Fashion AI - Generation Search System

By
Rajiv Gaba

Rajiv.eml69@iiitb.net

For submission of assignment at

Indian Institute of Information Technology, Bangalore

On

1st Oct 2025

Table of Contents

Project Objectives... 3
System Design and Architecture... 3
Overall System Design (RAG Pipeline)... 3
1. Data Ingestion Phase (Offline):..3
2. Query Execution Phase (Online):... 3
Project Layers (Layered Architecture)...4
Implementation Details.. 4
Core Technologies...4
Implementation Flow..4
Challenges & Lessons Learnt... 5
Challenges Encountered.. 5
Lessons Learnt.. 5
Way Forward... 5
Annexure.. 6

Project Objectives

The primary objective of the **Fashion Search AI** project was to develop a **generative search system** capable of performing semantic search over a large collection of product descriptions.

The core goals were to:

- **Create a Generative Search System:** Build a system that can accurately search a plethora of product descriptions.
- **Recommend Appropriate Choices:** Based on a natural language user query, the system must find and recommend appropriate fashion products.
- **Implement Retrieval-Augmented Generation (RAG):** Design and implement an RAG application to efficiently retrieve fashion products from a Kaggle dataset.

System Design and Architecture

The **Fashion Search AI** system is architected as a **Retrieval-Augmented Generation (RAG)** pipeline, utilizing modular components orchestrated by the **LangChain** framework.

Overall System Design (RAG Pipeline)

The system operates in two main phases: **Data Ingestion** (Offline) and **Query Execution** (Online).

1. Data Ingestion Phase (Offline):
    - The raw fashion product dataset (sourced from Kaggle) is loaded.
    - Product descriptions are processed and split into smaller, manageable chunks (documents).
    - The documents are transformed into numerical **vector embeddings** using a pre-trained embedding model (**all-MiniLM-L6-v2**).
    - These embeddings are stored in a **FAISS** (Facebook AI Similarity Search) index, creating an efficient vector store.
2. Query Execution Phase (Online):
    - A user submits a query via the Gradio interface.
    - The user's query is converted into an embedding using the same model used during ingestion.
    - The query embedding is used to perform a similarity search against the **FAISS** vector store, retrieving the **Top K** (e.g., 3) most relevant product descriptions (documents).
    - The retrieved documents are passed, along with the original user query, to a **Local Large Language Model (LLM)**.
    - The LLM generates a descriptive response and extracts the file paths for the top recommended product images.
    - The final response (text and images) is displayed to the user.

Project Layers (Layered Architecture)

The system is logically divided into four distinct layers:

Layer	Primary Function	Key Components
Data Layer	Source, storage, and initial processing of product information.	Kaggle Product Dataset, Image Files, Pandas
Retrieval/Vector Layer	Efficiently store and retrieve semantic information from the data.	FAISS Vector Store, Sentence Transformers (all-MiniLM-L6-v2)
Generative Layer	Generate the final, contextualized output using the retrieved data.	LangChain (Orchestrator), Perplexity Sonar-pro
Presentation Layer	Handle user input and display the system's output.	Gradio Interface

Implementation Details

Core Technologies

The project relies on a stack of popular open-source technologies:

- **Orchestration Framework: LangChain** was chosen for its flexibility and modularity in building LLM pipelines.
- **Large Language Model (LLM):** The **Perplexity sonar-pro** model was utilized.
- **Vector Store: FAISS** was selected for its high performance in similarity search.
- **Embedding Model:** The all-MiniLM-L6-v2 model from Sentence Transformers was used to convert text into embeddings.
- **Web Interface: Gradio** was used to rapidly build the user interface.

Implementation Flow

The system's core logic is executed within a function that encapsulates the RAG chain:

1. **Retrieval:** The retriever component fetches the top 3 most relevant product documents.
2. **Prompt Engineering:** A specific prompt template is constructed that contains the user's query and the retrieved product context.
3. **Generation:** The Perplexity model processes the prompt to generate a natural language response and identify the image file paths for the recommended products.

4. **Interface Update:** The function returns the descriptive text and the three image files to the Gradio interface for display.

Challenges & Lessons Learnt

Challenges Encountered

Data-to-Image Mapping: Reliably extracting the correct product image file paths from the LLM's generated response was critical to accurately represent the recommendations in the Gradio interface.

Vector Store Creation: Vector store creation was a challenge considering the data size. Vector store was created in several attempts by making use of Kaggle GPUs and eventually persisted for reuse.

Lessons Learnt

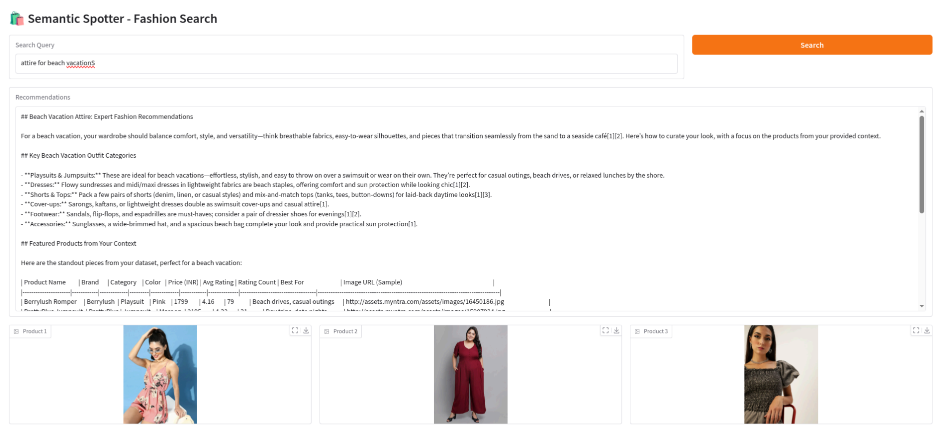
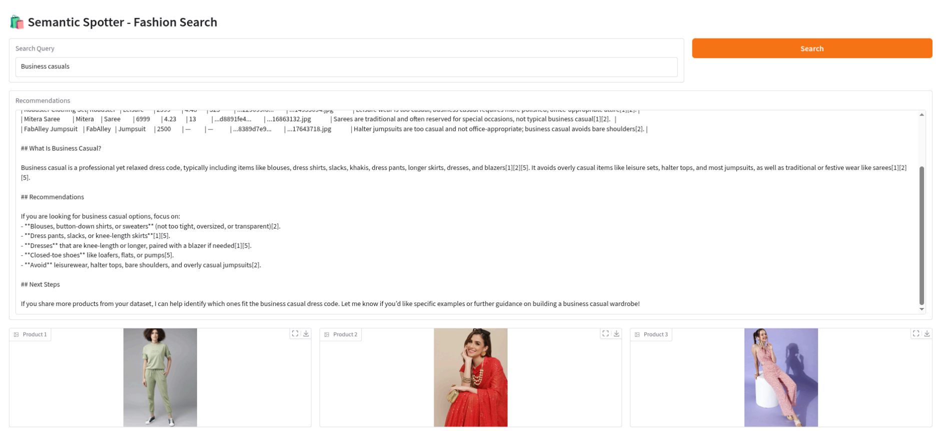
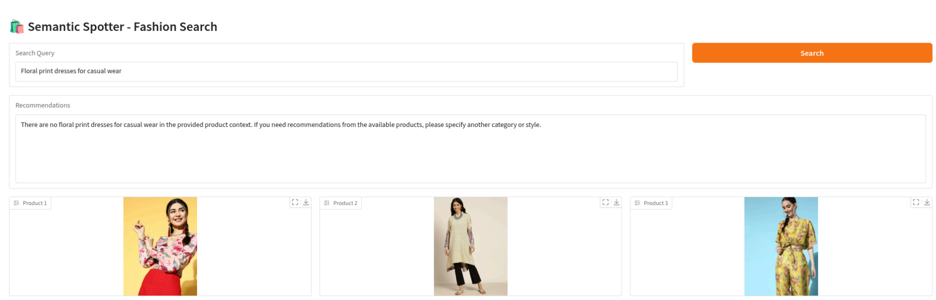
- **Langchain** is a powerful framework that offers modularity, customization and flexibility for chain creation, integration with LLMs. Switching LLMs is extremely easy.
- **RAG** project reaffirmed that RAG is a highly effective pattern for creating search experiences in specific domains (like fashion) where accuracy is paramount.
- **FAISS** similarity search and compression reranking ensures contextual results. It virtually eliminates the need for external cache

Way Forward

Enhancements can be done in future on this project. Next layers can include adding more datasets, optimising the UI, experimentation with evolving embedding models and use of alternative LLMs.

Annexure

Screenshots:

<table>
<tr><th>Query</th><th>Output</th></tr>
<tr><td>Attire for beach vacation</td><td>
</td></tr>
<tr><td>Business casuals</td><td>
</td></tr>
<tr><td>Floral print dresses for casual wear</td><td>
</td></tr>
</table>

Author:
Rajiv Gaba
Email: rajiv.eml69@iiitb.net
Project Artefacts: https://github.com/rajivgaba/semantic-spotter-langchain