# Assignment 1 - PDF and WEB scraping with LLM Integration

Pranjal Mahajan (002375449)

Srushti Patil (002345025)

Ram Putcha (002304724)

# **Introduction**

The tremendous expansion of web content has produced an increasing need for efficient information retrieval and document processing. The users might need to deal with enormous amounts of PDF files containing critical information, but it can be an inefficient and time-consuming task to derive insights manually. This project extends the foundation laid in Assignment 1 to develop a Streamlit-based web application with Large Language Models (LLMs) as an embedded entity with FastAPI as a backend and LiteLLM as an API management module. The app will allow users to Upload or select existing parsed PDFs, Generate AI-powered summaries of document content, Pose questions and receive corresponding answers from the document, Interact with multiple LLM providers as per user’s choice (GPT-4o, Gemini, Claude, etc.).

With the combination of FastAPI and LiteLLM, the platform will provide a scalable, modular, cloud-deployable solution, and enhance document summarization and Q&A to be made more accessible and effective.

# **Problem Statement**

As the use of digital documentation increases, users frequently encounter difficulties effectively retrieving and analyzing data from sizable PDF files. Manual reading, searching, and summarization are necessary for traditional methods, which are laborious, ineffective, non-scalable, and challenging to implement. We suggest a Streamlit-based web application that makes use of Large Language Models (LLMs) through FastAPI and LiteLLM in order to overcome these difficulties. Information retrieval will be smooth and effective because of this system's ability to automate document summaries and enable natural language querying.

# **Project Goals**

The project aims to develop an *AI-powered PDF processing application* with the following goals:

**Streamlit Frontend Development**

* + Build an intuitive UI to upload/select PDF documents.
  + Allow users to choose an LLM model (GPT-4o, Gemini, Claude, etc.).
  + Implement interactive input fields for summarization and Q&A.
  + Display AI-generated summaries and answers in a structured manner.

**Backend Development with FastAPI**

* + Set up REST API endpoints for document processing and AI queries.
  + Enable file handling for PDF uploads and previously parsed documents.
  + Process user questions and retrieve context-aware answers from LLMs.
  + Use Redis Streams for efficient backend communication.

**Integration with LLMs via LiteLLM**

* + Implement LiteLLM middleware to manage multiple AI models.
  + Monitor and log input/output token usage for cost tracking.
  + Ensure error handling and logging for API calls.

**Cloud Deployment & Scalability**

* + Containerize the entire application using Docker Compose.
  + Deploy the solution on the cloud for accessibility and scalability.
  + Ensure modular design for future expansions and improvements.

# **Proof of Concept**

To validate the feasibility of the proposed solution, we will develop a prototype with a structured frontend, backend, AI integration, and deployment strategy. This proof of concept will demonstrate how users can seamlessly upload and interact with PDF documents using AI-powered summarization and Q&A functionalities.

The frontend will be built using Streamlit, providing a simple and interactive interface. Users will be able to upload new PDFs or select previously processed ones, choose an LLM model (such as GPT-4o, Gemini, or Claude), and input questions related to the document. The application will have clearly defined sections to display AI-generated summaries and responses.

For the backend, we will implement FastAPI to handle API interactions. It will provide structured REST API endpoints that allow users to:

* Upload PDFs for processing.
* Retrieve previously parsed content.
* Generate AI-powered document summaries.
* Ask questions and receive answers based on the document content.

To ensure efficient communication between the frontend and the AI models, we will integrate LiteLLM as a middleware. This will:

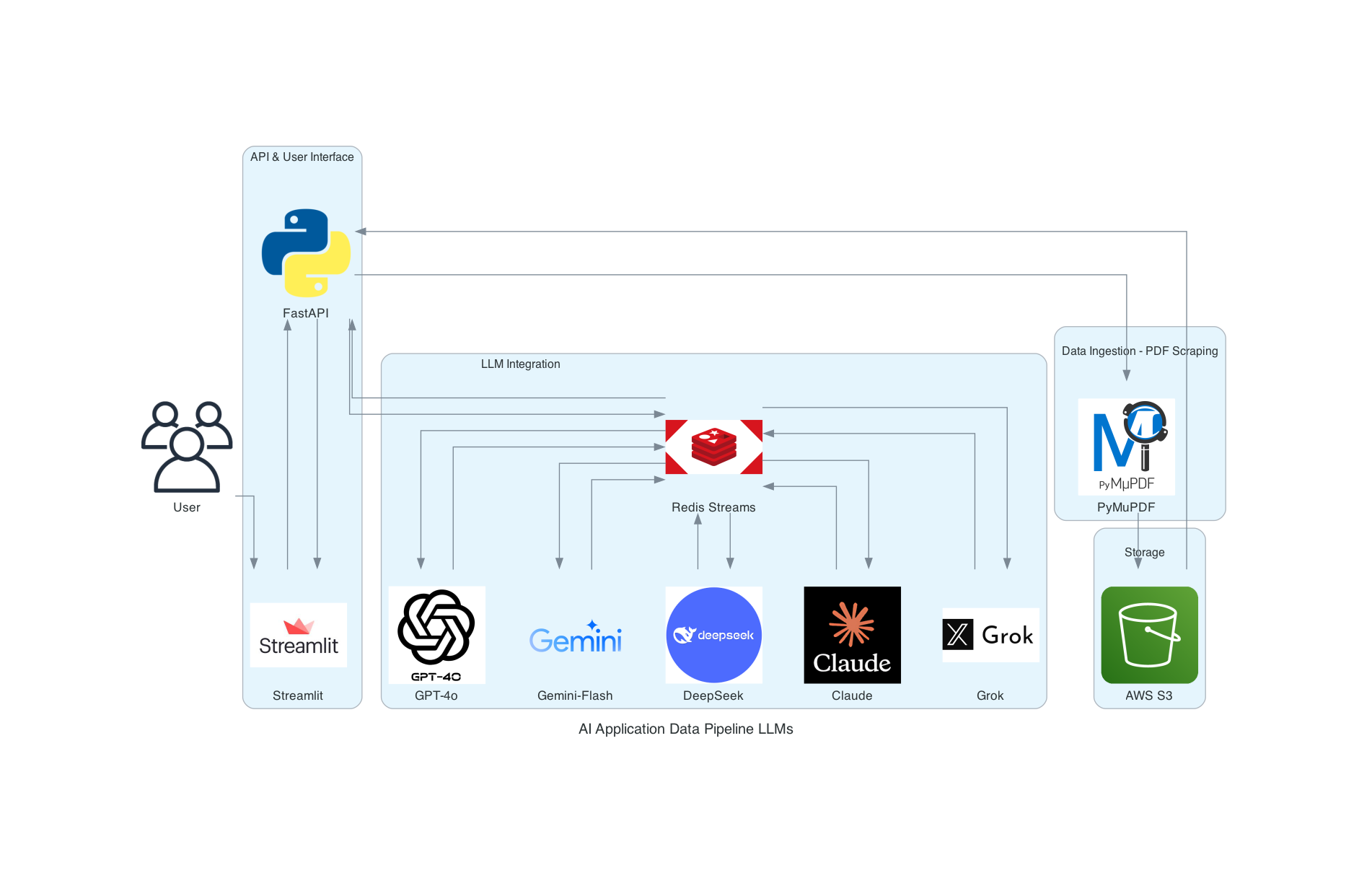
* Facilitate seamless connections with multiple LLM providers.
* Monitor API usage and token costs for optimized performance.
* Handle errors and log API interactions to improve reliability.

The entire system will be containerized using Docker Compose and deployed on the cloud, ensuring scalability and ease of access. This approach will allow for future enhancements, including multi-user support, improved AI model selection, and advanced document processing capabilities.

Through this PoC, we will demonstrate how an automated AI-driven document interaction system can significantly improve efficiency in summarizing and extracting insights from textual data.

# **Architecture Diagram**

This architecture diagram represents an **AI-powered document processing system** integrating **Streamlit, FastAPI, Redis Streams, LLMs, and PDF ingestion tools** for summarization and Q&A functionalities.



### **1. User Interaction Layer**

* Users interact with the system through a Streamlit-based UI.
* Streamlit provides a web-based front-end, allowing users to upload PDFs, request document summaries, and ask questions.
* FastAPI serves as the backend, handling API requests between Streamlit and other system components.

### **2. LLM Integration & Processing**

* FastAPI interacts with Redis Streams, which acts as a message broker for AI model queries.
* The system supports multiple LLMs for flexibility:
  + **GPT-4o** (OpenAI)
  + **Gemini-Flash** (Google)
  + **DeepSeek**
  + **Claude** (Anthropic)
  + **Grok** (xAI)
* Redis Streams manages the flow of data, ensuring efficient LLM interactions.

### **3. PDF Ingestion & Storage**

* PyMuPDF is used for PDF parsing and text extraction.
* Extracted text is processed and stored for later use.
* AWS S3 is used for storing uploaded PDFs, ensuring accessibility and scalability.

### **4. Workflow Summary**

1. User uploads/selects a PDF → Streamlit UI.
2. FastAPI processes the request → Sends data for parsing.
3. PyMuPDF extracts text → Stores it in AWS S3.
4. User requests a summary or asks a question.
5. FastAPI sends the request to Redis Streams.
6. Redis routes the request to the selected LLM.
7. LLM processes and returns the response.
8. FastAPI sends the response back to Streamlit for display.

# **Walkthrough of the Assignment**

## **Step 1: Clone the Repository**

1. Open a terminal or command prompt.

Run the following command to clone the repository:  
git clone <repository\_url>

1. cd Assignment\_4

## **Step 2: Create a Virtual Environment**

Create and activate a virtual environment:  
python -m venv venv

1. source venv/bin/activate # On Windows: venv\Scripts\activate

## **Step 3: Install Dependencies**

1. Install the required dependencies:  
   pip install -r requirements.txt

## **Step 4: Configure Environment Variables**

1. Create a .env file in the root directory.

Add required credentials such as API keys for AWS, LLM’s (GPT-4o, Gemini, Claude, Grok, Deepseek), etc.

AWS\_ACCESS\_KEY=<your\_access\_key>

AWS\_SECRET\_KEY=<your\_secret\_key>

## **Step 5: Run the Redis Server**

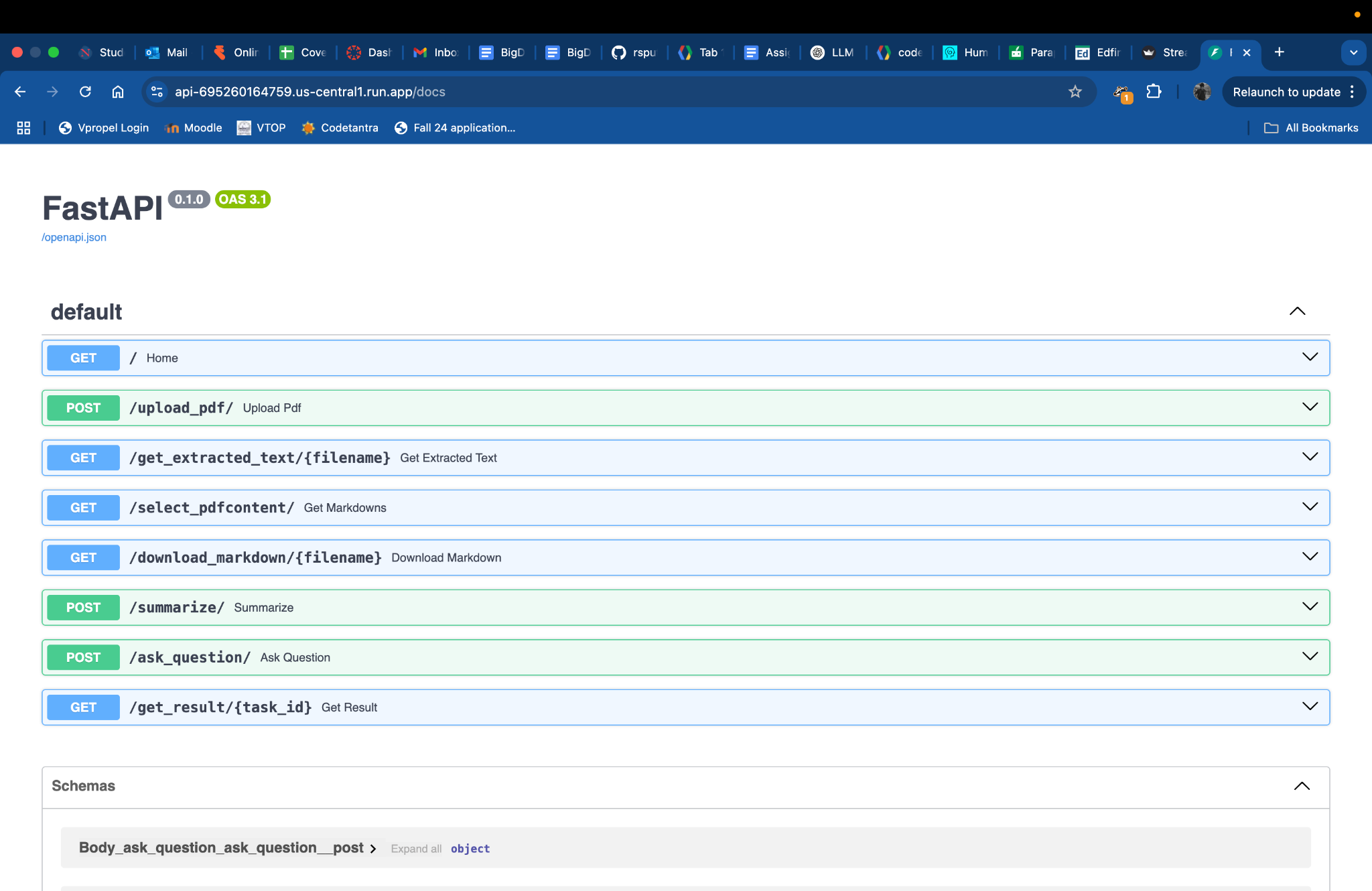
1. Start the Redis Server

docker run -d --name redis -p 6379:6379 redis:latest

1. The successful output should show “Redis”

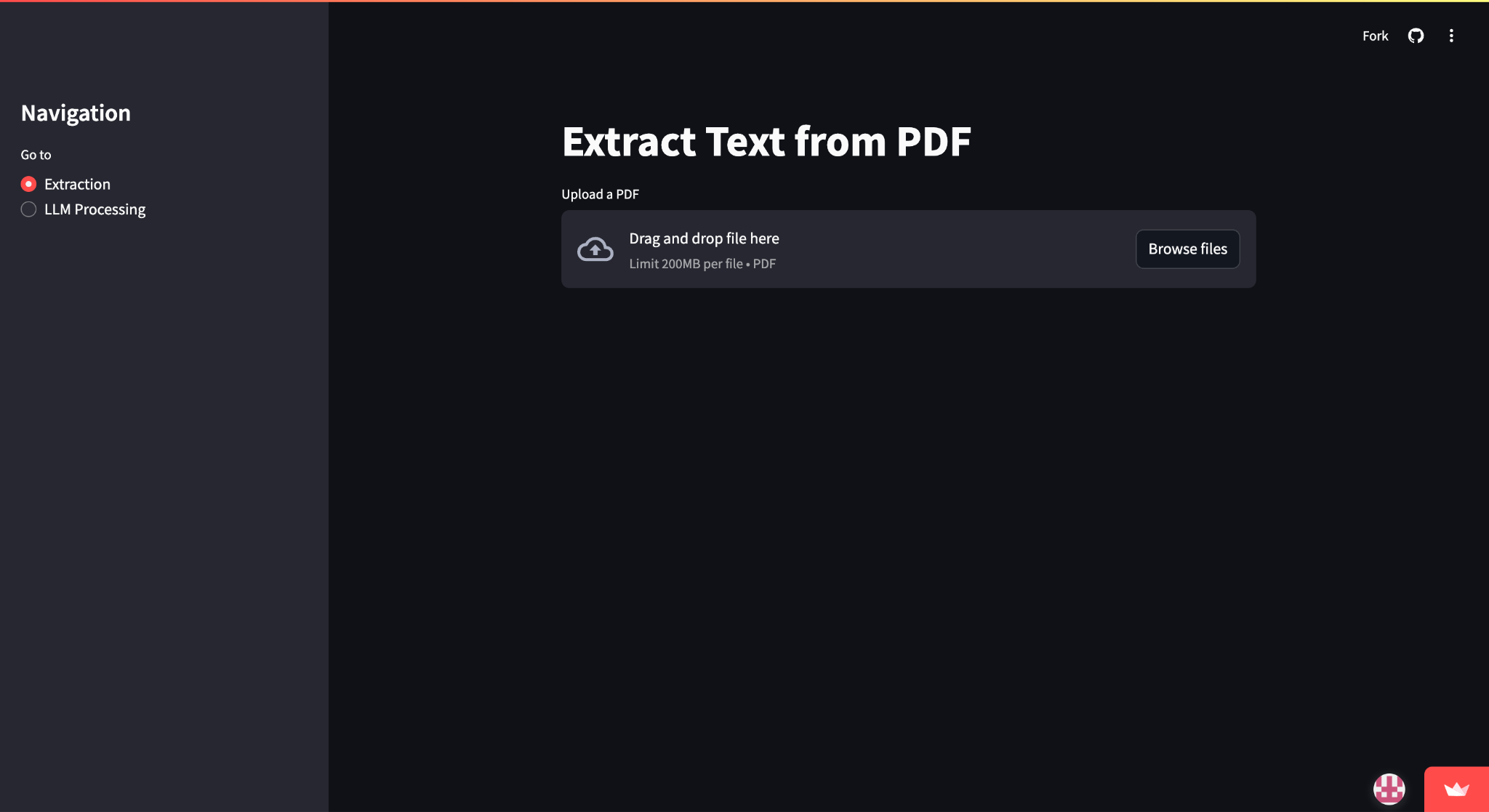
## **Step 6: Run the Backend Server**

1. Start the FastAPI backend server:  
   uvicorn backend.app:app --reload
2. The API will be available at <http://127.0.0.1:8000>.



## **Step 7: Run the Frontend Dashboard**

1. Start the Streamlit dashboard:  
   streamlit run frontend/dashboard.py
2. Open the displayed local URL to access the dashboard.



## **Step 8: Use the Application**

1. *Choose an LLM Model:* Select an LLM model (e.g., GPT-4, Gemini, Claude) to use for summarization and Q&A.
2. *Upload a PDF or Enter a URL:* Upload a PDF file or enter a website URL, then click "Process" to extract text and structured content.
3. *Summarize the Document:* Click "Summarize" to generate and display a summary of the extracted content.
4. *Ask Questions on the Document:* Enter a question, click "Ask Question," and view the AI-generated response.
5. *View Token Usage & Cost:* Check token usage, cost per token, and total processing cost for each request.
6. *Download Processed Data:* Select a processed document and click "Download Markdown" to retrieve the extracted content.
7. *View Logs & Task Status:* Monitor the processing status of summarization and Q&A tasks, and wait for results if still processing.

## **Step 9: Verify API Endpoints**

1. Use Postman or a browser to test API endpoints.
2. Example API request for Uploading a PDF:  
   curl -X POST "http://127.0.0.1:8000/upload\_pdf/" -F "file=@sample.pdf"
3. Example API request for Retrieving Extracted Text from Redis:  
   curl -X GET "http://127.0.0.1:8000/get\_extracted\_text/sample.md"
4. Example API request for Listing Stored Markdown Files in S3:  
   curl -X GET "http://127.0.0.1:8000/select\_pdfcontent/"
5. Example API request for Downloading an Extracted Markdown File:  
   curl -X GET "http://127.0.0.1:8000/download\_markdown/sample.md"
6. Example API request for Summarizing a PDF using an LLM Model:  
   curl -X POST "http://127.0.0.1:8000/summarize/" -F "pdf\_name=sample.md" -F "llm=gpt-4"
7. Example API request for Asking a Question on a PDF using an LLM Model:  
   curl -X POST "http://127.0.0.1:8000/ask\_question/" -F "pdf\_name=sample.md" -F "llm=gemini" -F "question=What are the key points?"
8. Example API request for Fetching the Result of an AI Task (Summarization or Q&A):  
   curl -X GET "http://127.0.0.1:8000/get\_result/task-abc123"

## **Common Issues and Troubleshooting**

1. Chrome WebDriver Not Found: Ensure ChromeDriver is installed, compatible with your Chrome version, and added to the system PATH.
2. Streamlit Dashboard Not Opening: Check for firewall restrictions, manually open the displayed URL, or restart the application.
3. API Requests Failing: Verify that the FastAPI server is running at http://127.0.0.1:8000, and ensure there are no port conflicts.
4. S3 Upload Fails: Confirm AWS credentials (AWS\_ACCESS\_KEY\_ID, AWS\_SECRET\_ACCESS\_KEY) are correctly set in the environment variables and that the bucket exists.
5. Redis Connection Error: Ensure Redis is installed and running locally (redis-server), and check the correct host and port settings.
6. PDF Extraction Not Working: Check if the uploaded PDF is valid and ensure PyMuPDF and Azure AI Form Recognizer are installed and configured correctly.
7. Summarization or Q&A Not Processing: Verify that the request is added to Redis (redis-cli MONITOR to check stream data) and that the worker process is running.
8. Task Results Not Showing: Confirm that the result exists in Redis (redis-cli GET response:<task\_id>) and wait for background processing to complete.
9. LLM Model Not Responding: Ensure the selected LLM is available, API keys are valid, and request limits are not exceeded.
10. Token Usage & Cost Not Displayed: Verify that the LLM response includes token usage metadata and check for issues in cost calculation logic.

# **Application Workflow**

This section describes the complete data flow from frontend input to backend processing and final output, including token usage tracking and cost estimation for each LLM model.

### 1. User Input (Frontend - Streamlit)

* The user uploads a PDF via the Streamlit dashboard.
* The user selects an LLM model (GPT-4o, Gemini, Claude, DeepSeek, or Grok).
* The user enters a query (e.g., "Summarize this document" or "Extract key points").
* The request is submitted to the backend for processing.

### 

### 2. Request Sent (Frontend → Backend - FastAPI)

* The frontend (Streamlit) sends the request to the FastAPI backend via API calls.
* The request contains the PDF file, user query, and selected LLM model.

### 

### 3. Processing (Backend - FastAPI & Redis Streams)

Document Processing:

* If the PDF is new, PyMuPDF extracts the text for processing.
* The extracted text is stored in AWS S3 for future access.

Query Routing:

* The FastAPI backend pushes the query & selected LLM to Redis Streams.
* Redis Streams routes the request to the chosen AI model for processing.

AI Model Processing (LLM Selection via LiteLLM):

* LiteLLM middleware identifies the user-selected model and sends the request to the respective LLM API (GPT-4o, Gemini, Claude, DeepSeek, or Grok).
* The LLM processes the request and generates a response.
* The number of tokens used for both input and output is calculated.
* The processing cost per token is retrieved for the selected model.

### 

### 4. Data Storage (AWS S3 & Redis Streams)

* Extracted PDF data and processed AI responses are stored in AWS S3 for future retrieval.
* Redis Streams ensures efficient handling of responses and request queues.

### 

### 5. Data Retrieval (API Layer - FastAPI)

* The backend provides an API endpoint to retrieve processed data.
* If the same document is uploaded again, FastAPI fetches pre-processed data from AWS S3, reducing redundant processing.

### 

### 6. Visualization (Frontend - Streamlit Dashboard)

* The frontend (Streamlit) fetches the AI-generated response via API calls.
* The processed results are displayed interactively in the dashboard.

The output includes:

* Generated response from the selected LLM model.
* Number of input and output tokens used.
* Cost per token for the selected model.
* Total processing cost for the request.

Users can compare LLM models by switching models and analyzing their performance and cost-effectiveness.

# **References**

* [PyMuPDF Documentation](https://pymupdf.readthedocs.io/)
* [AWS S3 Storage](https://aws.amazon.com/s3/)
* [OpenAI GPT-4o Documentation](https://platform.openai.com/docs/guides/gpt/gpt-models)
* [Google Gemini 2.0 Flash Documentation](https://docs.aimlapi.com/api-references/text-models-llm/google/gemini-2.0-flash-exp)
* [DeepSeek LLM Documentation](https://api-docs.deepseek.com/)
* [Anthropic Claude Documentation](https://docs.anthropic.com/en/docs/welcome)
* [xAI Grok Documentation](https://docs.x.ai/docs/tutorial)

# **Disclosures**

WE ATTEST THAT WE HAVEN’T USED ANY OTHER STUDENTS’ WORK IN OUR

ASSIGNMENT AND ABIDE BY THE POLICIES LISTED IN THE STUDENT HANDBOOK

| Pranjal Mahajan(002375449) | 33.33% |
| --- | --- |
| Srushti Patil (002345025) | 33.33% |
| Ram Putcha (002304724 | 33.33% |

* AI Usage disclosure

AiUseDisclousure :

* We have utilized AI to assist us in the following areas:
* Exploring tools for extracting data from PDFs and websites.
* Understanding how to deploy FastAPI for rendering and the necessary deployment steps.
* Learning how to create S3 buckets in AWS.
* Deploying a Streamlit frontend application to Streamlit Community Cloud.
* Connecting frontend and backend deployments to work seamlessly together