Hemant Goyal
Naptick AI Challenge Submission - Task 1
Multi-Collection RAG System with Memory Layer
24/05/25

1. Introduction

This document outlines the implementation of a multi-collection Retrieval Augmented Generation (RAG) system designed as a sleep and wellness AI assistant. The system, "Naptick AI," processes queries by retrieving relevant information from five distinct data collections (wearable data, chat history, user profile, location data, and a custom knowledge base) and uses a Large Language Model (LLM) to generate contextualized answers. It also incorporates a memory layer to maintain conversation context. The project is developed as a Google Colab notebook and features a Gradio web interface for interaction.

The core architecture focuses on generalizing all data sources, treating them as a unified knowledge base without user-specific filtering, simplifying the retrieval for the current single-persona ("Alice") setup.

2. Project Setup and Prerequisites

2.1. Environment

The project is designed to run in a **Google Colaboratory (Colab)** environment, preferably with a GPU runtime (e.g., T4) for efficient model loading and inference.

2.2. Required Accounts and Tokens

- **Google Account:** For accessing Google Colab and Google Drive.
- **Hugging Face Account:** To download the pre-trained language model (Mistral-7B-Instruct-v0.2) and embedding model (all-MiniLM-L6-v2).
- **Hugging Face User Access Token:** This token must be stored as a Colab Secret named HF_TOKEN for authenticated access to Hugging Face Hub.

2.3. Google Drive Structure

The code will create the following directory structure in your Google Drive:

/content/drive/MyDrive/Naptick_Challenge/

3. How to Run the code

1. **Open in Google Colab:** Upload or open the .py file in Google Colab.
2. **Set Hugging Face Token:**
3. Open the Notebook:
4. Ensure a T4 GPU is selected: In Colab, go to Runtime -> Change runtime type -> Hardware accelerator -> T4 GPU.
5. The notebook requires access to the mistralai/Mistral-7B-Instruct-v0.2 model, which is gated.
6. You must first visit https://huggingface.co/mistralai/Mistral-7B-Instruct-v0.2 and accept the terms to gain access with your Hugging Face account.
7. The notebook uses Colab Secrets to store the Hugging Face token. If you are running your own copy of the notebook, you will need to:
8. Create a Hugging Face access token with read permissions at https://huggingface.co/settings/tokens.
9. In your copy of the Colab notebook, click the "Key" icon (🔑) in the left sidebar, click "Add a new secret", name it "HF_TOKEN", and paste your token value. Ensure "Notebook access" is toggled ON.
10. **Select Runtime:**
    - Go to "Runtime" -> "Change runtime type."
    - Select a **GPU** accelerator (e.g., "T4").
11. **Interact with Gradio:**
    - Once Step 10 is executed, a Gradio interface will launch. Click on the local URL (usually ends with gradio.live if share=True or a localhost URL) provided in the output to open the chatbot UI in a new browser tab.
    - Ask questions using the input textbox. Examples are provided in the Gradio interface.

5. Key Technologies and Components

- **Programming Language:** Python
- **Environment:** Google Colaboratory
- **Core Framework:** LangChain
- **LLM:** mistralai/Mistral-7B-Instruct-v0.2 (via Hugging Face Transformers)

- **Embedding Model:** sentence-transformers/all-MiniLM-L6-v2
- **Vector Store:** FAISS (Facebook AI Similarity Search)
- **UI:** Gradio
- **Key Libraries:** transformers, torch, pandas, numpy, bitsandbytes, accelerate.

6. Observations and Potential Future Enhancements

- **Generalized Retrieval:** The current system treats all data as general knowledge. For a multi-user scenario, the original user-specific filtering logic in retrieve_context would need to be re-instated and refined.
- **Context Window:** The Mistral-7B model has a context window. Very long conversations or extremely large retrieved contexts might exceed this, leading to truncation or loss of older information.
- **Retrieval Tuning:** The k_per_store values and the page_content formatting are crucial for effective retrieval. Further experimentation could optimize these. For instance, using different k values for different query types or collections might be beneficial.
- **Advanced Chunking:** More sophisticated chunking strategies (e.g., semantic chunking) could be explored.
- **Re-ranking:** A re-ranking step after initial retrieval (e.g., using a cross-encoder) could improve the relevance of the final documents sent to the LLM.
- **Error Handling:** While robust error handling is in place, specific edge cases in data or queries might require further refinement.
- **Scalability:** For a production system with many users and larger datasets, more scalable vector database solutions (e.g., Pinecone, Weaviate, managed FAISS) would be considered.