**Project Documentation**

**Overview**

This project leverages FAISS (Facebook AI Similarity Search) and LangChain agents to create a task similarity and solution suggestion system. The application is designed to process and compare tasks, offering solutions based on semantic understanding and search capabilities.

**Datasets**

1. **Real-World Tasks (tasks.csv):** This dataset contains examples of tasks collected from the internet, offering a variety of real-world scenarios.
2. **Generated Tasks (tasks\_generated.csv)**: Generated using **generate\_dataset.py**, this dataset includes a wide range of synthetic tasks, mimicking common IT support issues. The synthetic nature of this dataset means many tasks might be very similar..

**Opening the Project**

The project is structured as a FastAPI application and can be accessed via a web server running locally. Please download the project from this **git** and open it.

**Installation and Setup**

1. **Requirements**: Install dependencies using requirements.txt. ***pip install -r requirements.txt***
2. **API Keys:** The project requires API keys for OpenAI, SerpAPI, and Tavily. These should be added to the **.env** file.

**Data Preparation (prepare\_data.py)**

* **Purpose:** Preprocesses the tasks.csv dataset by normalizing text (lowercasing, punctuation removal) and handling missing values.
* **Usage**: Run prepare\_data.py to generate preprocessed\_data.csv. You can modify this line **data = pd.read\_csv('tasks.csv')** with **data = pd.read\_csv('tasks\_generated.csv')** if you want to use the generated dataset.

**Generating Embeddings (embeddings\_generator.py)**

* **Purpose:** Generates vector embeddings from task descriptions using the SentenceTransformer model.
* **Usage**: Use the script after preprocessing the data to create embeddings for similarity searching. Run embeddings\_generatory.py

**Index Setup (index\_setup.py)**

* **Purpose:** Creates a FAISS index using the generated embeddings, enabling efficient similarity searches.
* **Usage:** Run after generating embeddings to set up the searchable index.

**Running the Project**

1. **Start the FastAPI App:** Run this command line in the terminal **uvicorn app:app --reload**
2. **Access Endpoints:** The app offers several endpoints for task similarity searches and solution generation.

**Accessing Endpoints**

1. **Using Postman for API Testing:**

* With the server running, you can test various endpoints using Postman by sending HTTP requests.

1. Similarity Search Endpoint (/search/):

* URL: <http://127.0.0.1:8000/search/>
* Method: POST
* Body: JSON containing the **task** key and the description of the task you want to search.
* Response: A list of similar tasks based on the input.

1. SelfAskWithSearch Agent with SerpAPI (/serpapi/):

* URL: http://127.0.0.1:8000/serpapi/
* Method: POST
* Body: JSON with the **task** key and the description of the task to be solved.
* Response: A solution for the given task.

1. Structured ReAct Agent with Tavily.ai Search (/tavily):

* URL: http://127.0.0.1:8000/tavily
* Method: POST
* Body: JSON with the **task2** key and the description of the task to be solved.
* Response: A solution for the given task.

1. **Using Python Scripts:**

* While the FastAPI app is running, you can test the endpoints locally by executing the Python scripts **test\_similarity.py**, **test\_solutionAsk.py**, and **test\_solutionReAct.py**.
* Modify the **task, task2** from the data JSON in these scripts to test different scenarios.
* Running these scripts will send requests to the FastAPI app and display the responses, enabling you to verify the functionality of the API endpoints.

**Testing Agents**

* **Without Verbose Output**: Use postman or run test\_solutionAsk.py and test\_solutionReAct.py for testing **without detailed agent steps** and **reasoning**.
* **With Verbose Output**: For a detailed view of the agent's reasoning process, run **ReAct.py** and **SelfSelfAskWithSearch.py** and replace the values of task and task2 to test different situations.

**Understanding app.py**

Central to the application, app.py sets up the FastAPI server, endpoints, and integrates FAISS and LangChain agents for task processing.

**Choice of Tools**

1. **SentenceTransformer**: Selected for generating embeddings due its ability to be used for similarity search operations, also because it is open source and free. The model is well suited for similarity search tasks, as it’s designed to map sentences and paragraphs to a dense vector space.When employed in a similarity search context, this model can effectively generate embeddings for textual data, such that semantically similar items are mapped to proximate points in the vector space, which is integral for identifying similar IT support tasks based on their textual descriptions.
2. **FAISS:** I utilized FAISS (Facebook AI Similarity Search) for similarity search in my scenario as it's specifically designed for efficiently handling such tasks. Given a dataset of IT support tasks, I needed a tool to swiftly find tasks similar to a user-input task. FAISS shines in this aspect as it allows indexing of the high-dimensional vectors generated from the text of tasks using SentenceTransformer('paraphrase-MiniLM-L6-v2'), and subsequently performs rapid similarity searches within this indexed space to find the most analogous tasks. Its ability to handle dense vectors and provide scalable similarity search functions, even in large datasets, made it a fitting choice for identifying similar tasks based on semantic embeddings, enhancing the search functionality in my setup. Additionally, FAISS is open source and free, which aligns with budget-friendly and collaborative development environments, allowing for a cost-effective solution to my similarity search needs.
3. **LangChain SelfAskWithSearch Agent and SerpAPI:**. Through its implementation and execution, it demonstrates a practical way to address complex questions or tasks, which could be beneficial in IT support tasks, where complex queries might need to be broken down and addressed in a step-by-step manner.SerpAPI is noted for its real-time API that facilitates access to search results from various search engines.
4. **LangChain Structured ReAct Agent and Tavily AI:** The ReAct agent in LangChain provides a structured way to handle complex queries by breaking them down. This agent's ability to blend reasoning, action and information retrieval while synthesizing the acquired data to answer the original query makes it an ideal solution for this project's objectives,.SerpAPI is noted for its real-time API that facilitates access to search results from various search engines. In light of the recent integration of Tavily Search with LangChain the Tavily Search is an exceptional fit because it offers a robust search API that is purpose-built for LLM Agents. Tavily Search seamlessly integrates with diverse data sources. This integration capability is valuable when the project demands access to a wide range of information, as it ensures that the ReAct Agent can retrieve data efficiently and effectively.The ReAct Agent can rely on Tavily Search to find the most pertinent information swiftly, enabling it to make informed decisions and take appropriate actions.

**Conclusion**

This project aimed to highlight the availability and effectiveness of open-source models and tools like Sentence Transformers for generating embeddings and FAISS for indexing and similarity search, as cost-effective alternatives to paid tools. By leveraging these open-source resources, the project sought to enhance task similarity search within large datasets and improve task solution generation through LangChain agents, without incurring the additional costs associated with utilizing paid tools like OpenAI for embeddings or vector databases like Pinecone.

The findings suggest that employing open-source models for embeddings and indexing is highly beneficial for conducting similarity searches within extensive task datasets. This approach yielded impressively relevant results when assessing task similarity. Although FAISS is not a vector database itself, it can be utilized to index a set of vectors and, using another vector (the query vector), search for the most similar vectors within the index, thereby enabling efficient similarity searches within a given set of vectors.

Furthermore, the project delved into the efficiency of LangChain agents, with a particular focus on the output results of integrating the ReAct agent with Tavily. This combination proved exceptionally reliable, offering detailed and clear solutions to the input tasks. Tavily's design, specifically for use with LLm agents and its access to real-time internet data from various sources, significantly enhances the ReAct agent's ability to reason and take action effectively.

The potential for scaling this project is evident. Expanding the range of embeddings and indexing tools to include paid options like OpenAI embeddings and a vector database like Pinecone could potentially lead to even better results, albeit at a higher cost. Moreover, the adaptability of the ReAct agent allows for the integration of various tools, ensuring that the most suitable resources are utilized for each task. This flexibility indicates that the ReAct agent is not only efficient in its current configuration but also prepared to evolve and incorporate new advancements in the field, making it a viable solution for solving tasks