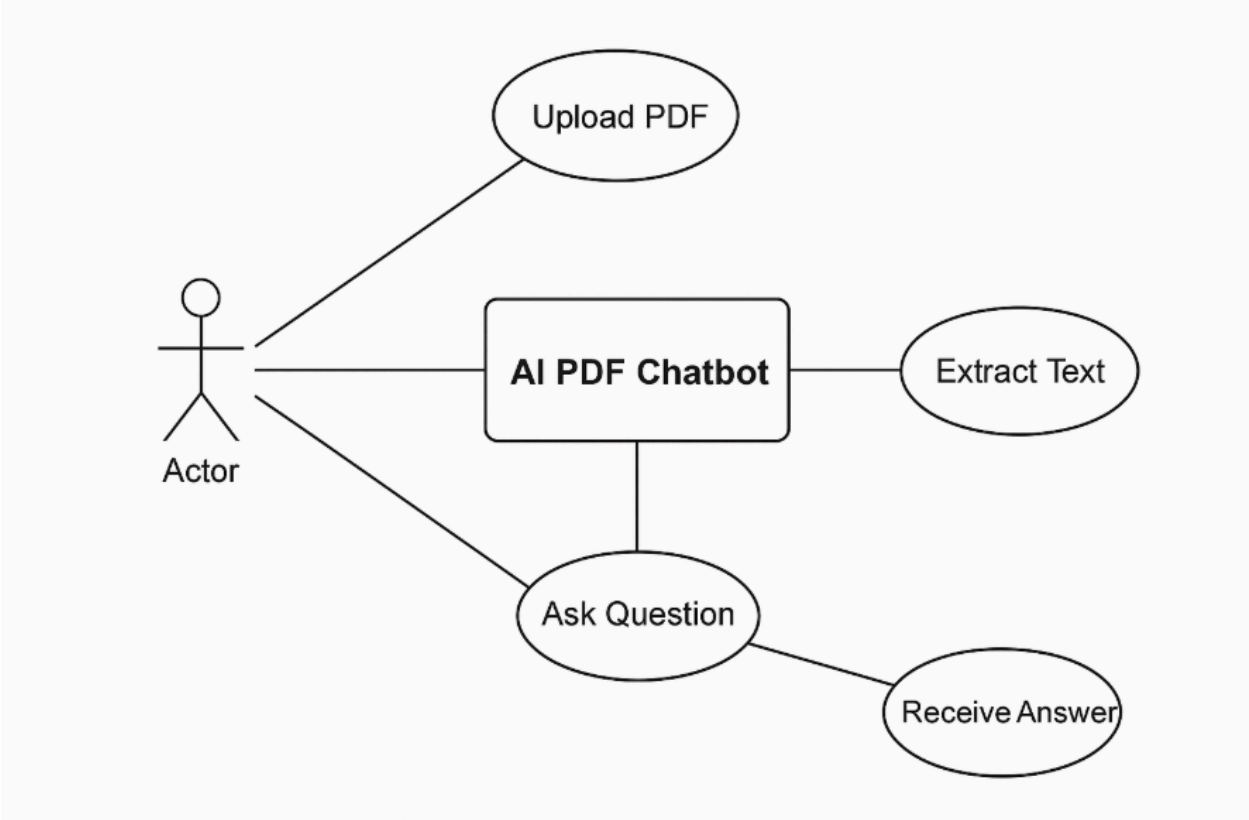
Cause:
The system stores the FAISS index and document chunks in global variables, which are overwritten on new uploads.

Solution / Future Enhancement:

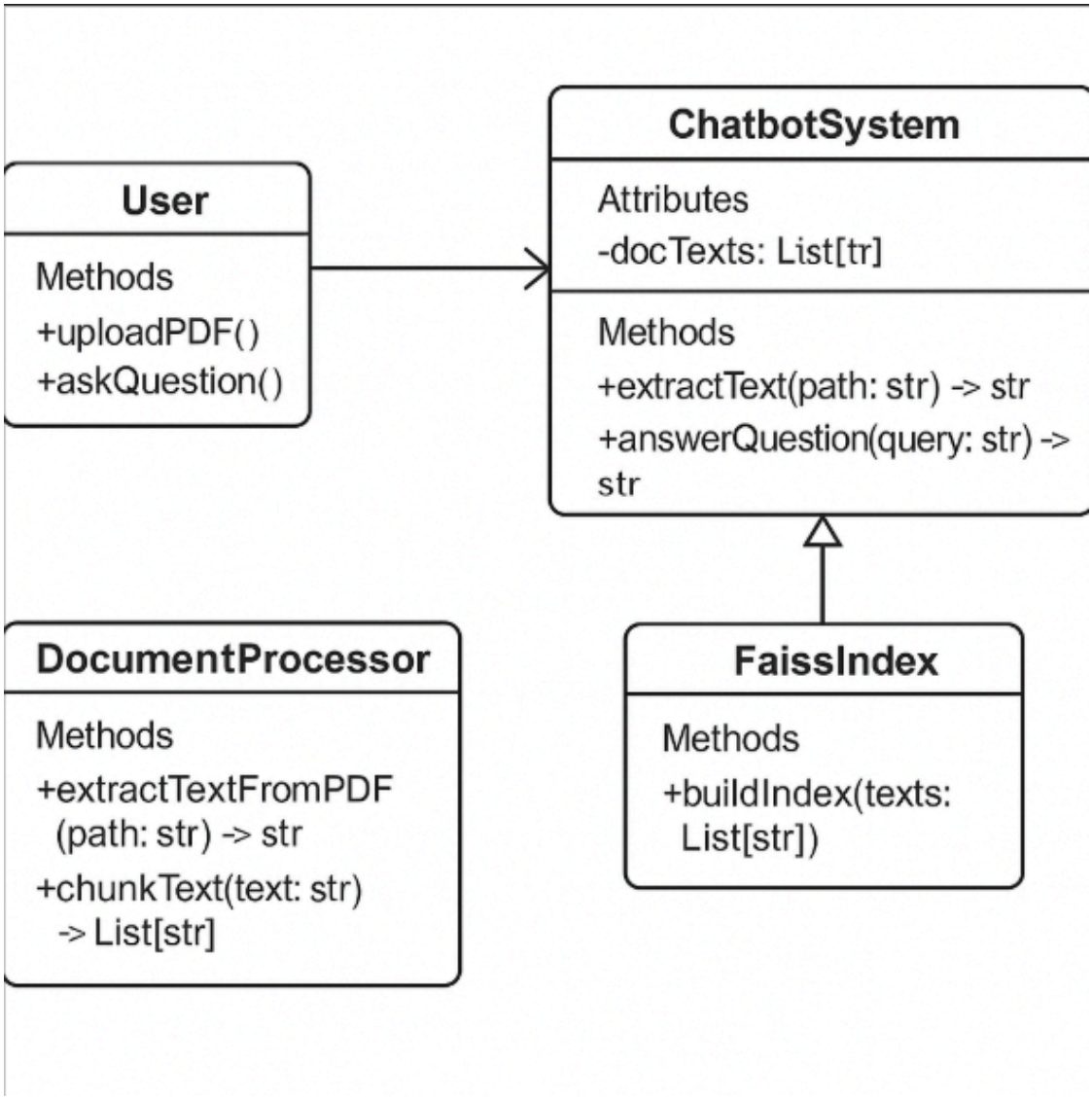
- Implement multi-document indexing and switching.
- Use session IDs or user-specific storage to maintain multiple indices.

SYSTEM DESIGN

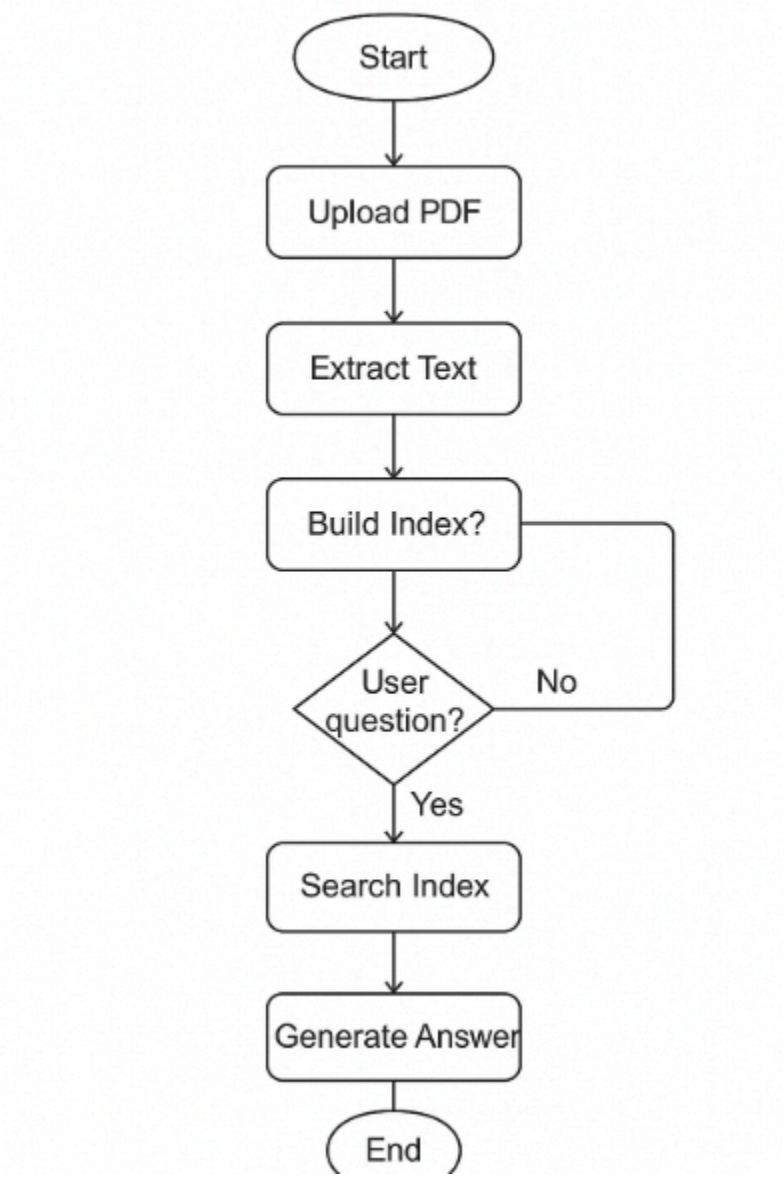
USE CASE DIAGRAM



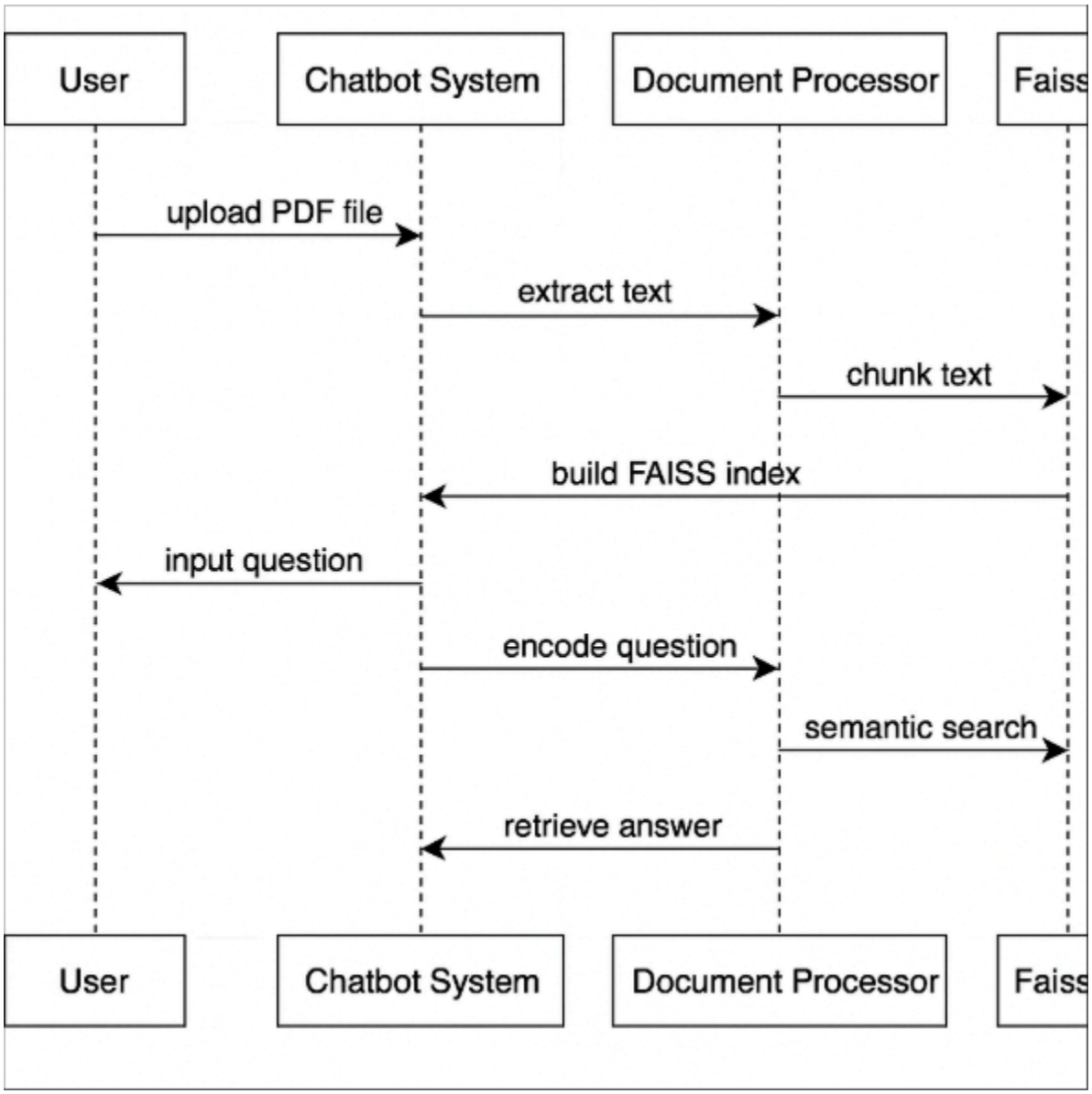
CLASS DIAGRAM



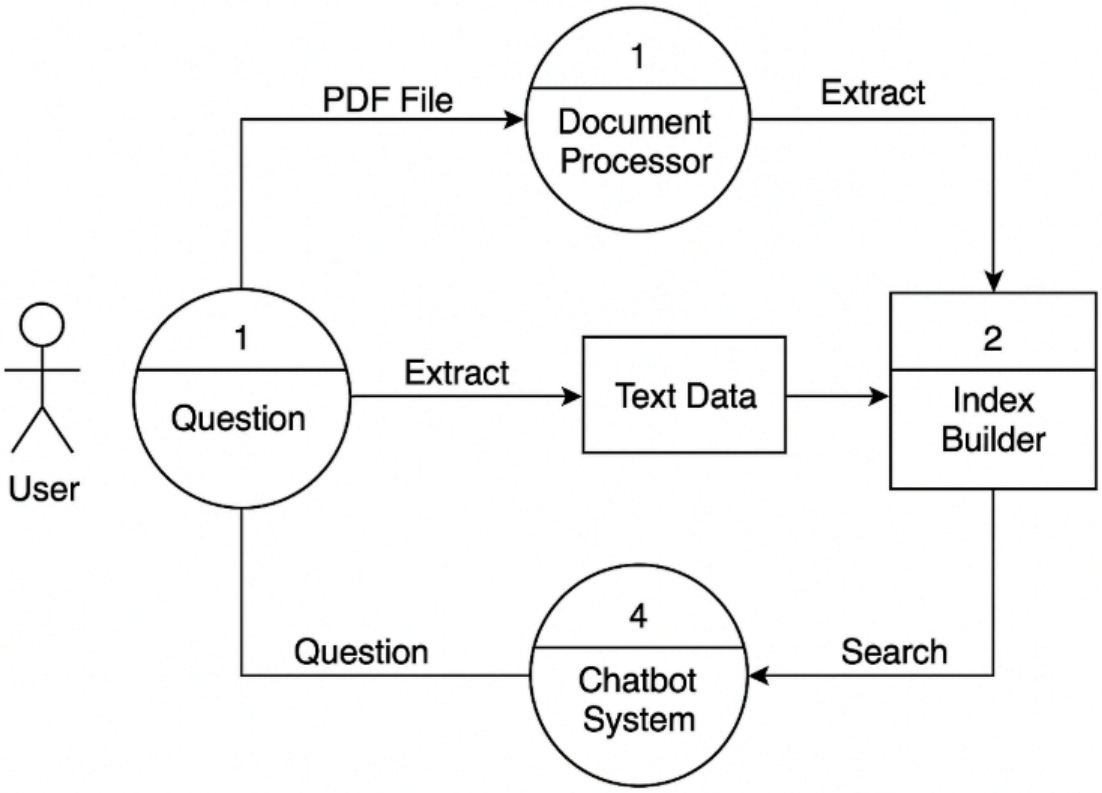
ACTIVITY DIAGRAM



SEQUENCE DIAGRAM

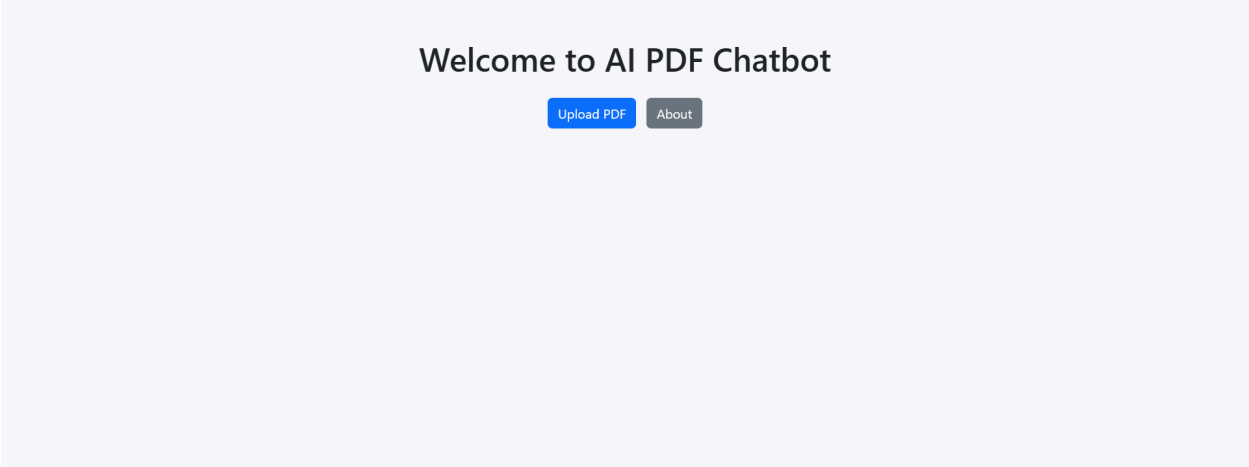


DATA FLOW DIAGRAM

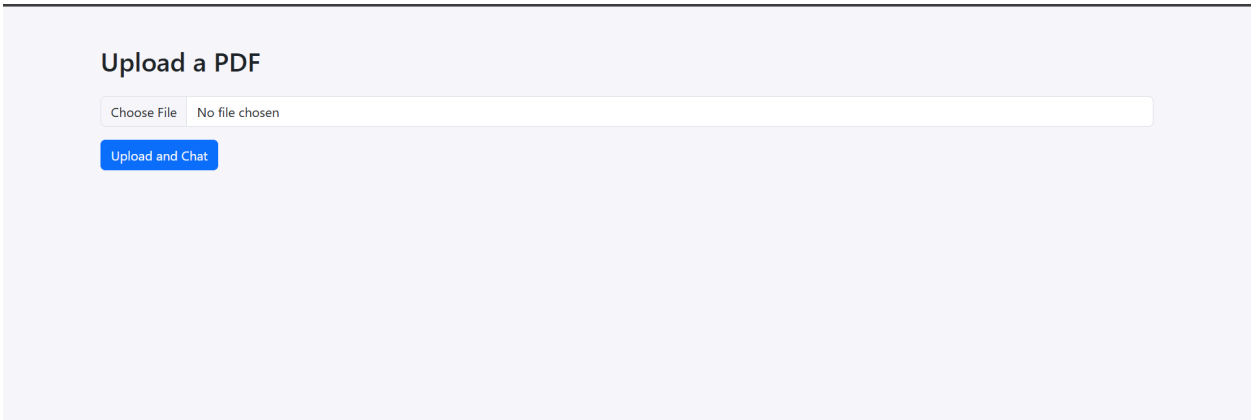


USER SCREENS

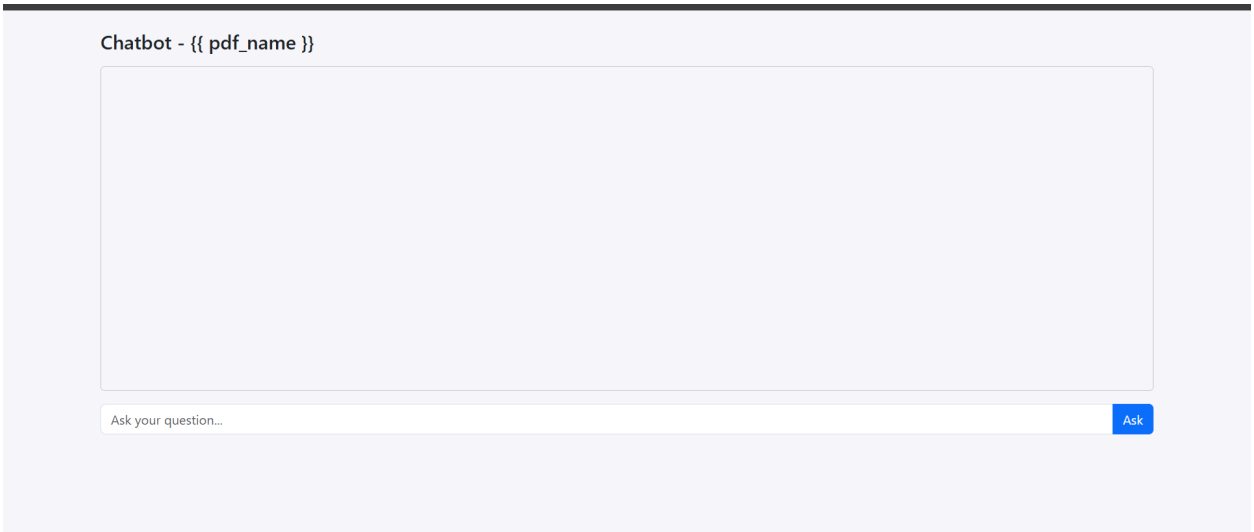
- **Home Page**

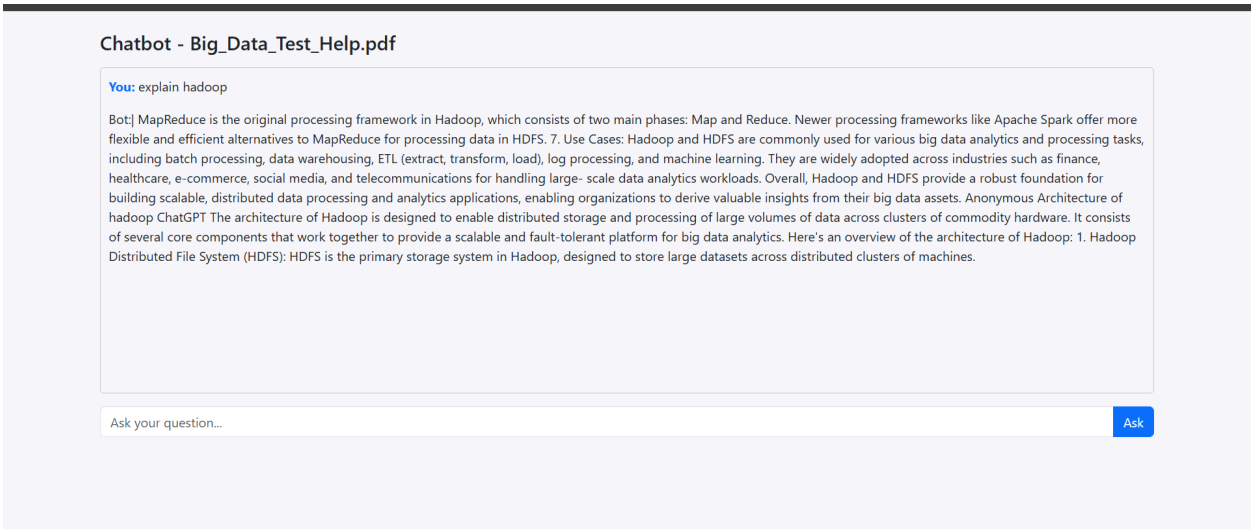


- **Upload Page**

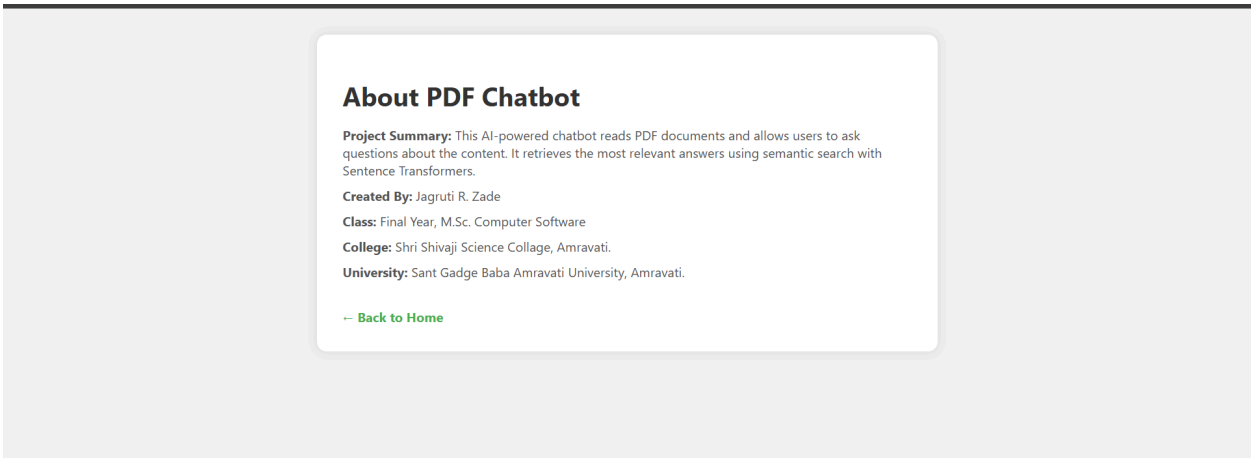


- **Ask Page**


Chatbot - {{ pdf_name }}
Ask your question...
Ask


Chatbot - Big_Data_Test_Help.pdf
You: explain hadoop
Bot:| MapReduce is the original processing framework in Hadoop, which consists of two main phases: Map and Reduce. Newer processing frameworks like Apache Spark offer more flexible and efficient alternatives to MapReduce for processing data in HDFS. 7. Use Cases: Hadoop and HDFS are commonly used for various big data analytics and processing tasks, including batch processing, data warehousing, ETL (extract, transform, load), log processing, and machine learning. They are widely adopted across industries such as finance, healthcare, e-commerce, social media, and telecommunications for handling large- scale data analytics workloads. Overall, Hadoop and HDFS provide a robust foundation for building scalable, distributed data processing and analytics applications, enabling organizations to derive valuable insights from their big data assets. Anonymous Architecture of hadoop ChatGPT The architecture of Hadoop is designed to enable distributed storage and processing of large volumes of data across clusters of commodity hardware. It consists of several core components that work together to provide a scalable and fault-tolerant platform for big data analytics. Here's an overview of the architecture of Hadoop: 1. Hadoop Distributed File System (HDFS): HDFS is the primary storage system in Hadoop, designed to store large datasets across distributed clusters of machines.
Ask your question...
Ask

- **About Page**


About PDF Chatbot
Project Summary: This AI-powered chatbot reads PDF documents and allows users to ask questions about the content. It retrieves the most relevant answers using semantic search with Sentence Transformers.
Created By: Jagruti R. Zade
Class: Final Year, M.Sc. Computer Software
College: Shri Shivaji Science Collage, Amravati.
University: Sant Gadge Baba Amravati University, Amravati.
← Back to Home

CODING

- **App.py**

```
from flask import Flask, render_template, request, jsonify, Response, redirect, url_for, session
import os
import fitz  # PyMuPDF
import numpy as np
from sentence_transformers import SentenceTransformer, util
from werkzeug.utils import secure_filename
import nltk
from nltk.tokenize import sent_tokenize

# Initialize Flask app
app = Flask(__name__)
app.secret_key = 'supersecretkey'
UPLOAD_FOLDER = 'uploads'
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Download NLTK sentence tokenizer
nltk.download('punkt')

# Load SentenceTransformer model
model = SentenceTransformer("multi-qa-mpnet-base-dot-v1")

# Global variables
doc_chunks = []
doc_embeddings = None

# ------------------ Utilities ------------------

def extract_text(filepath):
    """Extract full text from PDF using PyMuPDF."""
    doc = fitz.open(filepath)
    full_text = ""
    for page in doc:
        full_text += page.get_text()
    return full_text.strip()


def smart_chunk_text(text, max_tokens=200):
    """Chunk text based on complete sentences (not raw word count)."""
    sentences = sent_tokenize(text)
    chunks = []
    current_chunk = []
```

```
    total_tokens = 0

    for sentence in sentences:
        tokens = sentence.split()
        if total_tokens + len(tokens) > max_tokens:
            chunks.append(" ".join(current_chunk))
            current_chunk = []
            total_tokens = 0
        current_chunk.append(sentence)
        total_tokens += len(tokens)

    if current_chunk:
        chunks.append(" ".join(current_chunk))

    return chunks


def embed_chunks(chunks):
    """Create embeddings for all chunks."""
    return model.encode(chunks, convert_to_tensor=True, normalize_embeddings=True)


def refine_question(question):
    """Light preprocessing of the user question."""
    question = question.strip()
    if not question.endswith('?'):
        question += '?'
    return question.lower().capitalize()


def get_best_answer(question, k=5, threshold=0.5):
    """Return the most relevant chunk(s) based on cosine similarity."""
    global doc_chunks, doc_embeddings

    if doc_embeddings is None or len(doc_chunks) == 0:
        return "No document uploaded."

    question = refine_question(question)
    q_emb = model.encode(question, convert_to_tensor=True, normalize_embeddings=True)

    # Compute cosine similarity
    scores = util.cos_sim(q_emb, doc_embeddings)[0]
    top_indices = np.argsort(scores.cpu().numpy())[::-1]
```

```
    answers = []
    for idx in top_indices[:k]:
        if scores[idx] >= threshold:
            answers.append((doc_chunks[idx], float(scores[idx])))

    if not answers:
        return "Sorry, I couldn't find a confident answer in the document."

    return answers[0][0]  # Return best matching chunk


# ------------------ Routes ------------------

@app.route('/')
def home():
    return render_template('upload.html')


@app.route('/upload', methods=['POST'])
def upload():
    global doc_chunks, doc_embeddings

    file = request.files.get('file')
    if not file:
        return "No file uploaded", 400

    filename = secure_filename(file.filename)
    path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
    file.save(path)

    # Extract and process PDF content
    text = extract_text(path)
    doc_chunks = smart_chunk_text(text)
    doc_embeddings = embed_chunks(doc_chunks)
    session['pdf_name'] = filename

    return redirect(url_for('chat'))


@app.route('/chat')
def chat():
    pdf_name = session.get('pdf_name', 'No file uploaded')
    return render_template('ask.html', pdf_name=pdf_name)
```

```
@app.route('/ask', methods=['POST'])
def ask():
    data = request.get_json()
    question = data.get('question', '').strip()

    if not question:
        return jsonify({'error': 'Question is required'}), 400

    answer = get_best_answer(question)
    return Response(answer, content_type='text/plain')


# ------------------ Main ------------------

if __name__ == '__main__':
    app.run(debug=True)
```

- **Index.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>AI PDF Chatbot</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body class="bg-light text-center">
  <div class="container mt-5">
    <h1 class="mb-4">Welcome to AI PDF Chatbot</h1>
    <a href="/upload" class="btn btn-primary">Upload PDF</a>
    <a href="/about" class="btn btn-secondary ms-2">About</a>
  </div>
</body>
```

- **Upload.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Upload PDF</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
</head>
<body class="bg-light">
  <div class="container mt-5">
    <h2 class="mb-4">Upload a PDF</h2>
    <form action="/upload" method="post" enctype="multipart/form-data">
      <div class="mb-3">
        <input class="form-control" type="file" name="file" accept=".pdf" required>
      </div>
      <button class="btn btn-primary">Upload and Chat</button>
    </form>
  </div>
</body>
</html>
```

- **Ask.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Chat with PDF</title>
  <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap@5.3.0/dist/css/bootstrap.min.css">
  <style>
    #chat-box {
      height: 400px;
      overflow-y: auto;
      background: #f8f9fa;
      border: 1px solid #ccc;
      padding: 10px;
      border-radius: 5px;
    }
    .user-msg { font-weight: bold; color: #0d6efd; }
    .bot-msg.typing::after {
      content: '|';
      animation: blink 1s step-start 0s infinite;
    }
    @keyframes blink {
      50% { opacity: 0; }
    }
  </style>
</head>
<body class="bg-light">
  <div class="container mt-4">
    <h4 class="mb-3">Chatbot - {{ pdf_name }}</h4>
    <div id="chat-box" class="mb-3"></div>
    <form id="chat-form">
      <div class="input-group">
        <input type="text" id="question" class="form-control" placeholder="Ask your question..." required>
        <button type="submit" class="btn btn-primary">Ask</button>
      </div>
    </form>
  </div>

  <script>
    const form = document.getElementById("chat-form");
    const questionInput = document.getElementById("question");
    const chatBox = document.getElementById("chat-box");
```

```
  function addMessage(msg, isUser = false) {
    const p = document.createElement("p");
    p.innerHTML = isUser
      ? `<span class='user-msg'>You:</span> ${msg}`
      : `<span class='bot-msg typing'>Bot:</span> <span class='bot-response'></span>`;
    chatBox.appendChild(p);
    chatBox.scrollTop = chatBox.scrollHeight;
    return p.querySelector(".bot-response");
  }

  form.onsubmit = async (e) => {
    e.preventDefault();
    const question = questionInput.value;
    addMessage(question, true);
    const botMsg = addMessage("...");

    const res = await fetch("/ask", {
      method: "POST",
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ question })
    });

    const reader = res.body.getReader();
    const decoder = new TextDecoder();
    let text = "";

    while (true) {
      const { done, value } = await reader.read();
      if (done) break;
      text += decoder.decode(value);
      botMsg.textContent = text;
      chatBox.scrollTop = chatBox.scrollHeight;
    }
    botMsg.parentElement.classList.remove("typing");
    questionInput.value = "";
  };
 </script>
</body>
</html>
```

- **About.html**

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>About | PDF Chatbot</title>
  <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">
  <style>
    body {
      font-family: 'Segoe UI', sans-serif;
      background: #f0f0f0;
      margin: 0;
      padding: 2rem;
    }
    .about-container {
      max-width: 700px;
      margin: auto;
      background: white;
      padding: 2rem;
      border-radius: 12px;
      box-shadow: 0 0 10px rgba(0,0,0,0.1);
    }
    h1 {
      margin-bottom: 1rem;
      color: #333;
    }
    p {
      margin: 0.6rem 0;
      font-size: 16px;
      color: #555;
    }
    a {
      display: inline-block;
      margin-top: 1.5rem;
      text-decoration: none;
      color: #4CAF50;
      font-weight: bold;
    }
  </style>
</head>
<body>
  <div class="about-container">
    <h1>About PDF Chatbot</h1>
```

```
    <p><strong>Project Summary:</strong> This AI-powered chatbot reads PDF documents and allows users to ask questions about the content. It retrieves the most relevant answers using semantic search with Sentence Transformers.</p>
    <p><strong>Created By:</strong> Jagruti R. Zade</p>
    <p><strong>Class:</strong> Final Year, M.Sc. Computer Software</p>
    <p><strong>College:</strong> Shri Shivaji Science Collage, Amravati.</p>
    <p><strong>University:</strong> Sant Gadge Baba Amravati University, Amravati.</p>

    <a href="{{ url_for('home') }}">← Back to Home</a>
  </div>
</body>
</html>
```

- **Style.css**

```
#chatbox {
    background: #f0f0f0;
    border: 1px solid #ccc;
    height: 400px;
    overflow-y: auto;
    padding: 10px;
    border-radius: 10px;
    margin-bottom: 10px;
}

#input-area {
    display: flex;
    gap: 10px;
}

#input-area input {
    flex: 1;
    padding: 10px;
    border-radius: 8px;
    border: 1px solid #ccc;
    font-size: 16px;
}

#input-area button {
    padding: 10px 20px;
    border: none;
    background-color: #4CAF50;
    color: white;
    border-radius: 8px;
    font-size: 16px;
    cursor: pointer;
}

.message {
    max-width: 80%;
    margin: 8px 0;
    padding: 10px;
    border-radius: 8px;
    clear: both;
    line-height: 1.4;
}
```

```
.user {
    background-color: #d1e7dd;
    align-self: flex-end;
    float: right;
}

.bot {
    background-color: #f8d7da;
    align-self: flex-start;
    float: left;
}
```

CONCLUSION

FEATURES

Here are the Features of your AI PDF Chatbot Project explained in detail:

- 1. PDF Upload Functionality
  - Feature: Users can upload any PDF file from their device.
  - Why It's Important: Enables document-specific Q&A by extracting content from user-supplied files rather than a fixed database.

- 2. Automatic Text Extraction
  - Feature: Uses PyPDF2 to read and extract text from uploaded PDF documents.
  - Why It's Important: Converts unstructured PDF data into usable text for processing and answering queries.

- 3. Text Chunking
  - Feature: Large text is split into smaller, manageable chunks (e.g., 300 words).
  - Why It's Important: Improves the accuracy of semantic search by allowing the system to locate relevant sections instead of entire documents.

- 4. Semantic Embedding with SentenceTransformer
  - Feature: Uses the "multi-qa-mpnet-base-dot-v1" model to convert text and user questions into vector form.
  - Why It's Important: Translates natural language into numerical format that allows deep semantic understanding.

- 5. Smart Question Answering
  - Feature: Users can ask natural language questions, and the system finds the most contextually relevant answer.
  - Why It's Important: Makes it easy for users to get meaningful information without reading the entire PDF.

- 6. Streamed Answer Display
  - Feature: Bot answers are streamed word-by-word like a live chat.
  - Why It's Important: Enhances interactivity and mimics real-time conversation flow.

- 7. Interactive Web Interface
  - Feature: Built with Flask (backend) and HTML/CSS/JS (frontend).
  - Why It's Important: Provides a clean, simple, and user-friendly interface for uploads and chat.

RECOMMENDATION

The AI PDF Chatbot project presents a significant advancement in the way users interact with unstructured data in PDF documents. Based on the current implementation, performance, and user experience, the following recommendations are proposed to enhance the system's functionality, scalability, and impact:

- **1. Support for Multiple File Formats**
  - Extend the system to support additional document types such as .docx, .txt, and .html.
  - This would make the system more versatile and useful across various industries.

- **2. Integration of Optical Character Recognition (OCR)**
  - Enable OCR using libraries like **Tesseract** to extract text from scanned or image-based PDFs.
  - This will allow the chatbot to work with academic papers, handwritten notes, and scanned documents.

- **3. Multilingual Capabilities**
  - Integrate multilingual models or translation services to allow users to interact in languages other than English.
  - Useful for broader accessibility in diverse linguistic environments.

- **4. Answer Highlighting in PDF Context**
  - Provide the user with the page number and highlight the portion of the PDF from which the answer was derived.
  - This would increase the transparency and trust in the chatbot's responses.

- **5. User Authentication & History**
  - Add user login functionality and maintain a history of uploaded PDFs and questions asked.
  - This is beneficial for long-term users who want to track their interactions or retrieve old data.

- **6. Mobile-Friendly Interface**
  - Enhance the front-end design to be responsive and optimized for mobile devices and tablets.
  - Increases usability and accessibility.

- **7. Feedback System**
  - Allow users to rate answers or give feedback on the chatbot's accuracy.
  - Feedback can be used to further improve the response quality through retraining or fine-tuning.

- **8. Cloud Deployment for Scalability**
  - Deploy the system on cloud platforms like AWS, Azure, or Google Cloud to handle large-scale usage.
  - Ensures reliability, scalability, and broader accessibility.

REFERENCES

REFERENCES

1. Reimers, Nils and Gurevych, Iryna. (2019). *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks*. arXiv preprint arXiv:1908.10084. https://arxiv.org/abs/1908.10084
2. Johnson, Jeff, Douze, Matthijs, and Jégou, Hervé. (2017). *Billion-scale similarity search with GPUs*. IEEE Transactions on Big Data. https://arxiv.org/abs/1702.08734
3. PyPDF2 Documentation. (2024). *PyPDF2 Python Library for PDF Processing*. https://pypdf2.readthedocs.io
4. Facebook AI Research (FAISS). (2024). *FAISS: A library for efficient similarity search and clustering of dense vectors*. https://github.com/facebookresearch/faiss
5. Flask Documentation. (2024). *Flask: Web Development, One Drop at a Time*. https://flask.palletsprojects.com
6. Mozilla Developer Network (MDN). (2024). *JavaScript Fetch API Documentation*. https://developer.mozilla.org/en-US/docs/Web/API/Fetch_API
7. OpenAI. (2023). *Best Practices for Building AI-Powered Applications*. https://openai.com/research
8. Google Research. (2021). *Multilingual Universal Sentence Encoder*. https://tfhub.dev/google/universal-sentence-encoder-multilingual/3
9. W3Schools. (2024). *HTML, CSS, and JavaScript Tutorials*. https://www.w3schools.com
10. Stack Overflow. (2023). *Best practices in building PDF-based question answering systems*. https://stackoverflow.com