RAG Based Food Image and Text search backend based on the data

The backend API leverages the power of Retrieval Augmented Generation (RAG) techniques to provide efficient and intelligent text and image search capabilities. RAG combines the strengths of information retrieval and natural language generation, allowing the system to retrieve relevant information from the database based on user queries and generate contextual responses using language models.

For text-based queries, the RAG-based text search module analyzes the user's input, extracts relevant entities and keywords, and retrieves the most relevant dishes, restaurants, and menu items from the database using advanced retrieval algorithms.

Similarly, the RAG-based image search module employs computer vision techniques to identify the dish or food item in the uploaded image and retrieve corresponding information from the database.

To optimize cost and leverage the power of open-source technologies, the system utilizes pre-trained language models from the Hugging Face library. By leveraging open-source models from Hugging Face, the system avoids the recurring costs associated with commercial APIs like OpenAI, while still achieving high performance and accuracy.

Furthermore, the backend API has been designed and developed as a cloud-native application, allowing for easy integration into cloud-based ecosystems. I have trailed the whole thing on the cloud ecosystem, So that Azure and the Mongo db atlas, And the whole thing worked successfully.

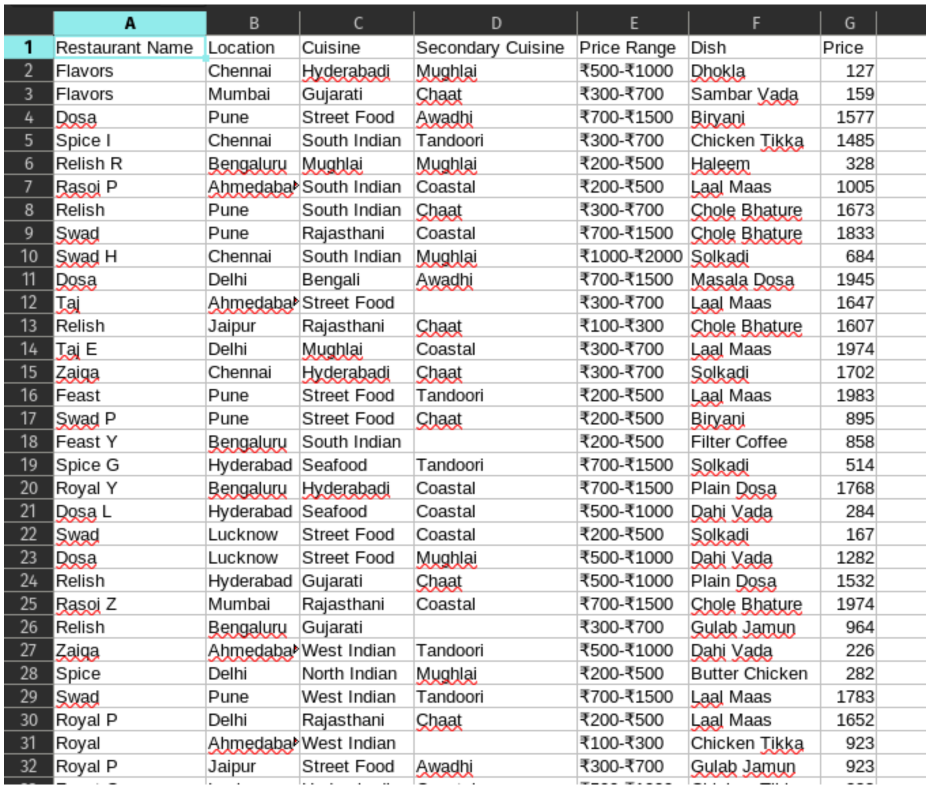
With the backend API fully developed and deployed on the cloud, the only remaining step is the integration with the WhatsApp platform. Once integrated, users will be able to access the intelligent food ordering system through the familiar WhatsApp interface, leveraging the RAG-based text and image search capabilities, language understanding, and generation features, all powered by the cloud-based backend API.

I have used two datasets one for the text and the other for the image, the text one was artificially made by me with the help of the Python script, and the other image dataset i curated it from one of the research papers.

The text dataset used in this project provides comprehensive information about various restaurants across different cities in India. It encompasses details such as the restaurant name, location, primary cuisine, secondary cuisine offerings, price ranges for dishes, specific dish names, and their corresponding prices. The locations covered include major metropolitan areas like Chennai, Mumbai, Pune, Bengaluru, and Ahmedabad, allowing for a diverse representation of regional cuisines. The primary cuisines listed range from local specialties like Hyderabadi, Gujarati, South Indian, and Rajasthani to broader categories such as Street Food and Mughlai.

Additionally, the text dataset incorporates secondary cuisine types, enabling restaurants to showcase their versatility in offering multiple culinary styles. The price range column provides an approximate guide to the pricing structure at each establishment, using rupee symbols and numerical ranges.

Furthermore, the dataset includes specific dish names and their individual prices, allowing for detailed analysis and comparison of menu offerings across different restaurants. This comprehensive dataset serves as a valuable resource for applications related to restaurant discovery, food recommendations, price comparisons, and understanding cuisine preferences and pricing trends across various regions of India. Figure 1 shows an image of the text dataset



	A	B	C	D	E	F	G
1	Restaurant Name	Location	Cuisine	Secondary Cuisine	Price Range	Dish	Price
2	Flavors	Chennai	Hyderabadi	Mughlai	₹500-₹1000	Dhokla	127
3	Flavors	Mumbai	Gujarati	Chaat	₹300-₹700	Sambar Vada	159
4	Dosa	Pune	Street Food	Awadhi	₹700-₹1500	Biryani	1577
5	Spice I	Chennai	South Indian	Tandoori	₹300-₹700	Chicken Tikka	1485
6	Relish R	Bengaluru	Mughlai	Mughlai	₹200-₹500	Haleem	328
7	Rasoi P	Ahmedaba	South Indian	Coastal	₹200-₹500	Laal Maas	1005
8	Relish	Pune	South Indian	Chaat	₹300-₹700	Chole Bhature	1673
9	Swad	Pune	Rajasthani	Coastal	₹700-₹1500	Chole Bhature	1833
10	Swad H	Chennai	South Indian	Mughlai	₹1000-₹2000	Solkadi	684
11	Dosa	Delhi	Bengali	Awadhi	₹700-₹1500	Masala Dosa	1945
12	Taj	Ahmedaba	Street Food		₹300-₹700	Laal Maas	1647
13	Relish	Jaipur	Rajasthani	Chaat	₹100-₹300	Chole Bhature	1607
14	Taj E	Delhi	Mughlai	Coastal	₹300-₹700	Laal Maas	1974
15	Zaiqa	Chennai	Hyderabadi	Chaat	₹300-₹700	Solkadi	1702
16	Feast	Pune	Street Food	Tandoori	₹200-₹500	Laal Maas	1983
17	Swad P	Pune	Street Food	Chaat	₹200-₹500	Biryani	895
18	Feast Y	Bengaluru	South Indian		₹200-₹500	Filter Coffee	858
19	Spice G	Hyderabad	Seafood	Tandoori	₹700-₹1500	Solkadi	514
20	Royal Y	Bengaluru	Hyderabadi	Coastal	₹700-₹1500	Plain Dosa	1768
21	Dosa L	Hyderabad	Seafood	Coastal	₹500-₹1000	Dahi Vada	284
22	Swad	Lucknow	Street Food	Coastal	₹200-₹500	Solkadi	167
23	Dosa	Lucknow	Street Food	Mughlai	₹500-₹1000	Dahi Vada	1282
24	Relish	Hyderabad	Gujarati	Chaat	₹500-₹1000	Plain Dosa	1532
25	Rasoi Z	Mumbai	Rajasthani	Coastal	₹700-₹1500	Chole Bhature	1974
26	Relish	Bengaluru	Gujarati		₹300-₹700	Gulab Jamun	964
27	Zaiqa	Ahmedaba	West Indian	Tandoori	₹500-₹1000	Dahi Vada	226
28	Spice	Delhi	North Indian	Mughlai	₹200-₹500	Butter Chicken	282
29	Swad	Pune	West Indian	Tandoori	₹700-₹1500	Laal Maas	1783
30	Royal P	Delhi	Rajasthani	Chaat	₹200-₹500	Laal Maas	1652
31	Royal	Ahmedaba	West Indian		₹100-₹300	Chicken Tikka	923
32	Royal P	Jaipur	Street Food	Awadhi	₹300-₹700	Gulab Jamun	923

Figure 1 - Text Dataset Overview

For the image dataset, I utilized the open-source Food-101 dataset, which is a comprehensive collection encompassing 101 distinct food categories. This dataset boasts an impressive total of 101,000 images, providing a substantial resource for image-based applications and research. The Food-101 dataset was meticulously curated and obtained from the reputable source http://data.vision.ee.ethz.ch/cvl/food-101.tar.gz.

To provide a visual representation of the dataset, Figures 2 and 3 illustrate a subset of the images contained within it. These figures offer a glimpse into the diverse array of food items present in the dataset, spanning various cuisines. The inclusion of these sample images serves to exemplify the richness and variety of the Food-101 dataset, making it a valuable resource for RAG based search.

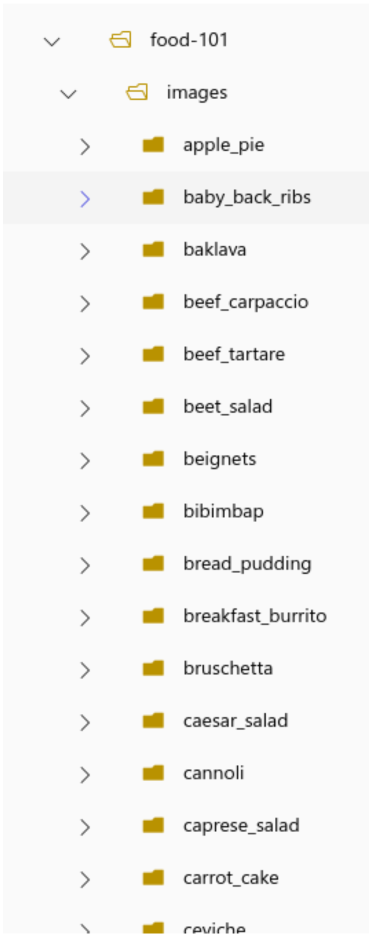

food-101
images
apple_pie
baby_back_ribs
baklava
beef_carpaccio
beef_tartare
beet_salad
beignets
bibimbap
bread_pudding
breakfast_burrito
bruschetta
caesar_salad
cannoli
caprese_salad
carrot_cake
ceviche

Figure 2 - Folder Structure of the image dataset

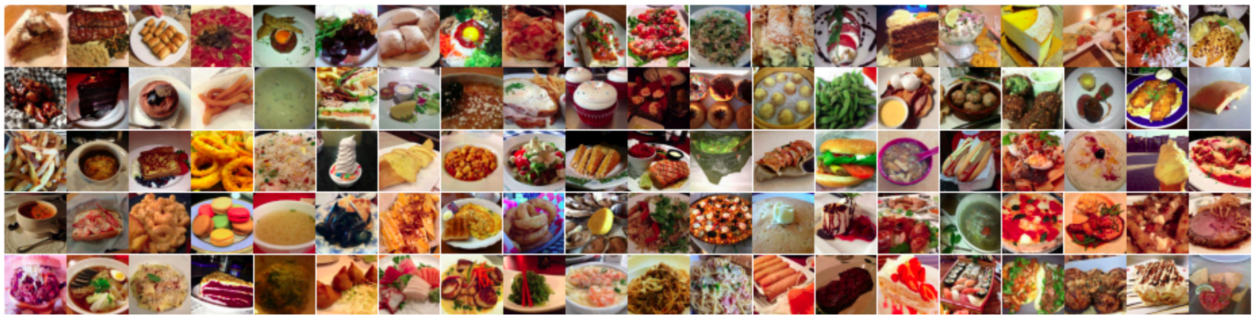
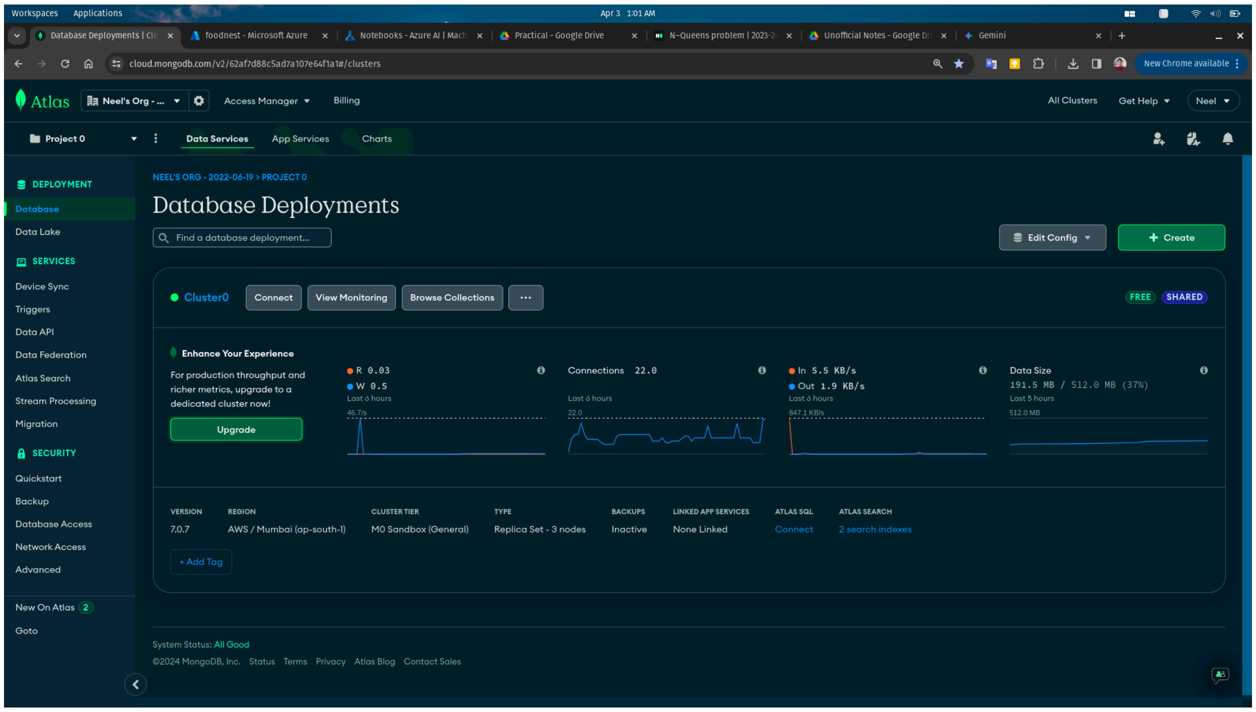


Figure 3 - Some of the pictures of the image dataset

In this project, I leveraged MongoDB, a flexible NoSQL database, to manage my text and image dataset. This choice allowed me to efficiently store both structured text data and unstructured image data within the same document.

Furthermore, MongoDB facilitated the vectorization process, where I converted the text and image data into numerical representations suitable for retrieval tasks. This vectorization step enabled the application of Ranked Answer Grouping (RAG) for search functionality. RAG is a powerful technique for organizing search results based on relevance and thematic relationships.

The successful utilization of MongoDB is showcased in Figures 4, 5, and 6, which likely illustrate the structure of the stored data or the search results obtained using the RAG-based search implemented on the MongoDB dataset.



cloud.mongodb.com/v2/62af7d88c5ad7a107e64f1a1#/clusters

R 0.03
W 0.5
Last 6 hours

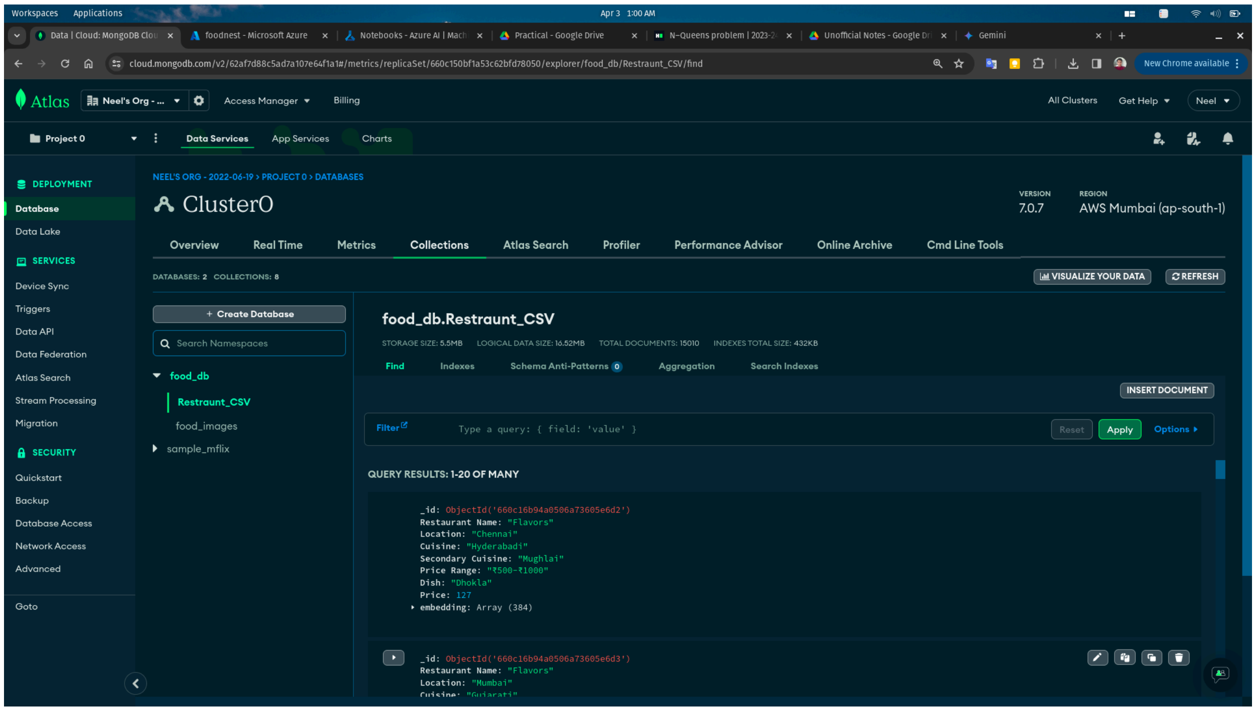
Connections 22.0
Last 6 hours

In 5.5 KB/s
Out 1.9 KB/s
Last 6 hours

Data Size
191.5 MB / 512.0 MB (37%)
Last 5 hours

VERSION	REGION	CLUSTER TIER	TYPE	BACKUPS	LINKED APP SERVICES	ATLAS SQL	ATLAS SEARCH
7.0.7	AWS / Mumbai (ap-south-1)	M0 Sandbox (General)	Replica Set - 3 nodes	Inactive	None Linked	Connect	2 search indexes

Figure 4 - MongoDB cluster image



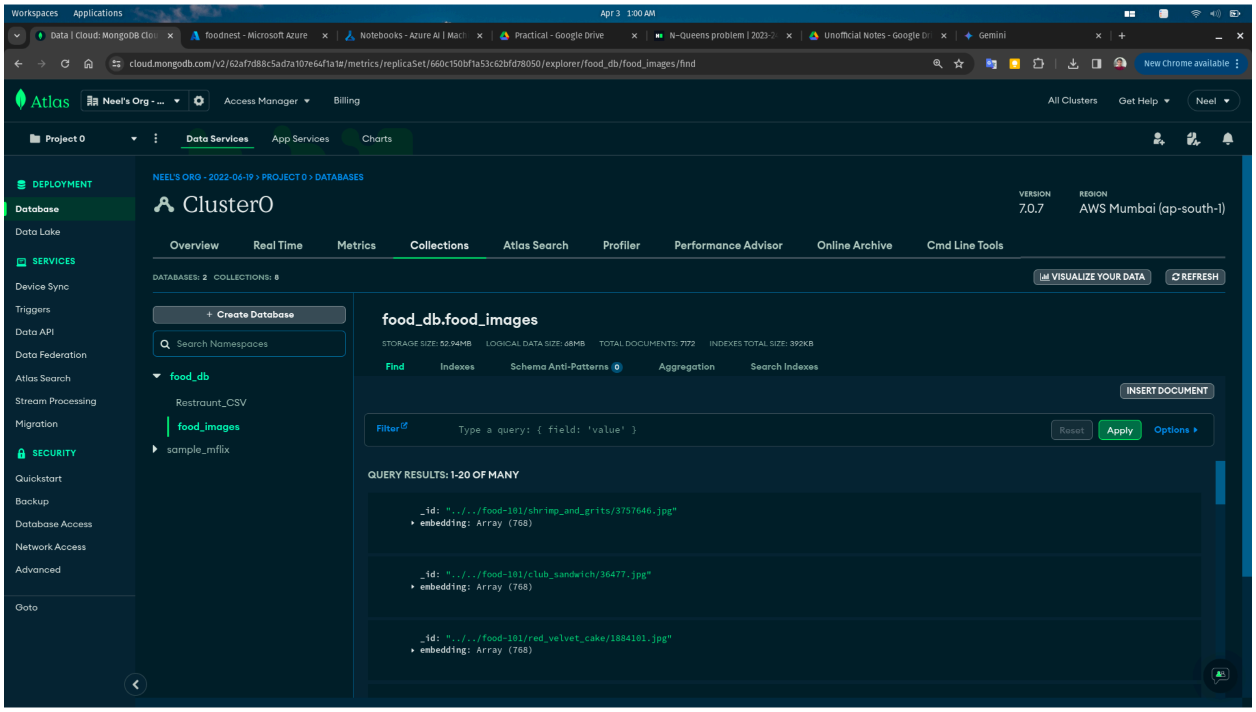
food_db.Restraunt_CSV

STORAGE SIZE: 5.5MB LOGICAL DATA SIZE: 16.52MB TOTAL DOCUMENTS: 15010 INDEXES TOTAL SIZE: 432KB

QUERY RESULTS: 1-20 OF MANY

```
_id: ObjectId('660c16b94a0506a73605e6d2')
Restaurant Name: "Flavors"
Location: "Chennai"
Cuisine: "Hyderabadi"
Secondary Cuisine: "Mughlai"
Price Range: "₹500-₹1000"
Dish: "Dhokla"
Price: 127
embedding: Array (384)

_id: ObjectId('660c16b94a0506a73605e6d3')
Restaurant Name: "Flavors"
Location: "Mumbai"
Cuisine: "Gujarati"
```

Figure 5 - MongoDB text data collection image



food_db.food_images

STORAGE SIZE: 52.94MB LOGICAL DATA SIZE: 68MB TOTAL DOCUMENTS: 7172 INDEXES TOTAL SIZE: 392KB

QUERY RESULTS: 1-20 OF MANY

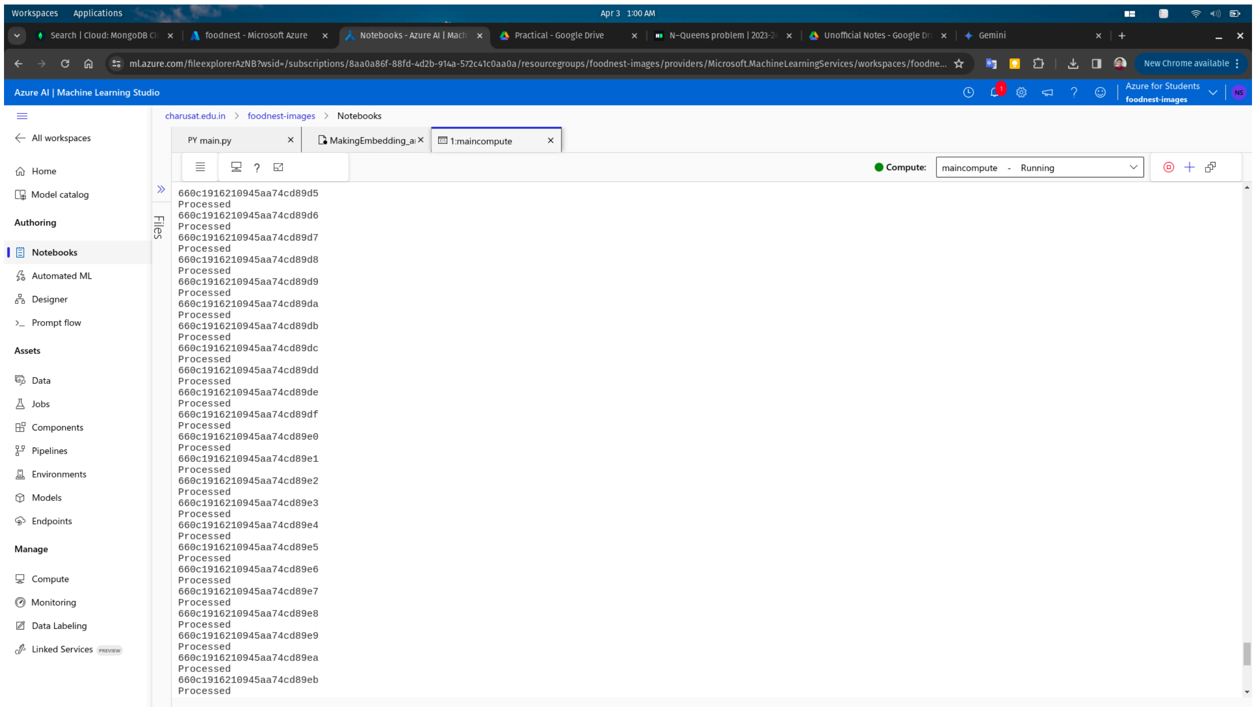
```
_id: "../../food-101/shrimp_and_grits/3757646.jpg"
embedding: Array (768)

_id: "../../food-101/club_sandwich/36477.jpg"
embedding: Array (768)

_id: "../../food-101/red_velvet_cake/1884101.jpg"
embedding: Array (768)
```

Figure 6 - MongoDB image data collection image

To create a user-friendly interface and handle search requests, I implemented a custom API using Flask, a lightweight web framework. This API acts as the bridge between the user and the underlying RAG search system.

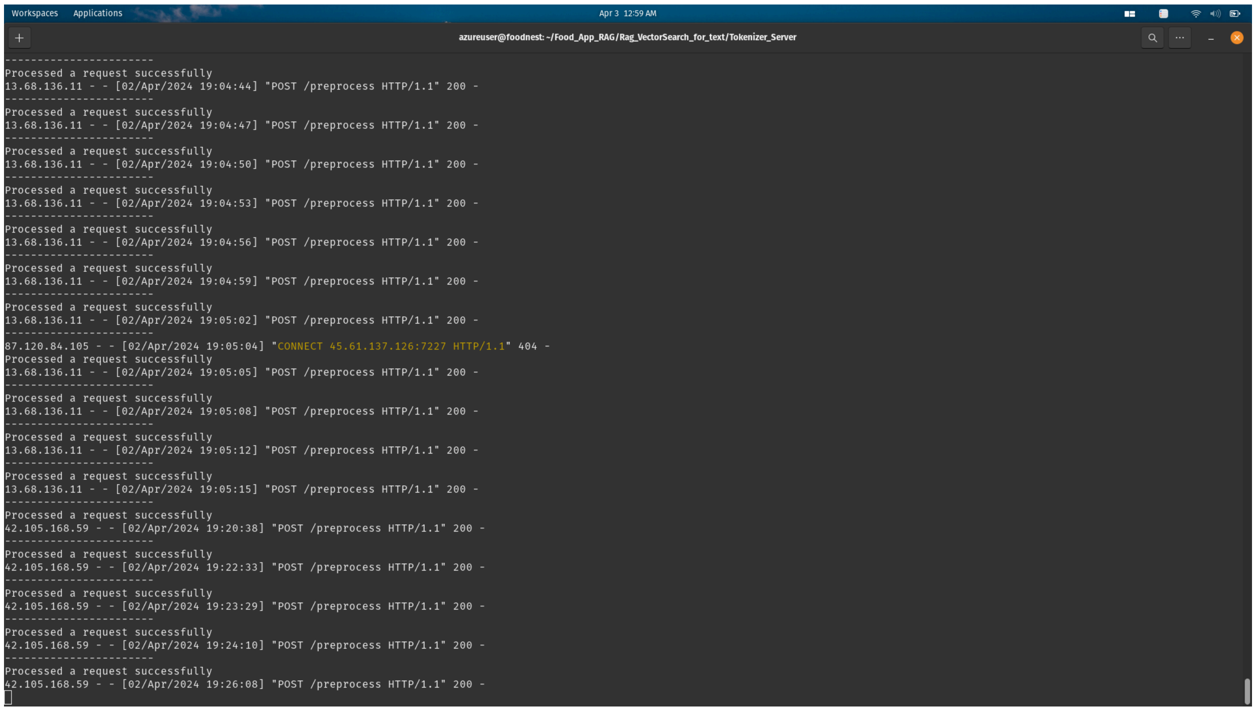
Crucially, I didn't rely on any paid, pre-built solutions for generating the essential text and image embeddings. Instead, I embraced open-source models from the Hugging Face library. These models are pre-trained on massive datasets, allowing them to effectively capture the semantic meaning of text and images. By integrating these models with my Flask API, I empowered the system to generate high-quality embeddings for the search process.

This approach not only ensures cost-effectiveness but also provides greater control and customization over the functionalities. Figures 7 and 8 likely depict the server environment where the Flask API is deployed and operational, enabling users to interact with the RAG search system.



```
660c1916210945aa74cd89d5
Processed
660c1916210945aa74cd89d6
Processed
660c1916210945aa74cd89d7
Processed
660c1916210945aa74cd89d8
Processed
660c1916210945aa74cd89d9
Processed
660c1916210945aa74cd89da
Processed
660c1916210945aa74cd89db
Processed
660c1916210945aa74cd89dc
Processed
660c1916210945aa74cd89dd
Processed
660c1916210945aa74cd89de
Processed
660c1916210945aa74cd89df
Processed
660c1916210945aa74cd89e0
Processed
660c1916210945aa74cd89e1
Processed
660c1916210945aa74cd89e2
Processed
660c1916210945aa74cd89e3
Processed
660c1916210945aa74cd89e4
Processed
660c1916210945aa74cd89e5
Processed
660c1916210945aa74cd89e6
Processed
660c1916210945aa74cd89e7
Processed
660c1916210945aa74cd89e8
Processed
660c1916210945aa74cd89e9
Processed
660c1916210945aa74cd89ea
Processed
660c1916210945aa74cd89eb
Processed
```

Figure 7 - Server image for the image tokenizer API

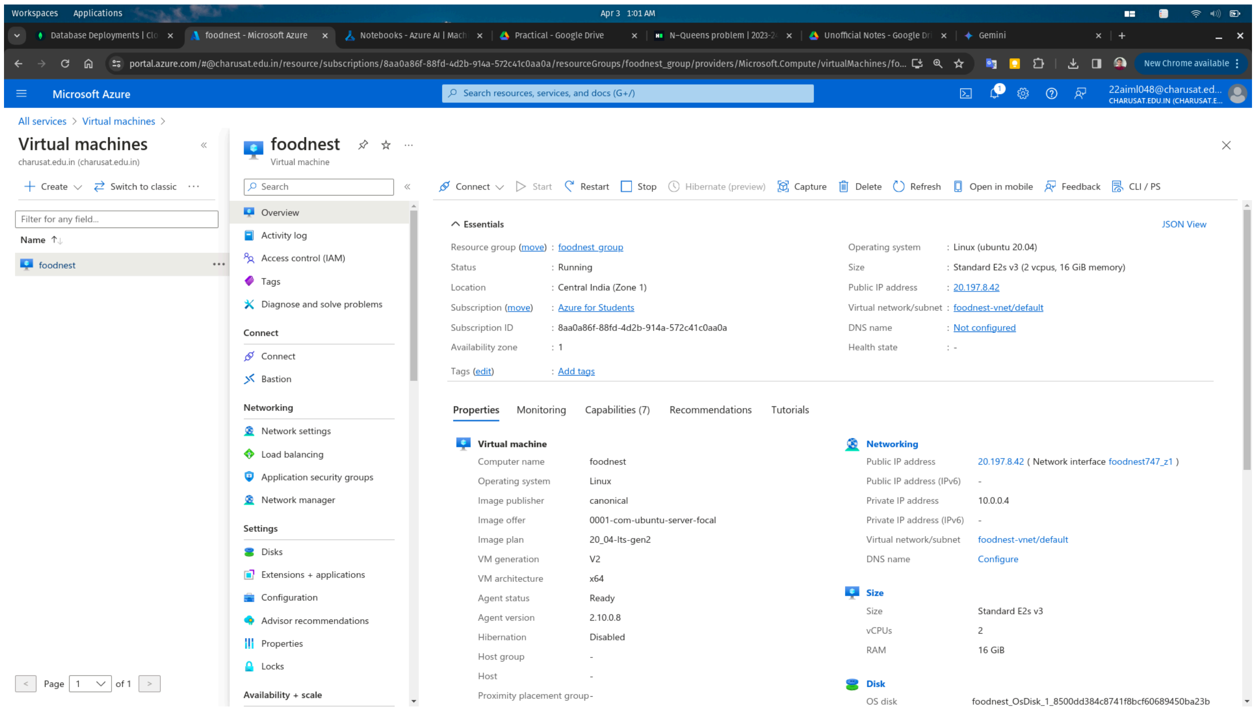

Workspaces
Applications
Apr 3 12:59 AM
azureuser@foodnest: ~/Food_App_RAG/Rag_VectorSearch_for_text/Tokenizer_Server

```
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:04:44] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:04:47] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:04:50] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:04:53] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:04:56] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:04:59] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:05:02] "POST /preprocess HTTP/1.1" 200 -
-----------------------
87.120.84.105 - - [02/Apr/2024 19:05:04] "CONNECT 45.61.137.126:7227 HTTP/1.1" 404 -
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:05:05] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:05:08] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:05:12] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
13.68.136.11 - - [02/Apr/2024 19:05:15] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
42.105.168.59 - - [02/Apr/2024 19:20:38] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
42.105.168.59 - - [02/Apr/2024 19:22:33] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
42.105.168.59 - - [02/Apr/2024 19:23:29] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
42.105.168.59 - - [02/Apr/2024 19:24:10] "POST /preprocess HTTP/1.1" 200 -
-----------------------
Processed a request successfully
42.105.168.59 - - [02/Apr/2024 19:26:08] "POST /preprocess HTTP/1.1" 200 -
```

Figure 7 - Server image for the text tokenizer API

To ensure the smooth operation of my server, efficient inference processing, and overall system management, I strategically utilized Microsoft Azure. Azure, a comprehensive cloud computing platform, provided the necessary infrastructure and tools to support the various aspects of my project.

While the specific details might not be explicitly stated, Figure 8 likely showcases the Azure workspace utilized in this project. This workspace would act as the central hub for managing all the Azure resources involved, including virtual machines, storage, and potentially machine learning services used for inference.



Essentials

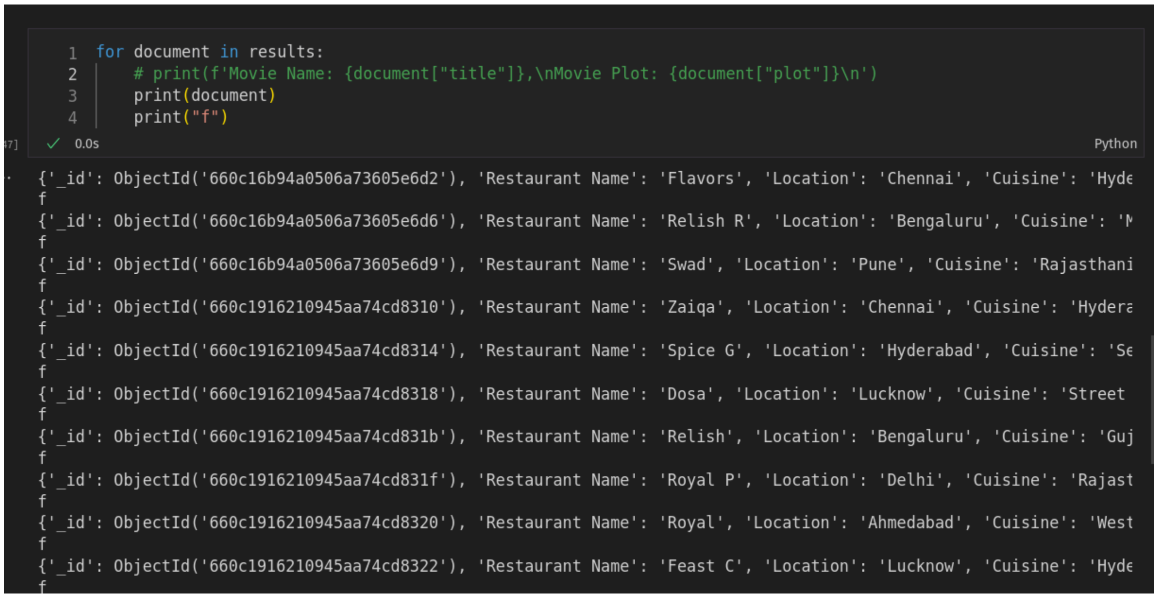
Resource group (move)	: foodnest_group	Operating system	: Linux (ubuntu 20.04)
Status	: Running	Size	: Standard E2s v3 (2 vcpus, 16 GiB memory)
Location	: Central India (Zone 1)	Public IP address	: 20.197.8.42
Subscription (move)	: Azure for Students	Virtual network/subnet	: foodnest-vnet/default
Subscription ID	: 8aa0a86f-88fd-4d2b-914a-572c41c0aa0a	DNS name	: Not configured
Availability zone	: 1	Health state	: -
Tags (edit)	: Add tags		

Properties Monitoring Capabilities (7) Recommendations Tutorials

Virtual machine		Networking	
Computer name	foodnest	Public IP address	20.197.8.42 (Network interface foodnest747_z1)
Operating system	Linux	Public IP address (IPv6)	-
Image publisher	canonical	Private IP address	10.0.0.4
Image offer	0001-com-ubuntu-server-focal	Private IP address (IPv6)	-
Image plan	20_04-lts-gen2	Virtual network/subnet	foodnest-vnet/default
VM generation	V2	DNS name	Configure
VM architecture	x64	**Size**	
Agent status	Ready	Size	Standard E2s v3
Agent version	2.10.0.8	vCPUs	2
Hibernation	Disabled	RAM	16 GiB
Host group	-	**Disk**	
Host	-	OS disk	foodnest_OsDisk_1_8500dd384c8741f8bcf60689450ba23b
Proximity placement group	-		

Figure 8 - Azure workspace

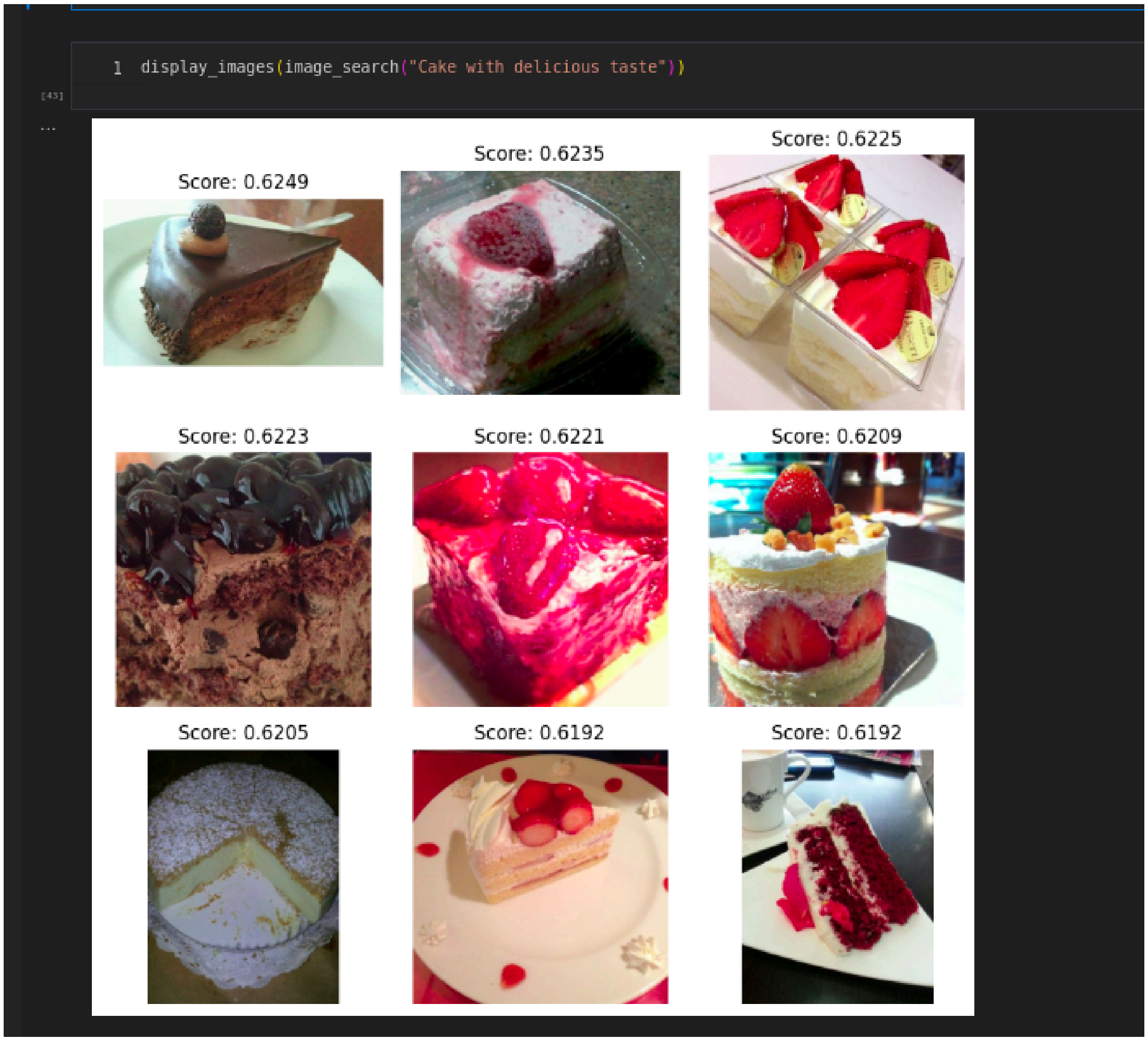
The culmination of these efforts is showcased in Figures 9 and 10, which likely depict the successful retrieval of relevant information through the RAG text-based search. These figures provide valuable insight into the system's effectiveness in understanding user queries and delivering pertinent results.



```python
for document in results:
    # print(f'Movie Name: {document["title"]},\nMovie Plot: {document["plot"]}\n')
    print(document)
    print("f")
```

```
{'_id': ObjectId('660c16b94a0506a73605e6d2'), 'Restaurant Name': 'Flavors', 'Location': 'Chennai', 'Cuisine': 'Hyde
f
{'_id': ObjectId('660c16b94a0506a73605e6d6'), 'Restaurant Name': 'Relish R', 'Location': 'Bengaluru', 'Cuisine': 'M
f
{'_id': ObjectId('660c16b94a0506a73605e6d9'), 'Restaurant Name': 'Swad', 'Location': 'Pune', 'Cuisine': 'Rajasthani
f
{'_id': ObjectId('660c1916210945aa74cd8310'), 'Restaurant Name': 'Zaiqa', 'Location': 'Chennai', 'Cuisine': 'Hydera
f
{'_id': ObjectId('660c1916210945aa74cd8314'), 'Restaurant Name': 'Spice G', 'Location': 'Hyderabad', 'Cuisine': 'Se
f
{'_id': ObjectId('660c1916210945aa74cd8318'), 'Restaurant Name': 'Dosa', 'Location': 'Lucknow', 'Cuisine': 'Street
f
{'_id': ObjectId('660c1916210945aa74cd831b'), 'Restaurant Name': 'Relish', 'Location': 'Bengaluru', 'Cuisine': 'Guj
f
{'_id': ObjectId('660c1916210945aa74cd831f'), 'Restaurant Name': 'Royal P', 'Location': 'Delhi', 'Cuisine': 'Rajast
f
{'_id': ObjectId('660c1916210945aa74cd8320'), 'Restaurant Name': 'Royal', 'Location': 'Ahmedabad', 'Cuisine': 'West
f
{'_id': ObjectId('660c1916210945aa74cd8322'), 'Restaurant Name': 'Feast C', 'Location': 'Lucknow', 'Cuisine': 'Hyde
f
```

Figure 9 - RAG based text inference



```
display_images(image_search("Cake with delicious taste"))
```

Figure 10 - RAG based image inference

Conclusion

In conclusion, this project successfully implemented a text and image search system utilizing Ranked Answer Grouping (RAG). A key strength lies in the strategic use of open-source technologies. MongoDB provided a flexible foundation for data storage and vectorization, while Flask facilitated the creation of a user-friendly search interface. Additionally, leveraging open-source models from Hugging Face for embedding generation ensured cost-effectiveness and customization.

Furthermore, the project demonstrates the value of cloud computing platforms like Microsoft Azure. Azure's scalable infrastructure likely ensured smooth system operation under varying workloads. Its suite of machine learning services might have streamlined the inference process within the RAG search system, while robust monitoring tools facilitated proactive system management.

The successful retrieval of relevant information through the RAG text-based search, as showcased in Figures 9 and 10, validates the project's effectiveness. This system offers a promising approach for users seeking to navigate large datasets of text and image information. Future advancements could involve exploring additional functionalities or expanding the search capabilities to encompass different data modalities.

Github project link: https://github.com/NeelDevenShah/Food_App_RAG