1. Introduction

Objective

Fashion Search AI is an intelligent search system that helps users find fashion products based on textual queries. It utilizes **Natural Language Processing (NLP)** and **vector embeddings** to match user queries with product descriptions in a dataset.

Key Features

- AI-powered search for fashion products.
- Uses **OpenAI embeddings** for text similarity.
- Stores and retrieves data using **ChromaDB**.
- Implements **LangChain** to generate structured responses.

---

2. System Design

Architecture Overview

The system consists of the following layers:

- **Data Layer**: Stores product data in structured format.
- **Embedding Layer**: Converts text descriptions into numerical vectors.
- **Storage Layer**: Uses **ChromaDB** for storing and retrieving embeddings.
- **Processing Layer**: Uses **LangChain** for query processing.
- **User Interaction Layer**: Takes user queries and returns relevant fashion product results.

---

3. Implementation

3.1 Data Preparation

We start by loading and processing the dataset.

```
import os
import pandas as pd

# Load dataset
file_path = "fashion_dataset.csv"
```

```
if not os.path.exists(file_path):
    raise FileNotFoundError(f"Dataset not found at {file_path}")

fashion_data = pd.read_csv(file_path)

# Ensure required columns exist
required_columns = ["description", "price", "brand", "colour"]
for col in required_columns:
    if col not in fashion_data.columns:
        raise ValueError(f"Missing required column: {col}")
```

3.2 Text Embeddings

Convert text descriptions into embeddings using **OpenAI's text-embedding-ada-002** model.

```
from langchain.embeddings import OpenAIEmbeddings
from langchain.schema import Document

api_key = os.getenv("OPENAI_API_KEY")
embedding_model = OpenAIEmbeddings(model="text-embedding-ada-002",
openai_api_key=api_key)

# Convert data into LangChain documents
documents = [
    Document(
        page_content=f"{row['description']} Price: {row['price']} INR. Brand: {row['brand']}. Color:
{row['colour']}",
        metadata={"price": row["price"], "brand": row["brand"], "colour": row["colour"]}
    )
    for _, row in fashion_data.iterrows()
]
```

3.3 Storing and Retrieving Data

Store embeddings using **ChromaDB** for efficient retrieval.

```
from langchain.vectorstores import Chroma

# Define ChromaDB storage path
db_path = "./chroma_fashion_db"

if os.path.exists(db_path):
    vectorstore = Chroma(persist_directory=db_path, embedding_function=embedding_model)
else:
```

```
    vectorstore = Chroma.from_documents(documents, embedding_model,
persist_directory=db_path)

vectorstore.persist()
retriever = vectorstore.as_retriever()
```

3.4 Query Processing

Use **LangChain** to process user queries and return relevant fashion products.

```
from langchain.llms import OpenAI
from langchain.chains import LLMChain
from langchain.prompts import PromptTemplate

# Define prompt template
prompt_template = PromptTemplate(
    input_variables=["context", "query"],
    template="""
    You are a fashion AI assistant. Use the following product database to find the best match:

    {context}

    User Query: {query}

    Provide a detailed response with product suggestions.
    """
)

# Set up LLM and chain
llm = OpenAI( openai_api_key=api_key)
llm_chain = LLMChain(llm=llm, prompt=prompt_template)
```

3.5 Search Function

A function to take user queries and fetch relevant products.

```
def search_fashion(query):
    relevant_docs = retriever.get_relevant_documents(query)
    context = "\n".join([doc.page_content for doc in relevant_docs])
    response = llm_chain.run(context=context, query=query)
    return response

# Example query
if __name__ == "__main__":
    query = "Find me a red dress under 2000 INR"
```

```
result = search_fashion(query)
print("Search Result:", result)
```

---

4. Lessons Learned

- **Efficient Vector Storage**: ChromaDB proved useful for fast retrieval.
- **Importance of Prompts**: Better prompts led to more accurate recommendations.
- **Dataset Quality Matters**: Clean, well-structured data significantly improved system performance.

---

5. Conclusion

The **Fashion Search AI** system successfully enables users to find fashion products using **Long chain search**. Future improvements may include **fine-tuned LLM models** and **multimodal embeddings** to incorporate image-based searches.