Project Report: Personalized Fashion Recommendation and Virtual Try-On App

Sanjay Bhaskar Kashyap NUID: 002780659

Snehil Aryan NUID: 002767640

Lissa Rodrigues NUID: 002769776

August 16, 2024

Contents

Chapter 1

Introduction

1.1 Problem Statement

Your company is interested in building a personalized app that recommends products similar to the ones users upload. The app will also include a chatbot feature where users can augment their pictures with text to receive more refined recommendations. For example, a user might ask, "Can you recommend a shirt that goes with these trousers?" while uploading a picture of the trousers.

1.2 Objectives

The primary objective of this project is to create a fully functional application that leverages generative AI, Retrieval-Augmented Generation (RAG), and Large Language Models (LLMs) to offer personalized product recommendations. The app provides a seamless and interactive shopping experience by integrating image processing, text-based queries, and virtual try-on capabilities.

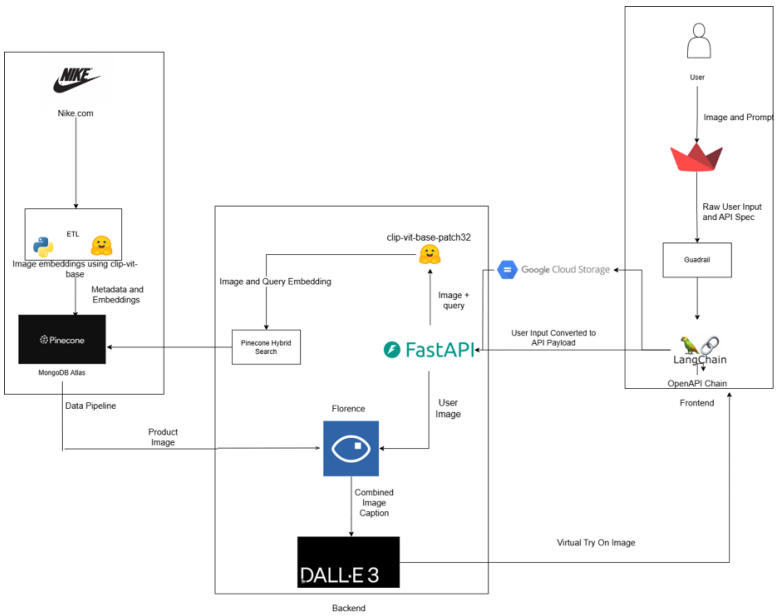

Figure 1.1: Architecture Diagram

Chapter 2

Detailed Explanation of the Use Case

2.1 Technology Stack

- **Frontend:** Streamlit for building the user interface.
- **Backend:** FastAPI for handling API requests and responses.
- **Vector Database:** Pinecone for storing and retrieving image embeddings.
- **Image Embedding Model:** CLIP for generating vector embeddings from images.
- **Language Models:** fine tune, DALL-E, Florence-2 for natural language processing and image generation.
- **Cloud Storage:** Google Cloud Storage for storing user images and processed data.

2.2 User Interface

The user interface, built with Streamlit, features a dark-themed design with chat-based interaction. Users can either input text queries or upload images to receive personalized fashion recommendations. The layout is user-friendly, with intuitive navigation and responsive elements, enhancing the overall user experience.

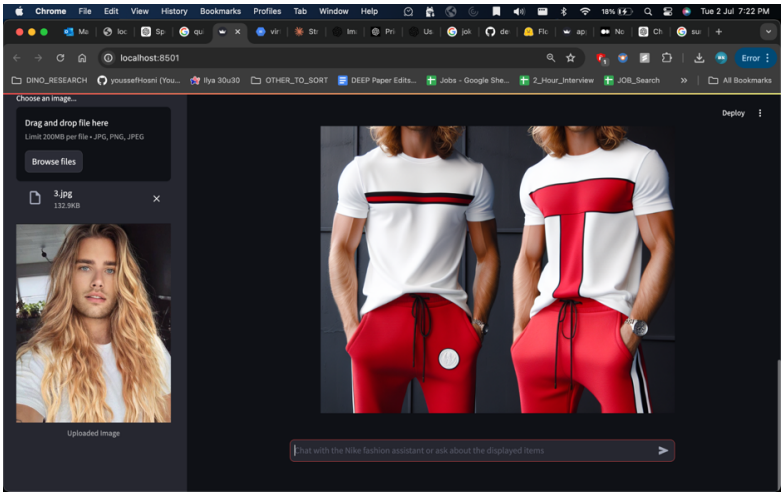

Figure 2.1: User Interface

2.2.1 Streamlit Interface Example

```
import streamlit as st
from PIL import Image
import requests

st.title("NikeBot: Your Personalized Fashion Assistant")
uploaded_file = st.file_uploader("Upload an image of your clothing item", type=["jpg", "png", "jpeg"])
query = st.text_input("Or ask for a recommendation")

if uploaded_file is not None:
    image = Image.open(uploaded_file)
    st.image(image, caption='Uploaded Image',
        use_column_width=True)
    # Call API to get recommendations based on the image
    response = requests.post("http://backend/api/
        fetch_similar_given_image", data={"image_url":
        uploaded_file})
    st.write("Recommendations:", response.json())

if query:
    # Call API to get recommendations based on the text query
    response = requests.post("http://backend/api/chat_fashion
        ", data={"query": query})
    st.write("Recommendations:", response.json())
```

2.3 Data Pipeline - Scraping & Storing Data

The data pipeline begins with scraping product details from Nike's website using BeautifulSoup and Selenium. The scraped data is stored in a Pandas DataFrame and saved to CSV files for further processing.

2.3.1 Data Scraping with BeautifulSoup and Selenium

```
from selenium import webdriver
from bs4 import BeautifulSoup
import pandas as pd

# Initialize Selenium WebDriver
driver = webdriver.Chrome()

# URL of the Nike product page
url = "https://www.nike.com/w/mens-clothing-6ymx6znik1"
driver.get(url)

# Scroll the page to load dynamic content
driver.execute_script("window.scrollTo(0, document.body.
    scrollHeight);")

# Parse the page source with BeautifulSoup
soup = BeautifulSoup(driver.page_source, 'html.parser')

# Extract product details
products = []
for product in soup.find_all('div', class_='product-card'):
    title = product.find('div', class_='product-card__title')
        .text
    price = product.find('div', class_='product-card__price')
        .text
    image_url = product.find('img', class_='product-card__img
        ')['src']
    products.append({'title': title, 'price': price, '
        image_url': image_url})

# Store data in a Pandas DataFrame
df = pd.DataFrame(products)
df.to_csv('nike_mens_clothing.csv', index=False)
```

2.4 Integration with CLIP Model

The CLIP model is used to generate embeddings for product images. These embeddings are stored in Pinecone for scalable and efficient retrieval during the recommendation process.

2.4.1 Generating Embeddings with CLIP and Storing in Pinecone

```
import torch
from transformers import CLIPProcessor, CLIPModel
import pinecone

# Initialize CLIP model and processor
model = CLIPModel.from_pretrained("openai/clip-vit-base-
    patch32")
processor = CLIPProcessor.from_pretrained("openai/clip-vit-
    base-patch32")

# Initialize Pinecone
pinecone.init(api_key='your-pinecone-api-key', environment='
    us-west1-gcp')

# Load images and generate embeddings
image_path = "path_to_image.jpg"
image = processor(images=image_path, return_tensors="pt")
with torch.no_grad():
    image_features = model.get_image_features(**image)

# Convert features to a list and upsert to Pinecone
image_features = image_features.tolist()
pinecone_index = pinecone.Index("fashion-recommendations")
pinecone_index.upsert([(image_path, image_features)])
```

2.5 Search via Text & Image Functionality

The app provides APIs that allow users to search for similar products based on text descriptions or images. The text-based search uses the fine tune gpt-4o-mini model to understand user queries and recommend relevant products.

Custom GPT Model Fine-Tuning

To enhance the capabilities of our fashion recommendation system, we implemented a fine-tuned version of the Custom GPT Model. This custom model is specifically tailored to understand and respond to queries related to fashion, product recommendations, and virtual try-ons.

```
{
  "messages": [
    {
      "role": "system",
      "content": "You are a knowledgeable salesperson
         specializing in Nike men's clothing."
    },
    {
      "role": "user",
      "content": "Can you recommend a lightweight
         jacket for running?"
    },
    {
      "role": "assistant",
      "content": "Sure! The Nike Windrunner is perfect
         for running. It's lightweight,
                  water-resistant, and designed to keep
                     you comfortable during your
                     workout."
    }
  ]
}
```

Data Preparation

We began by preparing a comprehensive dataset that included real-world scenarios encountered by salespersons. This dataset was crafted to capture various customer interactions, such as product inquiries, fashion advice, and personalized recommendations. Each interaction was structured to include a system prompt, user query, and an appropriate response from the assistant.

For instance:

```
{
  "messages": [
    {
```

```
      "role": "system",
      "content": "You are a knowledgeable salesperson
         specializing in Nike men's clothing."
    },
    {
      "role": "user",
      "content": "Can you recommend a lightweight
         jacket for running?"
    },
    {
      "role": "assistant",
      "content": "Sure! The Nike Windrunner is perfect
         for running. It's lightweight,
                water-resistant, and designed to keep
                   you comfortable during your
                   workout."
    }
  ]
}
```

Fine-Tuning Process

Using the OpenAI API, we fine-tuned the Custom GPT Model with this dataset. The fine-tuning process involved training the model on these specific interactions, allowing it to learn the nuances of fashion-related queries. This customization helped the model provide more relevant and contextually accurate responses.

2.5.1 Fetch Similar API Using FastAPI

```
from fastapi import FastAPI, Query, HTTPException
from pydantic import BaseModel
from typing import List

app = FastAPI()

class Product(BaseModel):
    id: str
    score: float
    metadata: dict

@app.post("/fetch_similar_given_image", response_model=List[
   Product])
```

```python
async def fetch_similar_given_image(image_url: str = Query(...)):
    # Implement search logic with Pinecone
    search_results = search_by_image(image_url)
    return [{"id": match.id, "score": match.score, "metadata": match.metadata} for match in search_results.matches]

@app.post("/fetch_similar_given_text", response_model=List[Product])
async def fetch_similar_given_text(description: str = Query(...)):
    # Implement search logic with Pinecone
    search_results = search_by_text(description)
    return [{"id": match.id, "score": match.score, "metadata": match.metadata} for match in search_results.matches]
```

2.6 Virtual Try-On Option with Florence-2 and DALL-E

The virtual try-on feature allows users to visualize how clothing items would look on them by leveraging Florence-2 for descriptive captions and DALL-E for realistic image generation.

2.6.1 Virtual Try-On Implementation

```python
import openai

def virtual_try_on(user_image_url, product_image_url):
    # Remove background from user image
    processed_user_image = remove_background(user_image_url)

    # Generate captions with Florence-2
    user_caption = generate_caption(processed_user_image)
    product_caption = generate_caption(product_image_url)

    # Generate try-on image with DALL-E
    prompt = f"{user_caption} wearing {product_caption}"
    response = openai.Image.create(prompt=prompt, n=1, size="1024x1024")

    try_on_image_url = response['data'][0]['url']

    return try_on_image_url
```

Chapter 3

Key Features and Functionalities

3.1 Chat-Based Interaction

Users can interact with an AI-powered fashion assistant through a chat interface. The assistant, powered by fine tune gpt-4o-mini, provides personalized fashion advice based on user queries.

3.2 Image-Based Recommendations

The app allows users to upload images to find visually similar products or items that complement the uploaded image.

3.3 Virtual Try-On

The virtual try-on feature uses advanced image processing and AI models to generate realistic visuals of users wearing the selected products.

3.4 Token Usage and Cost Tracking

To manage costs and usage effectively, the application tracks the number of tokens used in AI interactions, along with the associated costs.

3.4.1 Token Usage Tracking Example

```python
import tiktoken

class UsageTracker:
    def __init__(self):
        self.total_cost = 0
        self.total_tokens = 0
        self.encoding = tiktoken.encoding_for_model("gpt-4o-mini-2024-07-18:personal:prompt:9wcxTPfl")

    def update_usage(self, messages, model="gpt-4o-mini-2024-07-18:personal:prompt:9wcxTPfl"):
        num_tokens = sum([len(self.encoding.encode(message)) for message in messages])
        cost = num_tokens * 0.002 / 1000  # Assuming $0.002 per 1K tokens
        self.total_tokens += num_tokens
        self.total_cost += cost
        return cost, num_tokens

# Example usage
tracker = UsageTracker()
messages = ["Hello", "Recommend me a shirt to go with these pants"]
cost, tokens = tracker.update_usage(messages)
print(f"Tokens used: {tokens}, Cost: ${cost:.4f}")
```

Chapter 4

Conclusion

This project report has outlined the development of a personalized fashion recommendation and virtual try-on app. By combining cutting-edge AI models with a user-friendly interface, the app offers a unique shopping experience that bridges the gap between online and in-store shopping.

Chapter 5

Future Work

- **Enhanced Personalization:** Incorporating user preferences and feedback to refine recommendations.
- **Social Sharing:** Allowing users to share their virtual try-on experiences on social media platforms.
- **Expanded Inventory:** Integrating additional fashion brands and categories into the recommendation engine.

Chapter 6

References

- OpenAI API Documentation: https://beta.openai.com/docs/
- Pinecone Documentation: https://docs.pinecone.io/docs/
- Streamlit Documentation: https://docs.streamlit.io/