**COE-MAJOR-PROJECT**

**AI -POWERED SCIENTIFIC RESEARCH COMPANION**

**COE-AIDS**

A logo of a college of engineering

AI-generated content may be incorrect.

**Project Title:** AI-Powered Research Companion  
**Author:**

Snehanjali Kamatam-22B81A67B2

Srinidhi Poreddy-22B81A67B4  
**Institution:** CVR College of Engineering  
**Date:**6th August 2025

**Table of Contents**

1. **Introduction**

1.1. Project Overview

1.2. Purpose and Scope

1.3. Target Audience

1. **Key Features**
2. **System Architecture and Technology**

3.1. System Architecture

3.2. Technology Stack

1. **User Guide**
2. **Local Setup and Installation**

5.1. Prerequisites

5.2. Installation Steps

1. **Conclusion and Future Work**

6.1. Conclusion

6.2. Future Work

**1.Introduction:**

**1.1. Project Overview**

The "AI Research Companion" is a sophisticated web application designed to address a common challenge faced by researchers, students, and academics: the time-consuming process of extracting and synthesizing information from dense research papers and technical documents. The project provides an interactive chatbot interface where users can upload multiple PDF documents and engage in a natural language conversation to query information, understand complex topics, and discover new insights across the entire body of provided text.

**1.2. Purpose and Scope**

The primary purpose of this project is to accelerate the research process by leveraging modern AI techniques. It aims to transform the static nature of reading documents into a dynamic, conversational experience.

**The scope of the project includes:**

* Allowing users to upload between one and three PDF documents per session.
* Processing and indexing the content of these documents in real-time.
* Answering user questions based solely on the context provided within the uploaded documents.
* Proactively generating relevant follow-up questions to guide the user's inquiry.
* Providing a clean, intuitive, and modern user interface for seamless interaction.

**1.3. Target Audience**

The application is primarily intended for:

* **Researchers and Academics** who need to quickly review multiple papers.
* **Students** who are studying complex subjects and need help understanding technical documents.
* **Professionals** who need to extract key information from reports, manuals, or legal documents.

**2. Key Features:**

* **Dynamic Multi-PDF Upload**: Users can upload and analyze up to three PDF documents simultaneously, creating a unified knowledge base for their session.
* **Unified Contextual Understanding**: The AI backend processes and indexes all uploaded documents together, allowing it to answer questions that require synthesizing information from multiple sources.
* **Interactive Q&A**: A conversational interface allows users to ask direct questions in natural language and receive context-aware answers grounded in the provided text.
* **AI-Powered Suggestions**: To facilitate deeper exploration, the system automatically generates insightful topic suggestions after documents are uploaded and after each answer is provided.
* **Session Management**: A robust session management system ensures each user's documents and conversation are kept isolated. A "Clear Session" feature allows users to easily start over.
* **Modern User Interface**: The application features a polished, easy-on-the-eyes dark theme built with Bootstrap 5, ensuring a professional and user-friendly experience.

**3. System Architecture and Technology:**

**3.1. System Architecture**

The application is built on a client-server model and utilizes a Retrieval-Augmented Generation (RAG) pipeline to provide answers.

**The workflow is as follows:**

1. **Frontend Interaction**: The user interacts with the web interface built with HTML, CSS, and JavaScript. They use the interface to upload PDFs and submit questions.
2. **Backend Request Handling**: A Flask server on the backend receives these requests.
3. **Document Processing (RAG Pipeline)**:
   * **Loading**: When a PDF is uploaded, the Flask server uses PyPDFLoader from the LangChain library to load its content.
   * **Chunking**: The extracted text is split into smaller, manageable chunks using RecursiveCharacterTextSplitter.
   * **Embedding**: Each chunk of text is converted into a numerical vector representation (an embedding) using Google's gemini-embedding-001 model.
   * **Indexing**: These vectors are stored in a FAISS (Facebook AI Similarity Search) vector store, creating an efficient, searchable index of the document content. This index is stored in memory for the user's session.
4. **Query and Retrieval**: When a user asks a question, it is also converted into an embedding. The FAISS index is then queried to find the text chunks with the most similar embeddings (i.e., the most relevant context).
5. **Answer Generation**: The retrieved text chunks and the user's original question are passed to the Google Gemini LLM. The model is instructed to formulate a final answer based only on the provided context.
6. **Response**: The final answer and new suggestions are sent back to the frontend to be displayed to the user.

**3.2. Technology Stack**

|  |  |  |
| --- | --- | --- |
| **Category** | **Technology** | **Purpose** |
| **Backend** | Python 3.9+ | Core programming language for the server-side logic. |
|  | Flask | A lightweight web framework for handling API requests. |
|  | LangChain | A framework for developing applications with LLMs. |
|  | Google Generative AI (Gemini) | Used for text embeddings and answer generation. |
|  | FAISS | A library for efficient similarity search in vector stores. |
| **Frontend** | HTML, CSS, JavaScript | The foundation of the user interface. |
|  | Bootstrap 5 | A CSS framework for creating the modern, responsive UI. |
| **Tooling** | Git | Version control. |
|  |  |  | Virtual |

**4. User Guide:**

1. **Navigate to the Application**: Open the application in a web browser.
2. **Upload Documents**: Click the "Choose File" button to upload your first PDF. You can repeat this process to add up to three PDFs to the current session. The names of active documents will be displayed.
3. **Start the Conversation**: Once a document is processed, the chat input will become active. You can either:
   * Type your own question into the input box and press "Send".
   * Click on one of the AI-generated topic suggestions that appear.
4. **Continue the Discussion**: After each answer, the AI will provide new follow-up questions to keep the conversation going.
5. **Start Over**: Click the "Clear Session" button at any time to remove all uploaded documents and reset the application.

**5. Local Setup and Installation:**

**5.1. Prerequisites**

* Python 3.9 or higher
* Git version control
* A Google AI API Key

**5.2. Installation Steps**

1. **Clone the Repository**:
2. git clone <your-repository-url>
3. cd <repository-folder-name>
4. **Create and Activate a Virtual Environment**:
   * **On Windows:**
   * python -m venv venv
   * .\venv\Scripts\activate
   * **On macOS & Linux:**
   * python3 -m venv venv
   * source venv/bin/activate
5. **Install Dependencies**:
6. pip install -r requirements.txt
7. **Set Up Environment Variables**: Create a file named .env in the project root and add your Google API key:
8. GOOGLE\_API\_KEY="YOUR\_ACTUAL\_API\_KEY\_GOES\_HERE"
9. **Run the Application**:
10. python app.py

The application will be available at <http://127.0.0.1:5000>.

Demo:



**6. Conclusion and Future Work:**

**6.1. Conclusion**

The AI Research Companion successfully demonstrates the power of Retrieval-Augmented Generation (RAG) in creating practical, high-value tools. By combining a powerful language model with a user's specific documents, it provides a reliable and efficient way to interact with complex information, ultimately saving time and enhancing the research process.

**6.2. Future Work**

The project has a strong foundation with several avenues for future enhancement:

* **Support for More File Types**: Extend functionality to support .docx, .txt, and even website URLs.
* **Chat History Persistence**: Implement a database (e.g., SQLite or Firestore) to save conversations, allowing users to return to them later.
* **Streaming Responses**: Modify the backend to stream the AI's response token-by-token for a more interactive, real-time feel.
* **Highlighting Sources**: When an answer is provided, highlight the specific text chunks from the source documents that were used to generate it.
* **User Accounts**: Add user authentication to allow for private document storage and saved sessions.