Build a Multimodal RAG System using AWS Bedrock and FAISS Project Overview

Overview
The restaurant aggregator industry is becoming increasingly reliant on sophisticated technology solutions to stay competitive in the fast-paced food delivery market. In this context, providing users with highly personalized and contextually relevant food recommendations is crucial for enhancing user experience and fostering brand loyalty. Traditional recommendation systems, predominantly based on textual data, often fail to capture the comprehensive and nuanced preferences of users, especially when it comes to visual appeal and presentation of dishes. A multimodal approach that incorporates both text and images can vastly improve the accuracy and personalization of recommendations.

This project aims to design and implement a Multimodal Retrieval-Augmented Generation (RAG) system for a restaurant aggregator app, enhancing its capability to deliver precise food recommendations tailored to individual preferences and dietary needs. The system will integrate and process multiple data types—textual descriptions and visual content—from restaurants to generate personalized suggestions. Utilizing technologies such as Amazon S3 for data storage, Amazon Bedrock for image summarization, and FAISS for efficient similarity search, the system will encode and retrieve vectorized data representations. A user interface powered by Streamlit will facilitate interactive and user-friendly querying, ultimately leading to dynamic and context-aware recommendation generation.

Prerequisite Project: Kindly ensure the completion of the LLM Project for building and fine-tuning a large language model before proceeding with this project.

Prerequisites: Basics of LLMs, Vector Databases, Prompt Engineering, Python and Streamlit

Note: Utilizing AWS services for this project may result in charges; it is essential to thoroughly review the AWS documentation to understand the pricing structure and potential costs associated with different resources and usage patterns.

Aim

Develop a multimodal Retrieval-Augmented Generation (RAG) system to deliver personalized and context-aware food recommendations for a restaurant aggregator app, utilizing both textual and visual data.

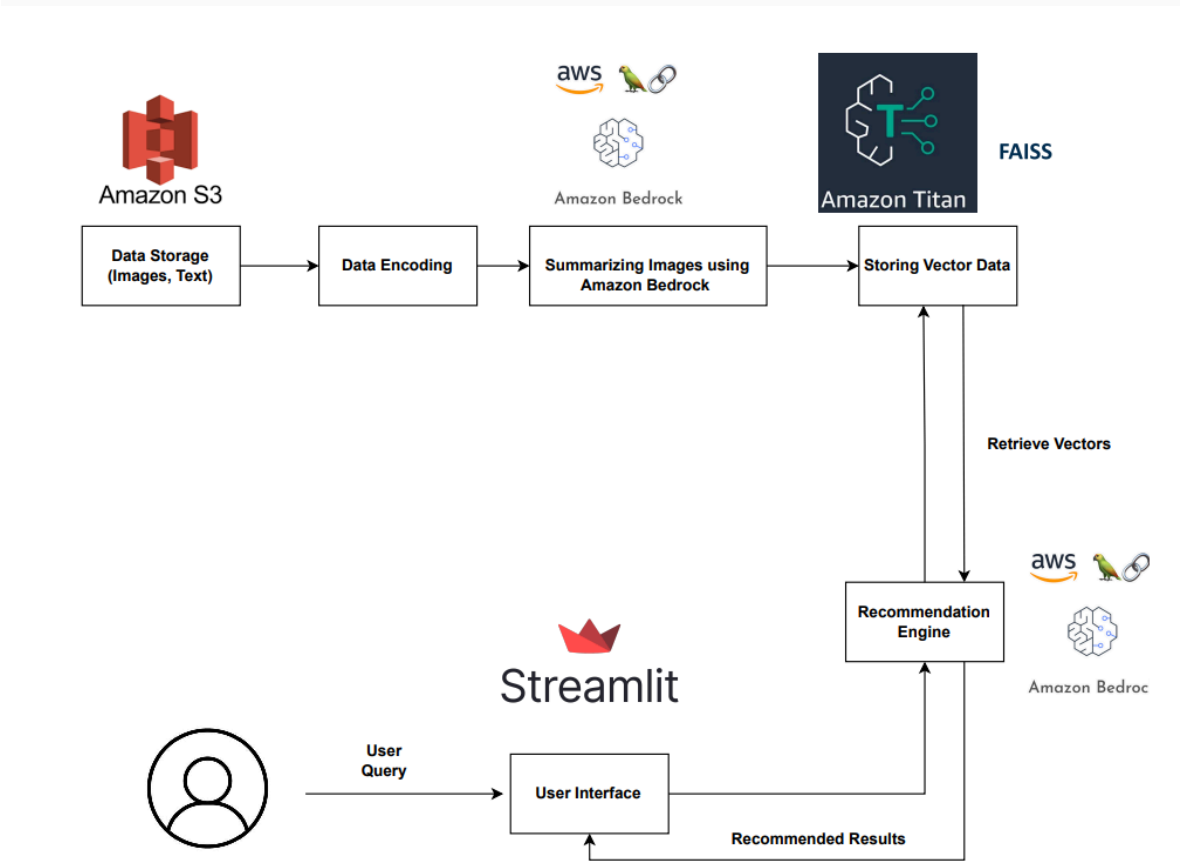
Data Description

The dataset comprises a CSV file containing detailed information on menu items from various restaurants, including identifiers, names, cuisine types, nutritional values, dietary warnings, vegetarian status, image paths, ratings, serves, and prices. Additionally, there is a corresponding folder of images for each menu item, enhancing the multimodal aspect of the data for analysis and recommendation system training.

Tech Stack

- ➔ Language: Python 3.10.4
- ➔ Libraries: boto3, awscli, pandas, langchain, langchain-community, faiss-cpu, streamlit, streamlit-chat
- ➔ Model: Claude-Sonnet Multimodal Model, AWS Titan Embeddings from AWS Bedrock
- ➔ Cloud Platform: Amazon Web Services (AWS)

Architecture



Approach

Data Reading and Preprocessing:

- **Data Storage:** Utilize Amazon S3 for scalable and reliable object storage of the collected data and read the images and metadata.

Data Processing:

- **Image Data:** Use Anthropic Claude-Sonnet model to generate image descriptions

Vector Database:

- **Storage and Retrieval:** Use Amazon Titan Embeddings and FAISS for storing and efficiently retrieving vectorized data, enabling fast and scalable search capabilities for the recommendation engine.

Recommendation Engine:

- **Utilizing Anthropic Claude Sonnet Multimodal Model:** To create conversational chatbot and generate recommendations from Vector DB results.

Streamlit API Development and Integration:

- **User Interface:** Develop an interactive user interface using Streamlit, which will serve the frontend for users to interact with the system.
- **Functionalities:** Users will be able to interact with the chatbot to search text or image inputs.

Modular code overview:

Once you unzip the modular_code.zip file, you can find the following:

```
├─ app.py
├─ data
│  ├─ images/
│  ├─ menu_descriptions_data.csv
│  └─ restaurants_menu_data.csv
├─ Multimodal RAG Presentation.pdf
├─ multimodal-llm.ipynb
├─ output
│  └─ faiss_index
├─ readme.md
├─ reference-images/
├─ requirements.txt
└─ utils.py
```

Here is a brief information on the files:

- **`app.py` and `utils.py`**: `app.py` is the main file that runs the Streamlit-based food recommendation assistant, handling the user interface, input processing, and generating recommendations. `utils.py` contains helper functions and utilities that support the main application, such as data processing and image encoding.

- **`data/` and `output/`**: The `data/` folder stores essential datasets like `menu_descriptions_data.csv` and `restaurants_menu_data.csv`, which

contain information about menu items and their descriptions. The `images/` subfolder holds associated images. The `output/` folder contains the `faiss_index`, a pre-built index used for efficient similarity search, essential for quickly retrieving relevant data during recommendations.

- **`Multimodal RAG Presentation.pdf` and `multimodal-llm.ipynb`**: The `Multimodal RAG Presentation.pdf` is a presentation that likely explains the concepts, methodologies, and findings related to the Multimodal Retrieval-Augmented Generation (RAG) used in the project. `multimodal-llm.ipynb` is a Jupyter notebook that may contain code, experiments, or demonstrations related to the development and testing of the multimodal LLM application.

- **`readme.md`, `requirements.txt`, and `reference-images/`**: `readme.md` provides instructions, project overview, and setup guidelines, while `requirements.txt` lists all necessary dependencies to run the application. The `reference-images/` folder contains additional images that may be used as references within the project, possibly for testing or as part of the dataset.
  Kindly follow all the instructions for running the code from Readme.md file.

Project Takeaways

1. Learn the fundamentals of multimodal data (text, images) for LLM applications
2. Gain insights into how Amazon S3, Amazon Bedrock, and FAISS can be utilized in recommendation system applications
3. Learn how to use S3 for efficient, scalable, and secure data storage.
4. Understand how to preprocess text and image data for large language model applications
5. Explore the capabilities of AWS Titan for transforming text data into embeddings
6. Understand the functionality of vector databases in search and retrieval systems
7. Learn how to use FAISS for efficient storage and retrieval of high-dimensional data vectors
8. Learn about prompt engineering techniques and result optimization techniques for RAG applications
9. Learn the steps to develop a recommendation system that can suggest items based on user preferences using a multimodal LLM

10. Develop skills in using Langchain and Streamlit to create interactive, user-friendly chatbots.