

INFO7375 - Prompt Engineering & AI
Summer 2024



“FashionBot”

Adaptive Recommendation Chatbot
with RAG and Vector Database

By – Kaushikee Bhawsar (002704590)

Master of Science in Information Systems
College of Engineering
Northeastern University, Toronto
Summer 2024

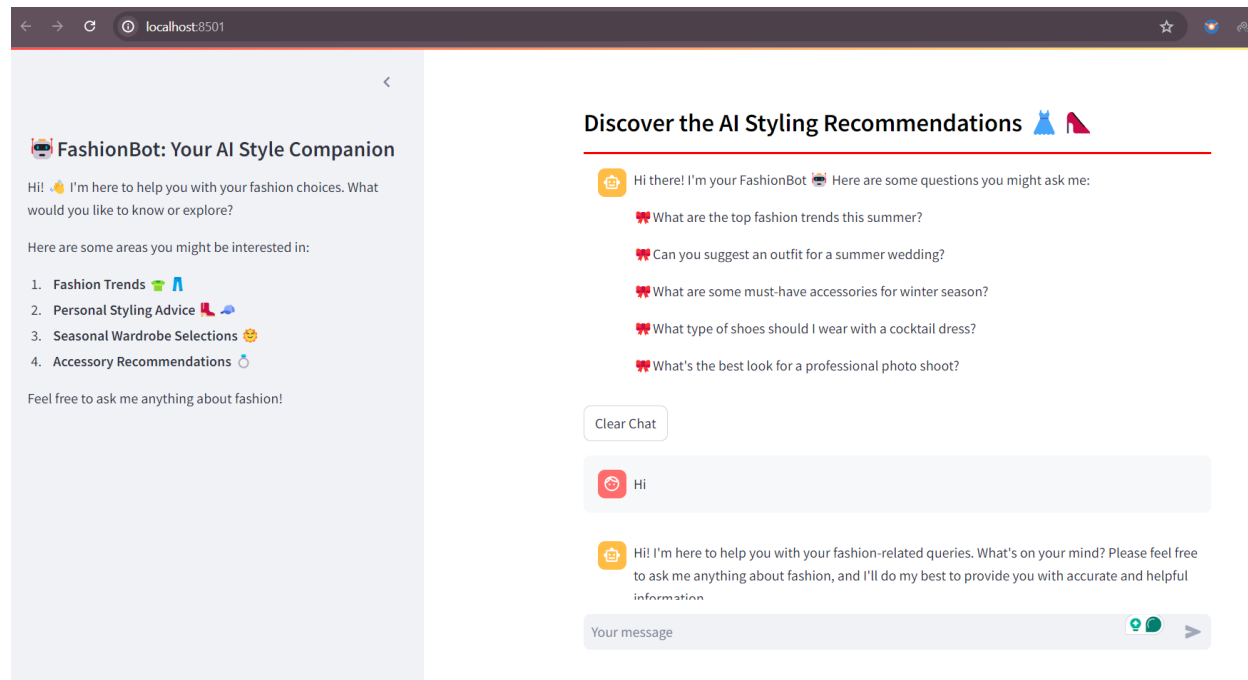
Table of Contents

Table of Contents	2
Introduction	4
Domain selection	4
Scope Definition	4
Type of Data Handled	5
Data Collection and Preprocessing	5
Preprocessing Steps.....	5
How FashionBot works?	6
Data Loading:	6
Text Chunking:	6
Language Model:.....	6
Similarity Matching:.....	6
Response Generation:	7
FashionBot Architecture	7
Vector Database Implementation	7
Importing dependencies and API key	8
PDF Data Loading	8
Text Splitting	8
Text Embedding.....	8
Storing in Vector Database.....	9
Validate the setup	9
Data Indexing and Retrieval.....	9
Application Development.....	10
Initialization of Environment and Dependencies	10
Setup of the Language Model and Vector Store.....	11
Preparation of the Retrieval System:	11
Integration of Custom Prompt Template:.....	11

Response Generation Mechanism:.....	12
Streamlit Interface Configuration:	12
User Interaction Flow:	13
Prompt Engineering for Mock Data	14
Prompt pattern	14
Evaluation and Testing	15
Testing Methodology.....	15
Testing Results	19
LLM Model Performance Comparison.....	19
Challenges and Overcome solutions.....	20
Dependency Installation Issues	20
Language Model Configuration.....	20
Streamlit Application Deployment.....	20
Conclusion	21

Introduction

The **FashionBot** is an adaptive recommendation system provides styling advice and fashion recommendations through a sophisticated chatbot interface. This system integrates several advanced technologies including LangChain, Hugging Face's Transformers, Retrieval-Augmented Generation (RAG), VectorDB implemented via Chroma, and Streamlit. These technologies collectively enable the system to utilize extensive fashion knowledge and linguistic intelligence, offering personalized fashion advice efficiently.



Domain selection

The initial step in the development of my application was to choose a specific domain, which is crucial for tailoring the application's functionality to meet the needs of its intended users. For my project, the selected domain is the fashion industry, specifically focusing on providing styling advice and fashion recommendations.

1. Scope Definition

The scope of my FashionBot app encompasses:

- **Fashion Trends:** Keeping users updated with the latest fashion trends.
- **Personal Styling Advice:** Offering personalized clothing recommendations based on user preferences and current trends.

- **Seasonal Wardrobe Selections:** Assisting users in selecting appropriate attire for different seasons.
- **Accessory Recommendations:** Suggesting complementary accessories to enhance outfits.

2. Type of Data Handled

FashionBot application manages a variety of data types that are crucial for delivering accurate and relevant fashion advice:

- **Clothing Attributes:** Includes type, material, color, size, and style.
- **User Preferences:** Personal tastes and preferences in clothing style and comfort.
- **Seasonal Data:** Information pertinent to fashion choices for different seasons and weather conditions.

Data Collection and Preprocessing

For FashionBot, I sourced a comprehensive dataset of fashion clothing from Kaggle, a popular platform for machine learning datasets. This dataset is crucial for powering FashionBot's recommendation capabilities.

Preprocessing Steps

The preprocessing of the dataset involved several key procedures to optimize the data for indexing in the vector database and processing by the language model (LLM):

1. **Data Cleaning:** I removed any irrelevant information that did not pertain to the specific attributes needed for fashion recommendations, such as unrelated metadata or extraneous text.
2. **Handling Missing Data:** I identified and appropriately handled any missing values, especially in critical fields like clothing type, material, or size. This often involved imputing data based on similar items or removing entries that lacked essential information.
3. **Column Selection:** I selected specific columns that are most relevant for the functioning of FashionBot, which include "Item", "Type", "Style", "Season", "Material", "Color", and "Sizes Available". These columns provide detailed descriptions that are vital for semantic searches in the vector database.
4. **Data Structuring:** Given the conversion from CSV to JSON to PDF, it was essential to maintain a structured format. Each clothing item's attributes were distinctly organized and easily extractable for vector embedding.

```
{
  "Item": "Twilight Velvet Blazer",
  "Type": "Jacket",
  "Style": "Formal",
  "Season": "Fall/Winter",
  "Material": "Velvet",
  "Color": "Deep Navy",
  "Sizes Available": ["S", "M", "L", "XL"],
  "Features": "Tailored fit, satin lapels, single-button closure, with decorative cuff buttons and inner pocket."
}
```

How FashionBot works?

The FashionBot system operates through a sequence of steps to process user queries and provide relevant responses:

1. Data Loading:

Utilizes **LangChain's PyPDFLoader** to read PDF document, extracting their text content. This step forms the foundation for creating a searchable database of fashion-related information.

2. Text Chunking:

The extracted text is divided into smaller chunks using **LangChain's RecursiveCharacterTextSplitter**. This method ensures that the text is manageable and retains sufficient context, which is crucial for producing accurate embeddings.

3. Language Model:

Employs Hugging Face's Transformers, specifically the **sentence-transformers/all-MiniLM-L6-v2** model, to generate vector representations (embeddings) of the text chunks. These embeddings capture the semantic essence of the text, allowing for sophisticated query handling.

4. Similarity Matching:

When a query is received, the system performs a semantic similarity search using the **Chroma** vector database. This search identifies text chunks that are most

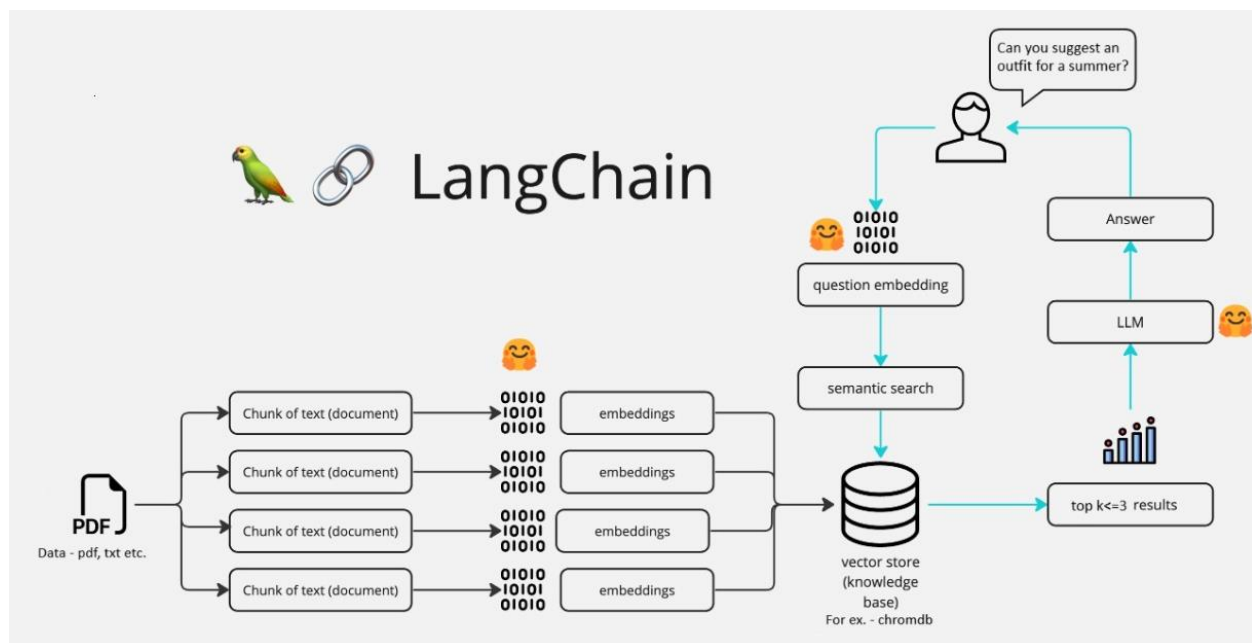
semantically similar to the query, ensuring that the most relevant content is selected for response generation.

5. Response Generation:

The relevant text chunks are passed to the Hugging Face Hub Language Model. Two models were tested:

- meta-llama/Meta-Llama-3-8B-Instruct
- mistralai/Mistral-7B-Instruct-v0.3

FashionBot Architecture



Vector Database Implementation

For FashionBot, I chose Chroma as the vector database due to its compatibility with LangChain and the specific requirements for handling fashion data. This decision supports efficient data retrieval based on semantic similarity, which is crucial for delivering accurate fashion recommendations. Below are the implementation steps:

1. Importing dependencies and API key

```
from langchain.document_loaders import PyPDFLoader
from langchain.text_splitter import RecursiveCharacterTextSplitter
from langchain.embeddings import HuggingFaceEmbeddings
from langchain.vectorstores import Chroma
import os
from dotenv import load_dotenv
from collections import OrderedDict

# Load environment variables from .env file
load_dotenv()

os.environ["HUGGINGFACEHUB_API_TOKEN"] = os.getenv("HUGGINGFACEHUB_API_TOKEN")
```

2. PDF Data Loading

I used the PyPDFLoader from LangChain to load the fashion data PDF. This step ensures that all the textual content needed for the vector database is accessible.

```
# Load the PDF
loader = PyPDFLoader("fashion_data.pdf") # Provide your PDF path here
documents = loader.load()
```

3. Text Splitting

With the RecursiveCharacterTextSplitter, I divided the loaded text into manageable chunks, optimizing them for the embedding process.

```
# Split the text
text_splitter = RecursiveCharacterTextSplitter(chunk_size=2000, chunk_overlap=300)
texts = text_splitter.split_documents(documents)
```

4. Text Embedding

Using HuggingFaceEmbeddings, specifically the sentence-transformers/all-MiniLM-L6-v2 model, I converted the text chunks into vector embeddings. This transformation is key to enabling semantic-based data retrieval.


```
# Initialize the embedding model
embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")

# Convert texts to embeddings
try:
    embeddings = embedding_model.embed_documents([doc.page_content for doc in texts])
    print("Vector Embeddings created successfully")
except Exception as e:
    print(f"Error creating vector embeddings: {e}")
```

5. Storing in Vector Database

The generated embeddings were stored in Chroma, set up with a directory to persist data. This database structures the data to facilitate quick and efficient similarity searches.

```
# Initialize Chroma vector store
vector_store = Chroma(embedding_function=embedding_model, persist_directory="data")

# Add documents to the vector store
vector_store.add_documents(documents=texts)
```

6. Validate the setup

```
# Validate the setup
try:
    # Test query to validate data retrieval
    test_query = "What are some popular items for winter?"
    results = vector_store.search(query=test_query, search_type='similarity')

    # Deduplicate results
    unique_results = OrderedDict()
    for doc in results:
        if doc.page_content not in unique_results:
            unique_results[doc.page_content] = doc

    # Convert unique results to a list and limit to top 3
    final_results = list(unique_results.values())[:3]
    print(f"Unique query results: {final_results}")
except Exception as e:
    print(f"Error during test query: {e}")
```

7. Data Indexing and Retrieval

I configured the database to index data in a way that supports semantic similarity searches. The system fetches the top 3 most similar entries based on a user's query,

using the `vector_store.as_retriever(top_k=3)` setting within the `RetrievalQA` from `LangChain`.

```
rag_chain = RetrievalQA.from_chain_type(
    llm=hf_hub_llm,
    chain_type="stuff",
    retriever=vector_store.as_retriever(top_k=3), # retriever is set to fetch top 3 results
    chain_type_kwargs={"prompt": custom_prompt})
```

Application Development

For FashionBot, I designed an interactive user interface using **Streamlit** that enables users to submit natural language queries about fashion. This interface is intuitive and tailored to facilitate easy engagement with the system, ensuring users can readily seek fashion advice.

Implementation steps:

1. Initialization of Environment and Dependencies

- Import necessary libraries such as Streamlit, LangChain, pysqlite3 and dotenv for environment management.
- Load environment variables like the HuggingFace API token to authenticate and utilize HuggingFace's services.

```
1  import streamlit as st
2  from langchain_huggingface import HuggingFaceEmbeddings
3  from langchain_community.vectorstores import Chroma
4  from langchain_community.llms import HuggingFaceHub
5  from langchain.prompts import PromptTemplate
6  from langchain.chains import RetrievalQA
7  import warnings, os
8  from dotenv import load_dotenv
9  warnings.filterwarnings("ignore")
10 __import__('pysqlite3')
11 import sys
12 sys.modules['sqlite3'] = sys.modules.pop('pysqlite3')
13
14 # Load environment variables from .env file
15 load_dotenv()
16
17 data_directory = os.path.join(os.path.dirname(__file__), "data")
18
19 os.environ["HUGGINGFACEHUB_API_TOKEN"] = os.getenv("HUGGINGFACEHUB_API_TOKEN")
```

2. Setup of the Language Model and Vector Store

- Initialize the Hugging Face Embeddings model (sentence-transformers/all-MiniLM-L6-v2) to process and embed textual data.
- Configure Chroma as the vector database to store these embeddings, which facilitates efficient retrieval based on semantic similarity.

```

21
22 # Load the vector store from disk
23 embedding_model = HuggingFaceEmbeddings(model_name="sentence-transformers/all-MiniLM-L6-v2")
24 vector_store = Chroma(embedding_function=embedding_model, persist_directory=data_directory)
25

```

3. Preparation of the Retrieval System:

- Set up the RetrievalQA system from LangChain, incorporating the Hugging Face Hub Language Model (Meta-Llama-3-8B-Instruct).

```

26 # Initialize the Hugging Face Hub LLM
27 hf_hub_llm = HuggingFaceHub(
28     repo_id="meta-llama/Meta-Llama-3-8B-Instruct",
29     # repo_id="mistralai/Mistral-7B-Instruct-v0.3",
30     model_kwargs={"temperature": 1, "max_new_tokens":1024},
31 )
32

```

- Configure the retriever within this system to fetch the top 3 most relevant entries from the vector database, ensuring responses are precise and relevant.

```

69 rag_chain = RetrievalQA.from_chain_type(
70     llm=hf_hub_llm,
71     chain_type="stuff",
72     retriever=vector_store.as_retriever(top_k=3), # retriever is set to fetch top 3 results
73     chain_type_kwargs={"prompt": custom_prompt})
74

```

4. Integration of Custom Prompt Template:

- Define a custom prompt template based on **Directive-based prompt pattern** which guides the language model on how to generate appropriate responses based on the query context and user interaction directives.

```

33 prompt_template = """
34 As a highly knowledgeable fashion assistant, your role is to accurately interpret fashion queries and
35 provide responses using our specialized fashion database. Follow these directives to ensure optimal user interactions:
36 1. Precision in Answers: Respond solely with information directly relevant to the user's query from our fashion database.
37    Refrain from making assumptions or adding extraneous details.
38 2. Topic Relevance: Limit your expertise to specific fashion-related areas:
39    - Fashion Trends
40    - Personal Styling Advice
41    - Seasonal Wardrobe Selections
42    - Accessory Recommendations
43 3. Handling Off-topic Queries: For questions unrelated to fashion (e.g., general knowledge questions like "Why is the sky blue?"),
44    politely inform the user that the query is outside the chatbot's scope and suggest redirecting to fashion-related inquiries.
45 4. Promoting Fashion Awareness: Craft responses that emphasize good fashion sense, aligning with the latest trends and
46    personalized style recommendations.
47 5. Contextual Accuracy: Ensure responses are directly related to the fashion query, utilizing only pertinent
48    information from our database.
49 6. Relevance Check: If a query does not align with our fashion database, guide the user to refine their
50    question or politely decline to provide an answer.
51 7. Avoiding Duplication: Ensure no response is repeated within the same interaction, maintaining uniqueness and
52    relevance to each user query.
53 8. Streamlined Communication: Eliminate any unnecessary comments or closing remarks from responses. Focus on
54    delivering clear, concise, and direct answers.
55 9. Avoid Non-essential Sign-offs: Do not include any sign-offs like "Best regards" or "FashionBot" in responses.
56 10. One-time Use Phrases: Avoid using the same phrases multiple times within the same response. Each
57    sentence should be unique and contribute to the overall message without redundancy.
58
59 Fashion Query:
60 {context}
61
62 Question: {question}
63
64 Answer:
65 """
66
67 custom_prompt = PromptTemplate(template=prompt_template, input_variables=["context", "question"])
68

```

5. Response Generation Mechanism:

- Implement a function `get_response` that takes a user's question, processes it through the RetrievalQA chain, and extracts the answer from the model's output. This ensures that the responses are not only accurate but also tailored to the user's specific inquiry.

```

69 rag_chain = RetrievalQA.from_chain_type(
70     llm=hf_hub_llm,
71     chain_type="stuff",
72     retriever=vector_store.as_retriever(top_k=3), # retriever is set to fetch top 3 results
73     chain_type_kwargs={"prompt": custom_prompt})
74
75 def get_response(question):
76     result = rag_chain({"query": question})
77     response_text = result["result"]
78     answer_start = response_text.find("Answer:") + len("Answer:")
79     answer = response_text[answer_start:].strip()
80     return answer

```

6. Streamlit Interface Configuration:

- Customize the Streamlit layout to remove unnecessary whitespace and add aesthetic elements like headers and sidebar messages.

```

82 # Streamlit app
83 # Remove whitespace from the top of the page and sidebar
84 st.markdown(
85     """
86     <style>
87     .appview-container .main .block-container {{
88         padding-top: {padding_top}rem;
89         padding-bottom: {padding_bottom}rem;
90     }}
91
92     </style>""".format(
93         padding_top=1, padding_bottom=1
94     ),
95     unsafe_allow_html=True,
96 )
97
98 # st.header("### Discover the AI styling recommendations :dress:", divider='grey')
99 st.markdown("""
100 <h3 style='text-align: left; color: black; padding-top: 35px; border-bottom: 3px solid red;'>
101     Discover the AI Styling Recommendations 🧢👗
102 </h3>""", unsafe_allow_html=True)
103

```

- Create interactive elements such as chat input and buttons to clear the chat history, enhancing user interaction.

```

104
105 side_bar_message = """
106 Hi! 🧢 I'm here to help you with your fashion choices. What would you like to know or explore?
107 \nHere are some areas you might be interested in:
108 1. **Fashion Trends** 🧢👗
109 2. **Personal Styling Advice** 👗👗
110 3. **Seasonal Wardrobe Selections** 🌞
111 4. **Accessory Recommendations** 🧢
112
113 Feel free to ask me anything about fashion!
114 """
115
116 with st.sidebar:
117     st.title('🧢 FashionBot: Your AI Style Companion')
118     st.markdown(side_bar_message)
119
120 initial_message = """
121 Hi there! I'm your FashionBot 🧢
122 Here are some questions you might ask me:\n
123 🧢 What are the top fashion trends this summer?\n
124 🧢 Can you suggest an outfit for a summer wedding?\n
125 🧢 What are some must-have accessories for winter season?\n
126 🧢 What type of shoes should I wear with a cocktail dress?\n
127 🧢 What's the best look for a professional photo shoot?
128 """

```

7. User Interaction Flow:

- Users input their fashion queries through a chat interface.
- The system processes these inputs, retrieves relevant information, and the language model generates responses which are then displayed back to the user in the chat interface.

```

130 # Store LLM generated responses
131 if "messages" not in st.session_state.keys():
132     st.session_state.messages = [{"role": "assistant", "content": initial_message}]
133
134 # Display or clear chat messages
135 for message in st.session_state.messages:
136     with st.chat_message(message["role"]):
137         st.markdown(message["content"])
138
139 def clear_chat_history():
140     st.session_state.messages = [{"role": "assistant", "content": initial_message}]
141 st.button('Clear Chat', on_click=clear_chat_history)
142
143 # User-provided prompt
144 if prompt := st.chat_input():
145     st.session_state.messages.append({"role": "user", "content": prompt})
146     with st.chat_message("user"):
147         st.markdown(prompt)
148
149 # Generate a new response if last message is not from assistant
150 if st.session_state.messages[-1]["role"] != "assistant":
151     with st.chat_message("assistant"):
152         with st.spinner("Hold on, I'm fetching the latest fashion advice for you..."):
153             response = get_response(prompt)
154             placeholder = st.empty()
155             full_response = response # Directly use the response
156             placeholder.markdown(full_response)
157         message = {"role": "assistant", "content": full_response}
158     st.session_state.messages.append(message)

```

Prompt Engineering for Mock Data

For FashionBot, prompt engineering was crucial in generating mock data that simulates realistic user interactions with the system. This mock data is then used to train and refine the application's ability to respond accurately and relevantly to user queries through the language model (LLM).

The specific prompt pattern used in this process is designed to guide the LLM in generating responses that adhere to predetermined standards of relevance and engagement.

Prompt pattern

The prompt template used for FashionBot is structured as a detailed guideline that instructs the LLM on how to formulate responses based on user queries within the domain of fashion. This template follows a **Directive-based prompt pattern**, which is

characterized by its explicit instructions on how to handle various aspects of user interaction. Here's how this pattern aids in generating responses:

```
prompt_template = """
As a highly knowledgeable fashion assistant, your role is to accurately interpret fashion queries and
provide responses using our specialized fashion database. Follow these directives to ensure optimal user interactions:
1. Precision in Answers: Respond solely with information directly relevant to the user's query from our fashion database.
   Refrain from making assumptions or adding extraneous details.
2. Topic Relevance: Limit your expertise to specific fashion-related areas:
   - Fashion Trends
   - Personal Styling Advice
   - Seasonal Wardrobe Selections
   - Accessory Recommendations
3. Handling Off-topic Queries: For questions unrelated to fashion (e.g., general knowledge questions like "Why is the sky blue?"),
   politely inform the user that the query is outside the chatbot's scope and suggest redirecting to fashion-related inquiries.
4. Promoting Fashion Awareness: Craft responses that emphasize good fashion sense, aligning with the latest trends and
   personalized style recommendations.
5. Contextual Accuracy: Ensure responses are directly related to the fashion query, utilizing only pertinent
   information from our database.
6. Relevance Check: If a query does not align with our fashion database, guide the user to refine their
   question or politely decline to provide an answer.
7. Avoiding Duplication: Ensure no response is repeated within the same interaction, maintaining uniqueness and
   relevance to each user query.
8. Streamlined Communication: Eliminate any unnecessary comments or closing remarks from responses. Focus on
   delivering clear, concise, and direct answers.
9. Avoid Non-essential Sign-offs: Do not include any sign-offs like "Best regards" or "FashionBot" in responses.
10. One-time Use Phrases: Avoid using the same phrases multiple times within the same response. Each
   sentence should be unique and contribute to the overall message without redundancy.
```

Evaluation and Testing

To ensure that FashionBot operates effectively and provides accurate fashion advice, a thorough evaluation and testing phase was conducted. This involved querying the system with a variety of inputs, ranging from simple to complex queries, to test the bot's responsiveness, accuracy, and ability to handle different types of user requests.

Testing Methodology

FashionBot was subjected to an extensive series of tests involving various types of queries:

1. **Simple Queries:** These straightforward queries are designed to assess the bot's ability to retrieve direct information from the database. Examples include:

- "What are the top fashion trends this summer?"



What are the top fashion trends this summer?



This summer, classic blue denim pants with a relaxed fit, adjustable straps, and front bib pocket are a must-have for casual, effortless style. Pair them with the vibrant Fiesta Maxi Skirt in multicolor print for a statement look. Don't forget to add a touch of rustic charm with the Gladiator Sandals in rustic brown leather. These fashion trends are sure to keep you looking fresh and stylish throughout the summer season.

- **"What type of shoes should I wear with a cocktail dress?"**



What type of shoes should I wear with a cocktail dress?



Based on your query, I recommend pairing a cocktail dress with a pair of Crimson Velvet Heels. The stiletto heel and pointed toe will complement the dress's style, while the cushioned insole and ankle strap will provide comfort and support. The Deep Red color will also match the dress's color, creating a cohesive and stylish look.

2. Complex/Nuanced Queries: These queries test the bot's capability to handle detailed inquiries and provide personalized advice. Examples include:

- **"i want to visit a museum in afternoon, and then have lunch party at a place , please suggest me some clothing outfit that will suit both the occasion"**



i want to visit a museum in afternoon, and then have lunch party at a place , please suggest me some clothing outfit that will suit both the occasion



For a museum visit in the afternoon, I recommend wearing something comfortable and stylish. Considering the "Heritage Wool Sweater" in Burgundy, it would be a great choice. The cable knit and crew neck will add a touch of sophistication, while the wool material will keep you warm and cozy. Pair it with a pair of dark jeans or trousers and loafers or boots for a chic and put-together look.

For the lunch party, you can easily dress up the outfit by adding a blazer or a cardigan. The "Nomad Leather Backpack" in Royal Blue would be a great accessory to add a pop of color and a casual touch. You can also add a statement piece of jewelry, such as a bold necklace or earrings, to elevate the outfit.

Remember to dress according to the dress code specified by the party host, if any. Overall, this outfit combination will help you look stylish and put-together for both occasions.

- **"What are some ways to style a white t-shirt for different occasions?"**



What are some ways to style a white t-shirt for different occasions



For a casual look, you can pair a white t-shirt with distressed denim jeans and sneakers for a relaxed, everyday outfit. For a more formal occasion, you can dress up a white t-shirt with a blazer and trousers for a stylish and sophisticated look. You can also add a pop of color with a statement necklace or a bold scarf to elevate the outfit. For a more bohemian look, you can layer a white t-shirt under a flowy cardigan or a kimono for a cozy and eclectic outfit. Ultimately, the key to styling a white t-shirt is to balance it with other pieces that complement its simplicity and versatility.

3. **Scenario-Based Queries:** These are practical, contextual queries that require the bot to understand and integrate multiple factors to provide suitable advice.

Examples include:

- **"I have a travel plan this weekend that includes visiting historical sites, shopping, and attending a concert. Could you please suggest outfits and accessories to match styles for all these occasions?"**



"I have a travel plan this weekend that includes visiting historical sites, shopping, and attending a concert. Could you please suggest outfits and accessories to match styles for all these occasions?"



Based on your travel plan, I recommend the following outfits and accessories:

For historical sites:

- Wear the Safari Linen Vest in Khaki, which is lightweight and suitable for warm weather. Pair it with a white or light-colored shirt and comfortable pants or a flowy skirt.

For shopping:

- The Celestial Silk Nightgown in Sky Blue is a versatile and elegant option for a casual shopping trip. You can wear it with sandals or sneakers and add a statement necklace for a stylish touch.

For the concert:

- The clutch in Royal Purple is a great accessory for a night out. You can wear it with a simple white or black top and high-waisted pants or a flowy skirt. Add some statement earrings and a bold lip color to complete the look.

Remember to dress according to the venue's dress code and the weather forecast. Have a great trip!

- **"I'm going on a first date and want to make a good impression. What should I wear and what are some styling tips?"**



I'm going on a first date and want to make a good impression. What should I wear and what are some styling tips?



For a first date, you want to make a good impression, and fashion plays a significant role in that. Considering the items in our database, I recommend the "Celestial Silk Nightgown" in Sky Blue. It's a floor-length dress with adjustable straps, lace trim, and a soft, flowing silhouette. This will make you look elegant and sophisticated.

Styling tips:

- Pair the dress with some statement jewelry, like a bold necklace or earrings, to add a touch of glamour.
- Keep your hair simple and sleek, perhaps in a low bun or a ponytail.
- Opt for a pair of heeled sandals or pumps to elongate your legs and add height.
- Keep your makeup natural and subtle, focusing on enhancing your natural features.

Remember, confidence is key, so own that dress and have fun on your date!

- 4. Irrelevant/Out-of-Scope Queries:** These queries are used to ensure that the bot can appropriately respond to or redirect queries that are not related to its domain. Examples include:

- **"Do you have any movie collection?"**



Do you have any movie collection?



"Sorry, I'm a fashion assistant, and I don't have information on movie collections. I can help you with fashion-related queries, though! What's your fashion concern or question?"

- **"Why is the sky blue?"**



Why is the sky blue?"



I'm happy to help you with your fashion-related questions! However, the query "Why is the sky blue?" is outside the scope of our fashion database. If you'd like to know more about the latest fashion trends or receive personalized styling advice, I'd be delighted to assist you. Please feel free to rephrase your question or ask something related to fashion.

5. Misspelled Queries: Testing the bot's ability to understand and correctly respond to queries with incorrect spelling, ensuring robustness in user interaction.

• “men cliothin recommenations?”



• “men cliothin recommenations?”



As a fashion assistant, I'd be happy to help you with some men's clothing recommendations! Based on your query, I'll provide you with some suggestions that cater to various styles and preferences.

For a casual look, I recommend the "Silhouette Slim Jeans" in Dark Wash. They're a versatile and comfortable option that can be paired with a variety of tops and shoes.

If you're looking for something more formal, the "Executive Winter Coat" in Charcoal Grey would be an excellent choice. Its slim fit and double-breasted design make it perfect for a professional setting.

For a sporty vibe, the "Alpine Trek Pants" in Black are a great option. They're waterproof, breathable, and feature reinforced knees for added durability.

Lastly, if you're looking for a stylish accessory, the "Leather Ankle Boots" in Brown would be a great addition to your wardrobe. They're waterproof, ankle-supportive, and feature a rugged sole and lace-up closure.

I hope these recommendations help you find the perfect outfit for your needs! If you have any further questions or would like more suggestions, feel free to ask!

Testing Results

FashionBot demonstrated a high degree of effectiveness with **Meta-Llama-3-8B-Instruct** model in handling both simple and complex queries, providing sensible and accurate responses across most tested scenarios. The bot was particularly adept at offering detailed fashion advice tailored to specific activities and preferences. It also effectively managed irrelevant and out-of-scope queries by politely redirecting them or suggesting relevant fashion-related questions.

LLM Model Performance Comparison

The following table compares the performance of the two Hugging Face models based on their response quality and relevance:

Model Name	Response Quality	Scope Relevance	Notes
Meta-Llama-3-8B-Instruct	High	High	Consistently relevant and accurate.
Mistral-7B-Instruct-v0.3	Moderate	Low	Often vague and out of scope.

The **Meta-Llama-3-8B-Instruct** model, which performed significantly **better**, is utilized for generating responses. This model has been configured to maintain contextual relevance and accuracy, focusing on generating responses that are precise and tailored to the fashion-related queries.

Challenges and Overcome solutions

I faced multiple changes throughout the development process of FashionBot, which are listed below:

1. Dependency Installation Issues

ChromaDB Installation:

- **Challenge:** Initial attempts to install ChromaDB were unsuccessful due to compatibility issues with Python 3.12.
- **Solution:** The Python environment was downgraded to Python 3.10, which supported ChromaDB installations without conflicts.

pysqlite Installation:

- **Challenge:** Difficulties were faced when installing pysqlite due to unavailable compatible versions for the current Python setup.
- **Solution:** A manual installation approach was adopted by downloading the zip file from the official site. The pysqlite wheels were then installed using this zip file within the virtual environment, ensuring proper integration with the project's Python version.

2. Language Model Configuration

- **Challenge:** Configuring the Hugging Face model initially led to several runtime errors and inefficiencies in query processing.
- **Solution:** Adjustments were made to the model's configuration parameters, such as `max_new_tokens` and `temperature`, to enhance performance and response quality. These tweaks ensured that the model generated more precise and contextually appropriate answers.

3. Streamlit Application Deployment

- **Challenge:** The deployment of the Streamlit application faced hurdles in maintaining session state and managing user interactions effectively.
- **Solution:** Enhanced session management logic was implemented using Streamlit's native capabilities to store conversation history and manage state between user interactions. This improved the usability and functionality of the chatbot interface.

Conclusion

FashionBot's development showcases a successful integration of AI with the fashion industry. Utilizing technologies like LangChain, Hugging Face Transformers, and the Chroma vector database, it provides personalized fashion advice through an intuitive user interface. FashionBot excels in handling a variety of user queries, from simple requests to complex scenarios, demonstrating high accuracy and robustness against irrelevant or misspelled inputs. Its comprehensive testing confirms its reliability and effectiveness, making FashionBot a pioneering tool for AI-driven fashion recommendations.

The development of the FashionBot was a challenging yet rewarding project. It demonstrated the effective use of advanced technologies in creating a practical and user-friendly application. The challenges encountered were significant learning opportunities, leading to a robust and efficient system ready for real-world usage.