## Short Report: Implementation Choices & Challenges

### 1. Project Overview

The **LLM-Powered Booking Analytics & QA System** is designed to process and analyze hotel booking data, providing insights through analytical reports and answering user queries in a retrieval-augmented question-answering (RAG) format. The project leverages **FastAPI** for building the API, **FAISS** for vector storage and similarity search, **Sentence-Transformers** for embedding generation, and **DistilGPT2** for natural language processing (NLP) tasks.

### 2. Key Implementation Choices

#### **a. Data Preprocessing**

We utilized the **Pandas** library to handle the raw hotel booking data (stored in CSV format), where data cleaning and preprocessing tasks were performed, such as:

* **Handling missing values**: Ensuring that any null values were appropriately managed.
* **Converting date columns**: Transforming arrival dates into datetime format for time-based analysis.
* **Optimizing data types**: We employed memory-efficient data types (e.g., int16, float32) for large datasets to minimize memory usage.

#### **b. Analytical Insights**

The following analytics were implemented to provide valuable insights into the data:

* **Revenue trends over time**: By grouping data by month, we calculated total revenue for each month using **Pandas’ groupby** functionality.
* **Cancellation rate**: The cancellation rate is computed as the percentage of canceled bookings relative to the total number of bookings.
* **Geographical distribution**: We identified the top 5 countries contributing the most bookings using **Pandas’ value\_counts()**.
* **Booking lead time distribution**: This was calculated by analyzing the distribution of the lead\_time field.

#### **c. Retrieval-Augmented Question Answering (RAG)**

The key feature of this system is the ability to answer user queries about the booking data. This was achieved using a **vector search** approach:

* **FAISS (Facebook AI Similarity Search)** was chosen to store and search the vector embeddings of booking records efficiently.
* **Sentence-Transformers** were used to generate vector embeddings of booking records, which allows the system to understand the semantic similarity between user queries and data records.
* The query is encoded into a vector, and **FAISS** is used to retrieve the most relevant records. These records are then passed to **DistilGPT2** (a lightweight variant of GPT-2) for generating human-readable answers.

#### **d. API Development**

A **FastAPI** RESTful API was built to:

* Provide booking analytics via a /analytics endpoint.
* Answer user queries via a /ask endpoint.
* Perform health checks with a /health endpoint to monitor the system’s status.

This choice was based on **FastAPI’s** speed, asynchronous support, and built-in validation with **Pydantic** models. The API is also designed to be extensible, allowing for future features like real-time updates or additional analytics.

### 3. Challenges Faced

#### **a. Large File Handling**

One significant challenge was dealing with large files (e.g., embeddings.npy and faiss\_index.bin, each ~175MB). These files were too large for **GitHub’s** regular file storage limit of 100MB. The solution was to use **Git LFS (Large File Storage)** to manage these files. However, this required setting up **Git LFS** and carefully tracking the large files to avoid errors during commits and pushes.

#### **b. Vector Embeddings and Model Integration**

Generating and using vector embeddings for similarity search is an advanced task. Initially, the integration of **Sentence-Transformers** and **FAISS** was complex, particularly when ensuring that the embeddings were stored correctly and retrieved efficiently. Additionally, using a pre-trained language model (DistilGPT2) for answering queries posed some challenges, especially around creating coherent prompts and generating meaningful answers. Balancing the trade-off between the quality of the answers and the computational resources was a key consideration.

#### **c. Handling User Queries**

While the integration of FAISS for retrieving similar records worked well, crafting queries that provide accurate and relevant answers required tuning. For instance, ensuring that the retrieved context (top-k records) was highly relevant to the question asked and that **DistilGPT2** generated accurate answers was a continuous challenge. It required iterative improvements in prompt engineering and context retrieval.

#### **d. Performance Optimization**

One of the major concerns was optimizing the response time of the API, especially when dealing with large datasets. **FastAPI** provided excellent asynchronous capabilities, but we also had to optimize **FAISS** searches and embedding generation. Caching frequently asked queries and improving the way embeddings were stored in memory helped mitigate performance issues. The middleware added for measuring processing time was also a critical part of understanding and optimizing the system’s performance.

### 4. Future Enhancements

* **Real-time Data Updates**: Integrating a **PostgreSQL** or **SQLite** database to allow for real-time updates and querying would improve the system’s flexibility and scalability.
* **Enhanced Analytics**: Expanding the range of analytical insights, such as identifying trends in booking patterns, pricing analysis, or sentiment analysis based on reviews.
* **Advanced NLP Models**: Incorporating more powerful language models such as **Llama 2** or **GPT-Neo** could improve the quality of answers provided to users.

### 5. Conclusion

The implementation of the **LLM-Powered Booking Analytics & QA System** was an exciting and challenging task that involved a variety of technologies to provide users with real-time analytics and advanced question-answering capabilities. The use of vector embeddings, FAISS for similarity search, and **DistilGPT2** for NLP allows the system to efficiently handle large datasets while providing users with valuable insights in a conversational format. Despite challenges around large file management, query handling, and system optimization, the project successfully demonstrated the power of retrieval-augmented question answering (RAG) and the potential of integrating machine learning into business analytics systems.