Project Overview Report

* Project Setup and Configuration : Follow Github README instruction to setup the project on your system.
* Output Result Screenshot in Result\_images Folder.
* Steps Taken for Each Component in code :

1. PDF Extraction and Chunking
2. PDF Fetching from Database
3. Implemented fetch\_pdf\_from\_db function to retrieve PDF files from PostgreSQL.
4. PDF Chunking and Text Extraction: Utilized PyPDFLoader to load and parse PDF documents (load\_chunk\_pdf function).
5. Segmented documents into smaller chunks using CharacterTextSplitter.
6. Vectorization and Embeddings: Created embeddings with OllamaEmbeddings(model='nomic-embed-text').
7. Established a vector store (Chroma) with Chroma.from\_documents.
8. Configured vector store as a retriever (retriever = vectorstore.as\_retriever()).
9. Language Model Setup: Integrated Ollama language model (llm = Ollama(model="mistral")).
10. Implemented a prompt template (ChatPromptTemplate) for generating responses based on user queries.

* Different Approach to use RAG :

1. Command-Line Interface Approach:   
   File: **rag\_llm\_cli\_based\_qna.py**   
   Description: CLI-based Q&A using RAG directly on the command line.
2. Flask with Postman Approach :   
   File: **flask-rag-chatbot.py**   
   Description: Q&A via POST requests using Flask and Postman for easy interaction with PDF files and questions.
3. Flask with PostgreSQL Approach :  
   Files: **pdf\_insert\_database.py**, **flask-rag-postgresql.py**, **postgresql\_query.txt**  
   Description:   
   Created a PostgreSQL database named ragpdf using pgAdmin.  
   Defined table structure for storing PDF files using SQL queries in postgresql\_query.txt.  
   Implemented functionality to insert PDF files into the PostgreSQL database in pdf\_insert\_database.py.  
   Run flask-rag-postgresql.py for performing Q&A operations.

* **Optional (**Streamlit Web-Based Approaches):

1. File: (1) Streamlit for simple file upload Q&A.  
   Description: Web interface for uploading files and performing Q&A.  
   python file : **streamlit-rag-chatbot.py**
2. File: (2) Streamlit with YouTube integration.  
   Description: Web interface allowing Q&A based on YouTube video content translated to text.  
   python file : **rag\_youtubevideolink\_steamlit.py**
3. GROQ implementation in RAG on cli based QNA :

Description : groq api based llm rag implementation.  
Python file : **groq\_rag\_chatbot\_cli.py**

* Project Enhancement Points :

1. Scalability: Consider deploying the application on scalable cloud infrastructure
2. Accuracy : Evaluate different pre-trained models or fine-tuning techniques to enhance performance like we can used : OPENAI API, other hugging face models.
3. User Interface : Use Streamlit (optional project) implementation
4. Security : encrypt data transmission between clients and server.
5. Logging and Monitoring : Implement logging of important events, errors, and user interactions.
6. Inference Time optimization using GROQ api as implemented in optional code.