# RAG System for Derma Product Recommendations

## Overview

This project implements a **Retrieval Augmented Generation (RAG) system** to provide **skincare product recommendations** and **general skincare advice**. The system uses **hybrid query routing**, combining **semantic search, keyword filtering, and external web search** to deliver the most relevant results. The pipeline integrates a **vector database (Weaviate), an embedding model (BAAI/bge-large-en), a language model (GPT-4o), and a web search API (Tavily)** to efficiently process and generate responses.

## Key Features

- **Hybrid Query Routing:** Determines whether a query is a **PRODUCT query** (recommendation-based) or a **GENERAL query** (skincare advice).
- **Vector Database Integration:** Uses **Weaviate** for efficient semantic and keyword-based retrieval.
- **Metadata Filtering:** Allows filtering of products based on **price, sale status, and ingredients**.
- **Web Search Integration:** Fetches external data using **Tavily API** for general skincare queries.
- **Conversational Memory:** Maintains chat history to **contextually enhance responses**.
- **User-Friendly UI:** Built using **Streamlit** for an intuitive interface.

---

## System Architecture

### 1. Data Ingestion to Response Pipeline

1. **Data Ingestion:**
    - Load and preprocess the **Cleaned_Product_Data.xlsx** file.
    - Extract relevant columns for **embedding** and **metadata storage**.
    - Convert **product descriptions** into vector embeddings using `BAAI/bge-large-en`.
    - Store **vectors + metadata** in **Weaviate**.
2. **User Query Processing:**
    - Classify the query as **PRODUCT** or **GENERAL**.
    - Extract **price, category, and sale filters** if applicable.
3. **Retrieval Process:**
    - **PRODUCT Queries:** Hybrid search in **Weaviate** (semantic + keyword filtering).
    - **GENERAL Queries:** External search via **Tavily API**.
4. **LLM Processing:**

    - Retrieved documents are passed to **OpenAI GPT-4o**.
    - LLM **contextually ranks & refines results**.
5. **Response Generation:**
    - The chatbot delivers **personalized product recommendations** or **expert skincare advice**.
    - **Chat history is stored** for contextual conversation continuity.

---

# Tech Stack & Justifications

## 1. Why Weaviate Over Other Vector Databases (Chroma, FAISS, Pinecone)?

- **Hybrid Search:** Combines **semantic search** and **keyword-based filtering**, unlike FAISS (which is pure vector-based).
- **Scalability & Cloud Integration:** Weaviate provides **managed cloud solutions**, unlike FAISS or ChromaDB.
- **Structured Metadata Storage:** Allows **filtering based on price, category, and sale status**.
- **Better Query Speed:** Optimized for **real-time retrieval with structured metadata filtering**.

## 2. Why Sentence Transformers (BAAI/bge-large-en) Instead of OpenAI Embeddings?

- **Performance on Benchmarks:** `BAAI/bge-large-en` outperforms OpenAI embeddings on **MTEB benchmarks**.
- **Local Deployment & Privacy:** Can be run **locally** without API calls, unlike OpenAI.
- **Cost-Efficiency:** OpenAI embeddings incur **per-call costs**, while BAAI models run **for free** once downloaded.
- **Customization:** Supports **variable embedding dimensions**, optimizing **storage and compute usage**.

## 3. Why Use OpenAI GPT-4o Instead of Hugging Face Models?

- **High Accuracy & Coherence:** GPT-4o generates **context-aware and coherent responses**.
- **Cloud-Based API:** No need for **local deployment** or fine-tuning.
- **Better Multi-Turn Conversations:** Handles **complex queries** and **maintains conversational memory**.

## 4. Why LangChain Over LangGraph for Query Classification?

- **Simpler OpenAI API Integration:** Pre-built LLM chains simplify query classification.
- **Flexible Query Routing:** Dynamically routes queries between **product search and web search**.
- **Faster Development: LangChain's predefined chains** streamline the RAG setup.

---

# Hybrid Querying Mechanism (Semantic + Keyword-Based Search)

## How It Works

1. **User submits a product-related query** (e.g., "Recommend a moisturizer under 1200 for oily skin").
2. **Query is vectorized** using `BAAI/bge-large-en` embeddings.
3. **Semantic Search in Weaviate** retrieves the top `k` similar products.
4. **Keyword Filtering is applied** (e.g., `price < 1200`, `category == moisturizer`).
5. **Final refined product list** is passed to the LLM for personalized recommendations.

## Why Hybrid Search?

- **Semantic search alone** may retrieve irrelevant products.
- **Keyword-based search alone** lacks contextual understanding.
- **Combining both** ensures accurate, **filter-specific recommendations**.

---

# Query Classification (PRODUCT Queries vs GENERAL Queries)

## Classification Logic

1. **PRODUCT Queries:**
   - Keywords: "recommend," "suggest," "find," "moisturizer," "face wash".
   - Example: "Recommend a sunscreen under 1500 for dry skin".
   - Routed to **Weaviate hybrid search** for recommendations by prompt engineering.
2. **GENERAL Queries:**
   - Keywords: "how to," "benefits of," "does," "what is".
   - Example: "How can I treat acne scars?".
   - Routed to **Tavily API (Web Search)** for skincare insights by prompt engineering.

## How Classification Works

- The **Query Router (LangChain-based)** processes user input.
- The **LLM determines intent** and extracts **filters** (if applicable).
- The query is routed to the **appropriate retrieval system**.

---

# Implementation Details

### Data Processing (`data_processor.py`)

- Loads and **cleans XLSX product data**.
- Converts **product descriptions into embeddings**.
- Prepares structured format for **Weaviate ingestion**.

### Vector Database (`vector_store.py`)

- Connects to **Weaviate** and **stores embeddings**.
- Implements **semantic and hybrid search**.
- Enables **metadata filtering** (e.g., price, category, on-sale status).

### Query Router (`query_router.py`)

- **Classifies queries** as `PRODUCT` or `GENERAL`.
- **Extracts filters** (price, category, etc.) if applicable.

### RAG System (`rag_system.py`)

- **Orchestrates the pipeline**.
- Routes queries to **Weaviate or Tavily API**.
- Implements **Conversational Memory** for chat history.

### Web Search (`web_search.py`)

- Uses **Tavily API** to fetch **real-time skincare knowledge**.
- Provides **external context** to the LLM.

### Frontend UI (`app.py`)

- Built using **Streamlit**.
- Includes options to **initialize, process data, and reset chat**.
- Displays **chat history and responses**.

---

# Conclusion

This RAG-based skincare recommendation system effectively combines:

- **Hybrid Retrieval (Semantic + Keyword)** for accurate product searches.
- **Query Classification (PRODUCT vs GENERAL)** for intelligent query routing.
- **Weaviate's Advanced Search Capabilities** for optimized vector retrieval.
- **Real-Time Web Search Integration** for updated skincare insights.
- **LLM-Powered Conversational Agent** for context-aware recommendations.

By integrating **Weaviate, OpenAI GPT-4o, LangChain, and Tavily API**, the system provides an **efficient, scalable, and intelligent** skincare recommendation experience.