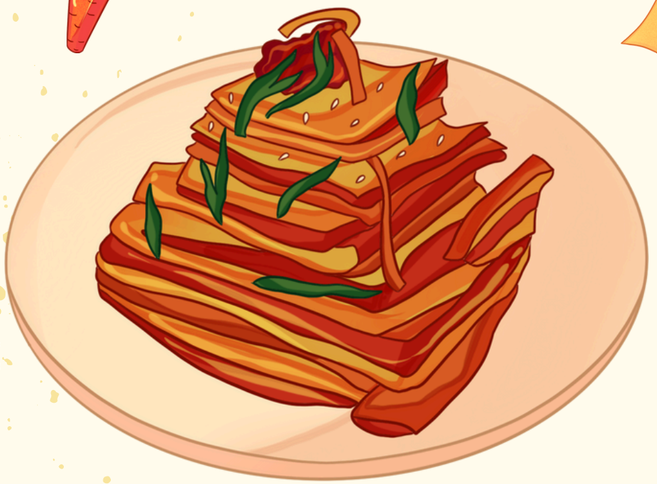




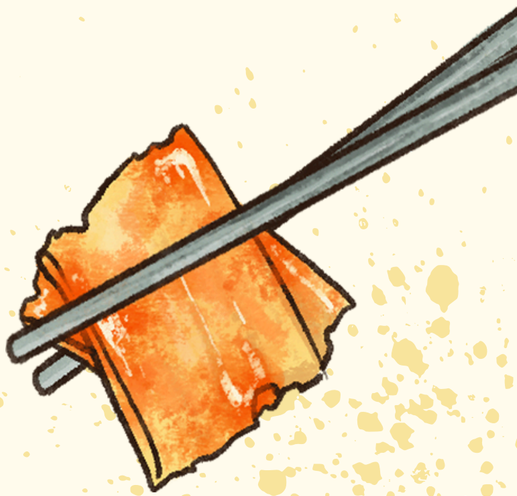


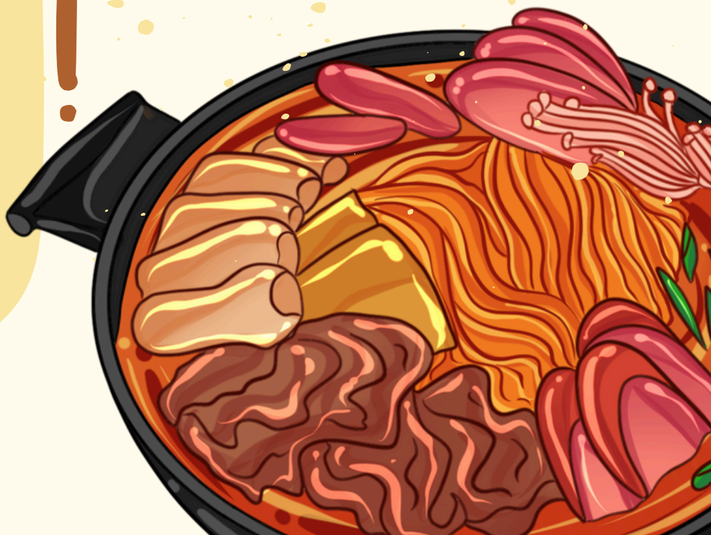

Amazon Fine Food Reviews

Presented by Caroline Berto

Objectives

- Implement a Retrieval-Augmented Generation (RAG) system.
- Explore dataset with product reviews to understand structure and content.
- Embed documents using Sentence Transformers.
- Store and manage document embeddings in a vector database (ChromaDB).
- Connect to OpenAI API for response generation.
- Develop an end-to-end pipeline for query-based document retrieval and response.
- Evaluate system performance and identify areas for improvement.





Import libraries and load the dataset

```
[2]  import pandas as pd
     import numpy as np
     import os
     import kagglehub

     ### Load Dataset: Download latest version
     path = kagglehub.dataset_download("snap/amazon-fine-food-reviews")

     # Load dataset from downloaded path
     dataset_path = f"{path}/Reviews.csv"
     df = pd.read_csv(dataset_path, nrows=1000)  # Load a small subset of 1000 rows

     # Output preview of dataset
     df.head()
```

The dataset

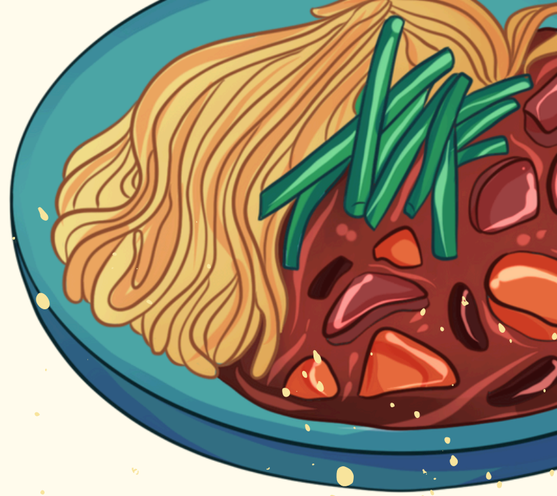
STANFORD NETWORK ANALYSIS PROJECT AND 1 COLLABORATOR · UPDATED 8 YEARS AGO

2257 | New Notebook | Download

Amazon Fine Food Reviews

Analyze ~500,000 food reviews from Amazon

EDA



```
[3]  # General overview of dataset
     df.info()

     # Distribution of reviews by score
     import matplotlib.pyplot as plt

     df['Score'].value_counts().plot(kind='bar', title='Review Distribution by Score')
     plt.show()

     # Check for missing values
     missing_values = df.isnull().sum()
     print(f"Missing Values: \n{missing_values}")

     # Understanding review length
     df['review_length'] = df['Text'].apply(len)
     df['review_length'].describe()

     # Plot histogram of review length
     df['review_length'].plot(kind='hist', title='Review Length Distribution')
     plt.show()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 10 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   Id                      1000 non-null   int64
 1   ProductId               1000 non-null   object
 2   UserId                  1000 non-null   object
 3   ProfileName             1000 non-null   object
 4   HelpfulnessNumerator    1000 non-null   int64
 5   HelpfulnessDenominator  1000 non-null   int64
 6   Score                   1000 non-null   int64
 7   Time                    1000 non-null   int64
 8   Summary                 1000 non-null   object
 9   Text                    1000 non-null   object
dtypes: int64(5), object(5)
memory usage: 78.2+ KB
```

Embedding and Storing Chunks

```
[4]  from sentence_transformers import SentenceTransformer
     from sklearn.model_selection import train_test_split

     # Load embedding model
     embedding_model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')

     # Split dataset into chunks (e.g., reviews)
     chunks = df['Text'].tolist()

     # Generate embeddings for each chunk
     embeddings = embedding_model.encode(chunks)

     # Output embedding size for first review
     print("Embedding size:", len(embeddings[0]))
```

```
Embedding size: 384
```

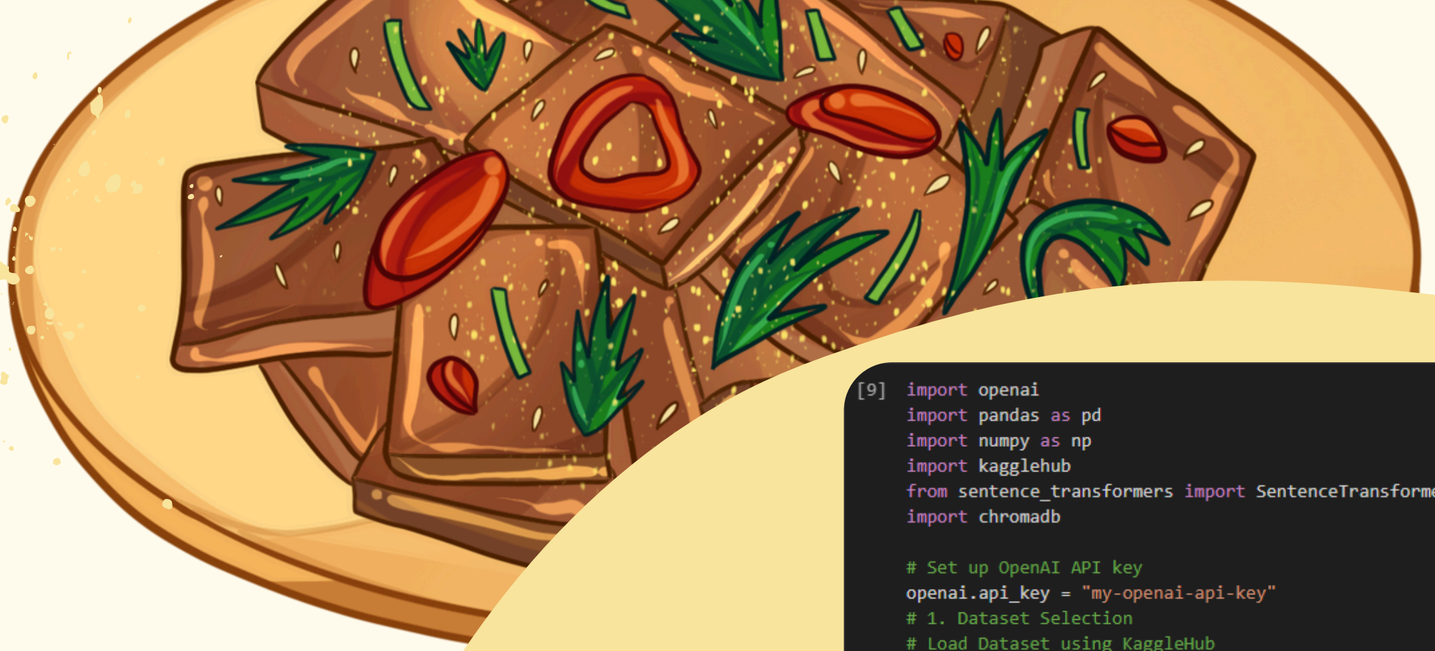
Connecting to Vector Database

```
[7]  import chromadb

     # Connect to ChromaDB client
     chroma_client = chromadb.PersistentClient()

     # Define the collection name
     collection_name = 'product_review_embeddings'

     # Create a collection in ChromaDB to store embeddings, if it doesn't already exist
     try:
         collection = chroma_client.create_collection(name=collection_name)
     except chromadb.errors.UniqueConstraintError:
         collection = chroma_client.get_collection(name=collection_name)

     # Add embeddings to the collection
     for i, embedding in enumerate(embeddings):
         collection.add(embeddings=[embedding.tolist()], metadatas=[{'content': chunks[i]}], ids=[str(i)])
```

Document Retrieval

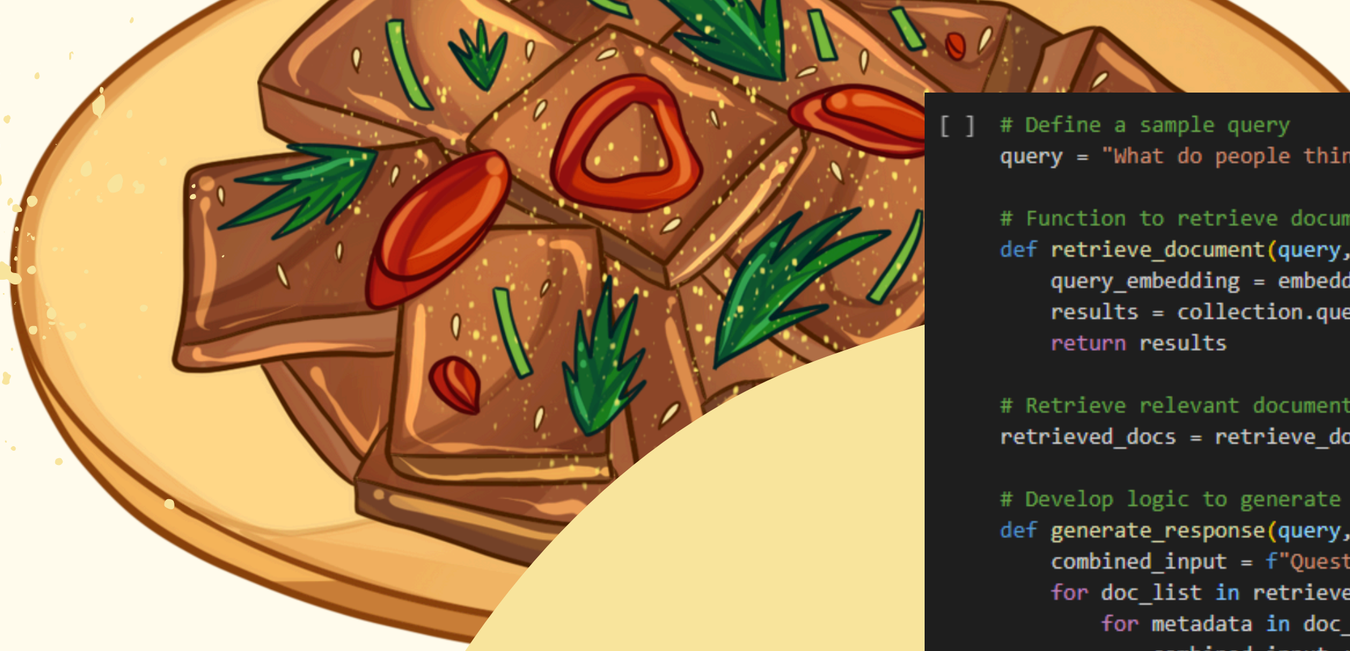
```
[ ]  # Test the retrieval logic
     def retrieve_document(query, top_k=3):
         query_embedding = embedding_model.encode([query])
         results = collection.query(query_embeddings=query_embedding, n_results=top_k)
         return results

     # Example query
     query = "What do people think about the taste of this product?"
     retrieved_docs = retrieve_document(query)
     print("Top Retrieved Documents:\n", retrieved_docs)
```

```
Top Retrieved Documents:
 {'ids': [['887', '606', '948']], 'embeddings': None, 'documents': [[None, None, None]],
```



Connecting to LLM

```
[9]  import openai
     import pandas as pd
     import numpy as np
     import kagglehub
     from sentence_transformers import SentenceTransformer
     import chromadb

     # Set up OpenAI API key
     openai.api_key = "my-openai-api-key"
     # 1. Dataset Selection
     # Load Dataset using KaggleHub
     path = kagglehub.dataset_download("snap/amazon-fine-food-reviews")
     dataset_path = f"{path}/Reviews.csv"
     df = pd.read_csv(dataset_path, nrows=1000)

     # 2. Load Embedding Model
     embedding_model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')

     # 3. Connect to ChromaDB and store embeddings
     chroma_client = chroma_client = chromadb.PersistentClient()
     collection_name = 'product_review_embeddings'

     try:
         collection = chroma_client.create_collection(name=collection_name)
     except chromadb.errors.UniqueConstraintError:
         collection = chroma_client.get_collection(name=collection_name)

     chunks = df['Text'].tolist()
     embeddings = embedding_model.encode(chunks)

     for i, embedding in enumerate(embeddings):
         collection.add(embeddings=[embedding.tolist()], metadatas=[{'content': chunks[i]}], ids=[str(i)])
```



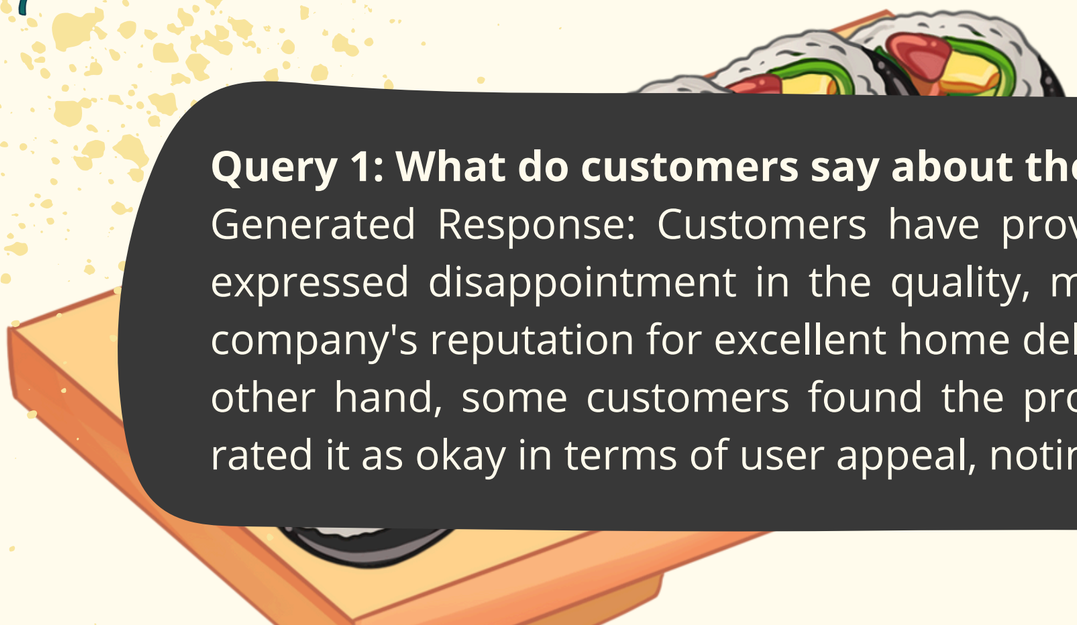
Connecting to LLM

```
[ ]  # Define a sample query
     query = "What do people think about the taste of this product?"

     # Function to retrieve documents
     def retrieve_document(query, top_k=3):
         query_embedding = embedding_model.encode([query])
         results = collection.query(query_embeddings=query_embedding, n_results=top_k)
         return results

     # Retrieve relevant documents
     retrieved_docs = retrieve_document(query)

     # Develop logic to generate response based on retrieved documents
     def generate_response(query, retrieved_docs):
         combined_input = f"Question: {query}\n\nRelevant Documents:\n"
         for doc_list in retrieved_docs['metadatas']:
             for metadata in doc_list:
                 combined_input += f"- {metadata['content']}\n"

         # Generate response using the OpenAI Chat API
         response = openai.ChatCompletion.create(
             model="gpt-3.5-turbo",
             messages=[
                 {"role": "system", "content": "You are a helpful assistant."},
                 {"role": "user", "content": combined_input}
             ],
             max_tokens=100
         )
         return response['choices'][0]['message']['content']

     # Example response generation
     response = generate_response(query, retrieved_docs)
     print(f"Generated Response: {response}")
```

```
Generated Response: Based on the feedback provided in the relevant documents, people generally have positive opinions about the taste of the product. Some describe it as absolutely delicious with great ingredients and texture. However, there are also mentions that it may be spicy, so those who do not like spicy flavors should be cautious. Overall, the general consensus seems to be that the taste is good, although not everyone may find it to be the best.
```

Evaluation

```
[ ]  # Create test set of queries
     queries = [
         "What do customers say about the quality of the product?",
         "Are people happy with the price of this item?",
         "What are the complaints regarding the packaging?"
     ]

     # Evaluate system performance manually
     for q in queries:
         docs = retrieve_document(q)
         resp = generate_response(q, docs)
         print(f"Query: {q}\nGenerated Response: {resp}\n")
```

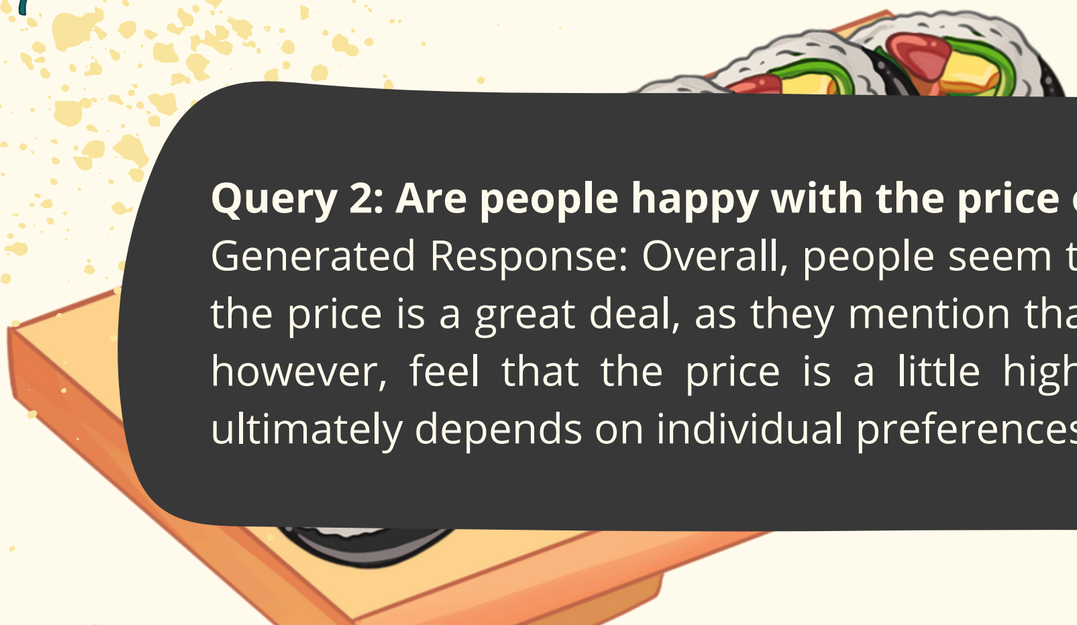


Query 1: What do customers say about the quality of the product?

Generated Response: Customers have provided mixed reviews about the product. Some customers have expressed disappointment in the quality, mentioning that it did not meet their expectations based on the company's reputation for excellent home delivery products and that it tasted like cheap thickened jelly. On the other hand, some customers found the product to be a cute stocking stuffer that arrived quickly but only rated it as okay in terms of user appeal, noting that you get what you pay for.

Evaluation

```
[ ]  # Create test set of queries
     queries = [
         "What do customers say about the quality of the product?",
         "Are people happy with the price of this item?",
         "What are the complaints regarding the packaging?"
     ]

     # Evaluate system performance manually
     for q in queries:
         docs = retrieve_document(q)
         resp = generate_response(q, docs)
         print(f"Query: {q}\nGenerated Response: {resp}\n")
```

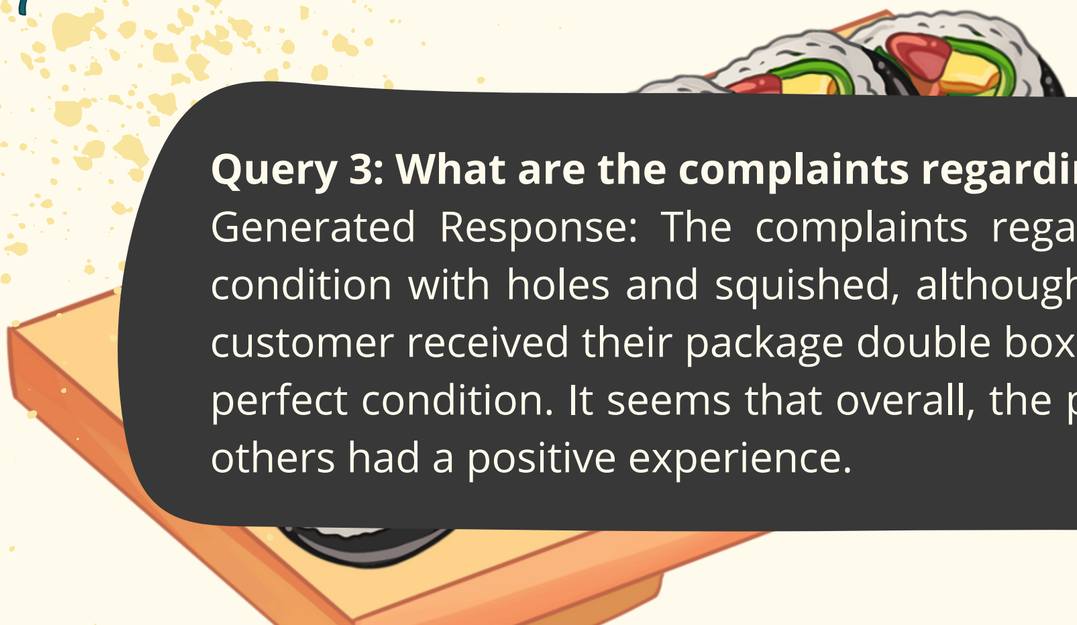


Query 2: Are people happy with the price of this item?

Generated Response: Overall, people seem to have mixed feelings about the price of the item. Some feel that the price is a great deal, as they mention that it is at least half the retail price and the quality is good. Others, however, feel that the price is a little high, especially when factoring in additional costs like shipping. It ultimately depends on individual preferences and priorities when it comes to spending.

Evaluation

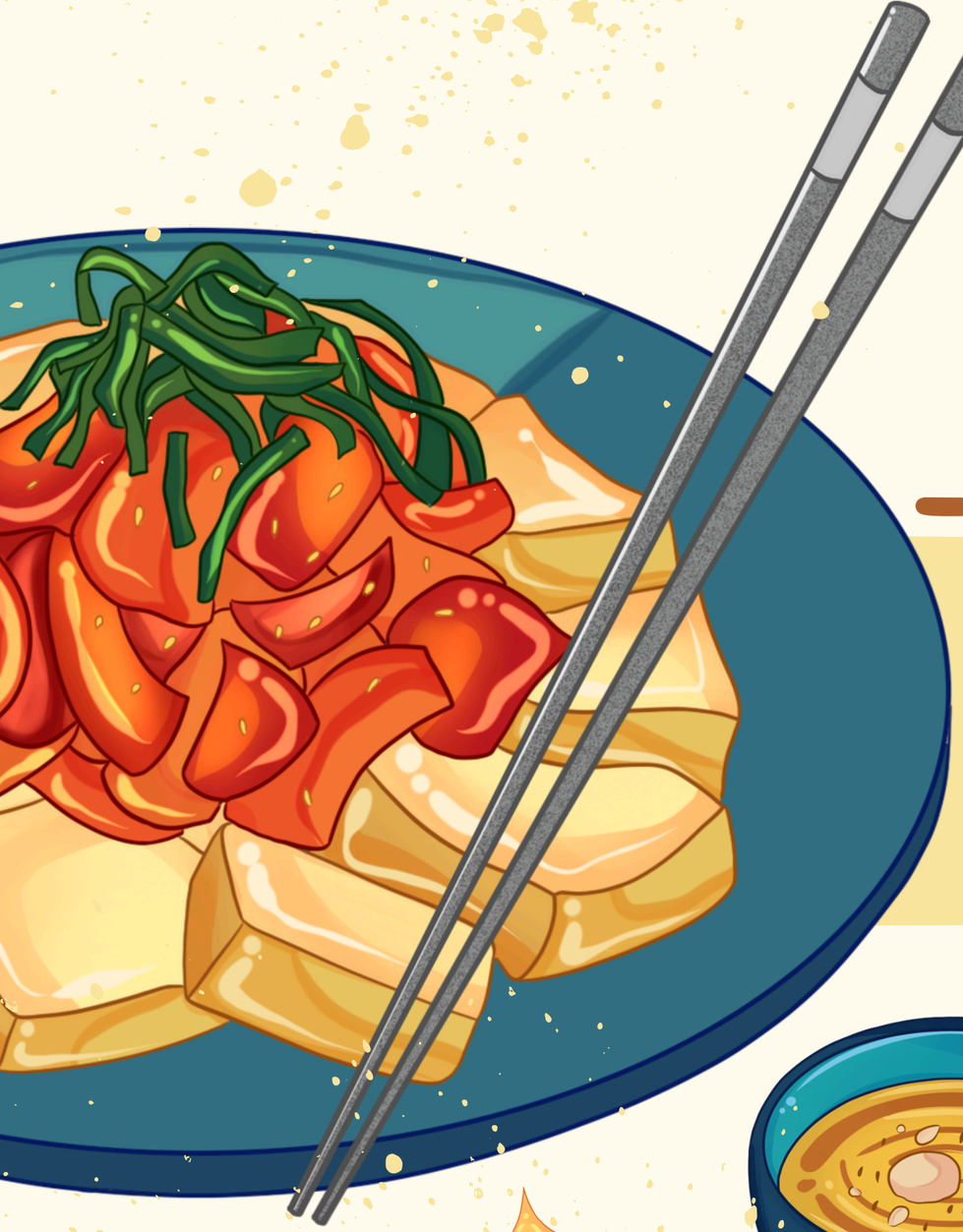
```
[ ]  # Create test set of queries
     queries = [
         "What do customers say about the quality of the product?",
         "Are people happy with the price of this item?",
         "What are the complaints regarding the packaging?"
     ]

     # Evaluate system performance manually
     for q in queries:
         docs = retrieve_document(q)
         resp = generate_response(q, docs)
         print(f"Query: {q}\nGenerated Response: {resp}\n")
```



Query 3: What are the complaints regarding the packaging?

Generated Response: The complaints regarding the packaging include receiving a box that was in poor condition with holes and squished, although the chips inside were unaffected. On the positive side, another customer received their package double boxed and wrapped securely, with the inside box holding the chips in perfect condition. It seems that overall, the packaging feedback is mixed with some experiencing issues while others had a positive experience.

Conclusion



Successfully implemented a RAG system using embeddings, ChromaDB, and OpenAI API. The system retrieves relevant information and generates responses to user queries.

Future Improvements:

- Handle longer and more complex queries effectively.
- Experiment with multimodal datasets.

Thank you for your attention!