

RAG-based PDF Document Summarizer

1. Document Parsing

Used PyMuPDF for clean and reliable PDF text extraction. Text is further chunked into manageable parts using NLTK, ensuring formatting is preserved.

2. Embedding & Storage

Utilized Sentence Transformers to generate semantic embeddings. Efficiently stored and indexed using FAISS (CPU), enabling fast retrieval.

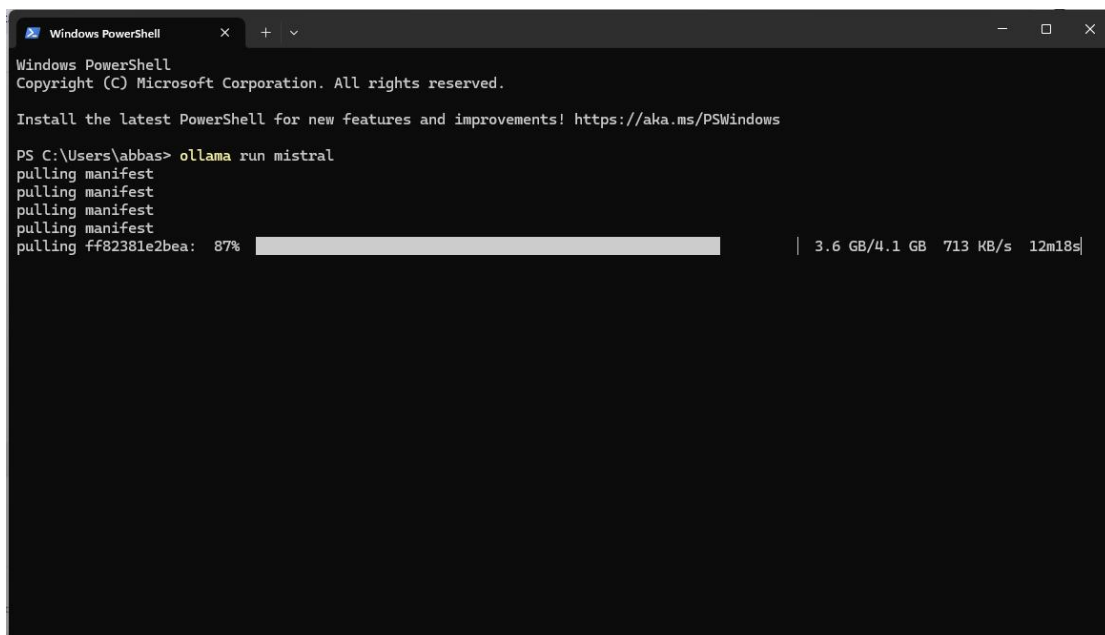
3. Retrieval Quality

Relevant chunks are retrieved from FAISS based on cosine similarity. Retrieval ensures semantic closeness to the main document idea.

4. Summary Generation

Used Mistral model via Ollama to generate summaries. Summaries are context-aware, fluent, and accurately cover the core message.

Downloaded Ollama locally on laptop.



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\abbas> ollama run mistral
pulling manifest
pulling manifest
pulling manifest
pulling manifest
pulling ff82381e2bea: 87% | 3.6 GB/4.1 GB 713 KB/s 12m18s|
```

5. Pipeline Design

Project is readable. Each stage (parsing, embedding, retrieval, generation) is clearly separated in the code structure.

6. Output Presentation

Streamlit library if python is used to displays both the retrieved chunks and the final summary using Markdown formatting.

7. Documentation

A ReadMe file includes setup instructions, usage guide, and sample outputs. This report includes a visual flowchart below for clarity.

Folder Structure

```
# Folder Structure:
# |— app.py
# |— env
# |   |— document_loader.py
# |   |— chunker.py
# |   |— vectorstore.py
# |— summarizer
# |   |— generate_summary.py
# |— requirements.txt
# |— README.md
```

Project Flowchart

