**Report on Github Repository Search and Chat System**

**Methodology**

1. Data Acquisition: Clone GitHub repositories using clone\_repo function.
2. Document Extraction and Loading: Traverse and load various file types from the repository using load\_and\_index\_files.
3. Document Splitting and Embedding: Split documents into chunks with RecursiveCharacterTextSplitter.
4. Generate embeddings using SentenceTransformer ('all-MiniLM-L6-v2') and store them in Chroma.
5. Query Processing and Retrieval: Convert user queries to embeddings. Retrieve relevant document chunks from Chroma using similarity search.
6. Response Generation: Create a detailed prompt with conversation history, query, and documents. Generate responses using the Ollama model (Llama2).

**Dataset Source**

The dataset comprises various files from GitHub repositories, including code files, readme files, and notebooks. These files are cloned directly from specified repositories, enabling the processing and indexing of diverse document types for effective information retrieval.

**Retrieval-Augmented Generation (RAG) Technique**

A two step RAG technique is leveraged. The first step involved document retrieval after creating embeddings using SentenceTransformer and creating a vector database store. In the second step the retrieved documents are p rovided to Ollama model using Llama2 – 7B LLM. Prompt is especially curated for this purpose. Langchain was utilized for document loading , text splitting and RAG pipeline.

Why?

1. Using Langchain simplifies the integration of several components and allow easy customization of pipeline
2. Using Ollama for LLM allows to run Llama without GPU and produces contextually accurate answer
3. Sentence Transformer provide high quality embeddings
4. RecursiveCharacterTextSplitter helps to split documents into mangeable chunks.
5. Curating a prompt for the task utilizing several techniques improves the quality of answers.

**Vector Database**

Chroma has been chosen as our vector database as it handles the embeddings in a efficient manner providing semantic searching. It can quickly process and retrieve documents from large datasets thereby increasing the accuracy of our application.

**Preventing Model Hallucinations**

Following steps has been taken to prevent hallucinations:

1. Model generates answer strictly only on the basis of the retrieved document and handles the case when it doesn’t have adequate context. The prompt has been curated to handle this
2. A detailed context is provided including the history and relevant documents and metadata so as to limit the hallucinations
3. Prompt is curated to instruct model to work in step by step manner providing reasoning to the context provided.
4. When asked about a particular file, the presence of that file in context is ensured by matching query with metadata.

**Automated Checking for Response Accuracy**

Semantic similarity score can be used between the retrieved document and model response to check whether the generated response is relevant to the context or not. Keyword matching can be done by identifying key entities in the answer and retrieved document.



