

ShopChat App

A Complete Guide and Solution Document

ShopChat App Document Contents

1	<i>Pre-requisites</i>	5
2	<i>RAG Architecture</i>	5
2.1	Tools and models used	5
2.2	LLM Performance Evaluation (Latency)	5
2.3	Finetune Model	6
2.4	Architecture Diagram	7
2.5	Inference UI (Prototype)	7
2.6	Feedback Loop	8
2.7	Improvements and Performance Enhancements	8
3	<i>Initial Setups</i>	8
3.1	Project Folder	8
3.2	Load project in VS Code	9
3.3	Create AWS user and add IAM access	9
3.4	Create AWS API keys for the user	10
3.5	Create AWS S3 bucket	11
3.6	Create Google Generative API key	11
3.7	Create OpenAI API key	12
3.8	Create Virtual environment and install dependencies	12
3.9	Update environment key and variables	13
4	<i>Inference ShopChat App</i>	14
4.1	Run Locally	15
4.1.1	Run Streamlit app	15
4.1.2	Inference ShopChat app	15
4.2	Run using Docker	16
4.2.1	Start Docker desktop	16
4.2.2	Build and Run Docker Image	17
4.2.3	Inference ShopChat app	18
5	<i>Data Loading and Preprocessing</i>	19
5.1	Data Ingestion	19
5.1.1	Ingestion Processes	19
5.1.2	Running data ingestion	19
5.2	<i>Data Captioning [Optional Step]</i>	20
5.2.1	Captioning Processes	21
5.2.2	Running data captioning	21

5.3 Data Embedding	22
5.3.1 Embedding Processes	22
5.3.2 Running Google Embedding	22
5.3.3 Running OpenAI Embedding	23
6 Full inferencing after data loading	25
6.1 Upload Your S3	25
6.2 Selection of LLM model	25
6.3 Display more images	26
6.4 Item not found	27
6.5 Listing Link	27
6.6 Feedback	28
6.7 Home button	28
7 CICD with GitHub Action and AWS	29
7.1 Create Git Repository	29
7.2 Create AWS ECR Repository	30
7.3 Create AWS EC2 Instance	31
7.4 Create IAM Role and add to EC2 instance	34
7.5 Install Docker into EC2 instance	37
7.6 Add Secrets to GitHub Actions	38
7.7 Add Runner	39
7.8 Add Workflow yml to GitHub repository	41
7.9 CICD Pipeline Triggers	43
7.10 Open port 8501 in EC2	45
7.11 ShopChat App inference from AWS using public URL	47
8 EDA and Data Preparation	49
9 Supporting scripts and files	50
9.1 env_variables.txt	50
9.2 exception.py	50
9.3 logging.py	51
9.4 utils.py	51
9.5 Dockerfile	52
9.6 .dockerignore	52
9.7 .gitignore	53

9.8	setup.py	54
9.9	requirements.txt	54
10	<i>Observations and Issues</i>	55
10.1	Finetune Embedding and LLM models	55
10.2	Inference with OpenAI LLM model	56
10.3	Response Thruput	56
10.4	Bucket selection	56
10.5	Image Caption	56
11	<i>Challenges</i>	57
11.1	Data Handling	57
11.2	Language and Localization	57
11.3	Integration of AI Models	57
11.4	Scalability and Performance	57
11.5	User Experience Design	58
12	<i>Future Improvements</i>	58
12.1	Model Enhancement	58
12.2	Multi-language Support	58
12.3	Personalization	58
12.4	Visual Search Capabilities	58
12.5	Analytics and Feedback Loop	58

1 Pre-requisites

- GitHub account
- VS Code installed and setup in your local system
- Docker desktop client installed in your local system
- AWS account with access to S3, EC2 and ECR
- Google Gen AI account and API key
- OpenAI API account and API key

2 RAG Architecture

RAG based application is created from scratch using Amazon Barkley Object open-source Database.

2.1 Tools and models used

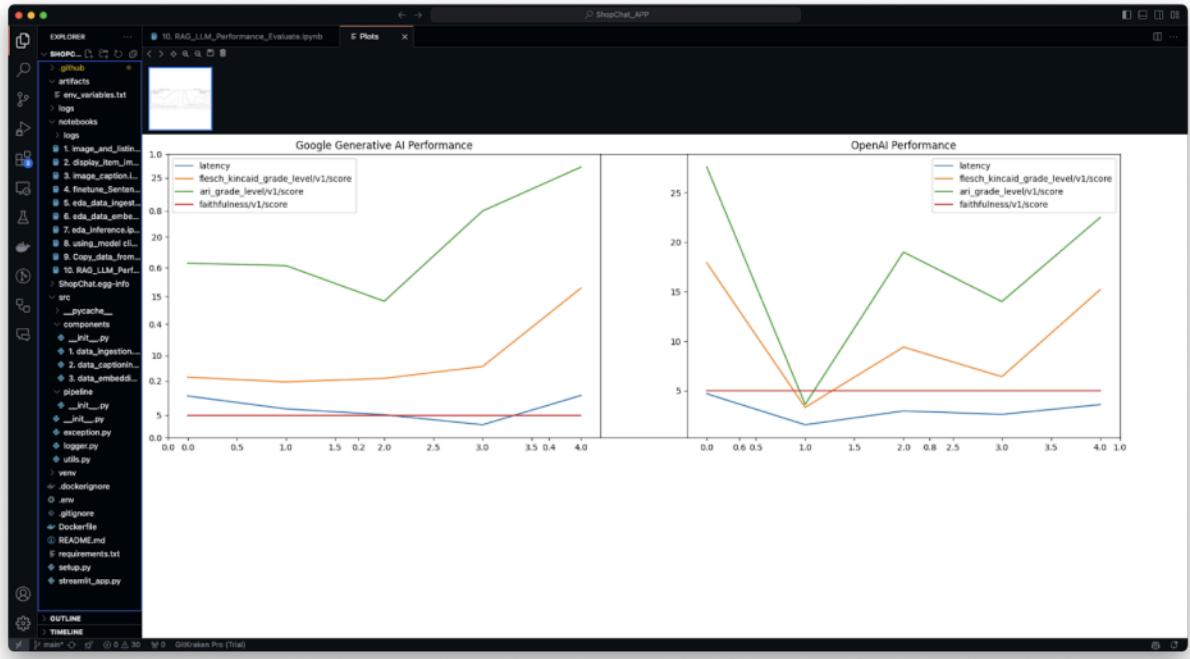
Following tools and models are used in the RAG architecture

- Streamlit: is used to design inference UI
- AWS S3: storing data and vector database
- AWS ECR: repository to save application docker image
- AWS EC2: web service compute capacity to run the application
- Docker Container: containerize the application, environment
- GitHub Action: used to create CICD pipeline
- FAISS: vector store DB to store generated embedding indexes
- LangChain: framework/tools used for building applications that leverage LLM
- Embedding models used are:
 - GoogleGenerativeAIEmbeddings: For Google's embedding model
 - OpenAIEMBEDDINGS: For OpenAI's model
 - HuggingFaceEmbeddings: For a fine-tuned model loaded from S3
- LLM models used:
 - Google Gemini-pro
 - OpenAI 3.5 turbo
 - Finetune Model

2.2 LLM Performance Evaluation (Latency)

Using mlflow evaluate function, we measured the faithfulness performance metrics. We had taken 5 simple and common questions, ran them thru the Langchain LLM chain and retriever to get the answers and stuffed documents retrieved from FAISS vector DB.

We then pass them to mlflow.evaluate() function to get the faithfulness_metric. Source code can be found in: **10. RAG_LLM_Performance_Evaluate.ipynb** notebook



Observation: Although latency for OpenAI model is less, there are lot of variances on other metrices. Only evaluated with 5 questions. We can try with more questions and get the mean for all metrices for better quality

2.3 Finetune Model

We finetuned sentence transformer model “BAAI/bge-small-en” using synthetic questions generated from OpenAI GPT-3 model. We used 100 Json datapoint to create questions from GPT-3. These 100 questions are then split into (80%) train and (20%) validation. We finetuned sentence transformer model with test questions dataset and evaluate the performance against validation data.

Compared the Hit rate with OpenAI and non-finetuned sentence transformer “BAAI/bge-small-en” model. The compared results were not so impressive

model	epoch	steps	cos_sim-Accuracy@1	cos_sim-Accuracy@3	cos_sim-Accuracy@5	cos_sim-Accuracy@10	cos_sim-Precision@1	cos_sim-Recall@1	cos_sim-Precision@3	cos_sim-Recall@3	...	dot_score-Recall@1
bge	-1	-1	0.85	1.0	1.0	1.0	0.85	0.85	0.333333	1.0	...	0.85
fine_tuned	-1	-1	0.85	1.0	1.0	1.0	0.85	0.85	0.333333	1.0	...	0.85

Observation: Synthetic questions generated by OpenAI model were not so buyer friendly and hence the finetuned model did not perform well on inference.

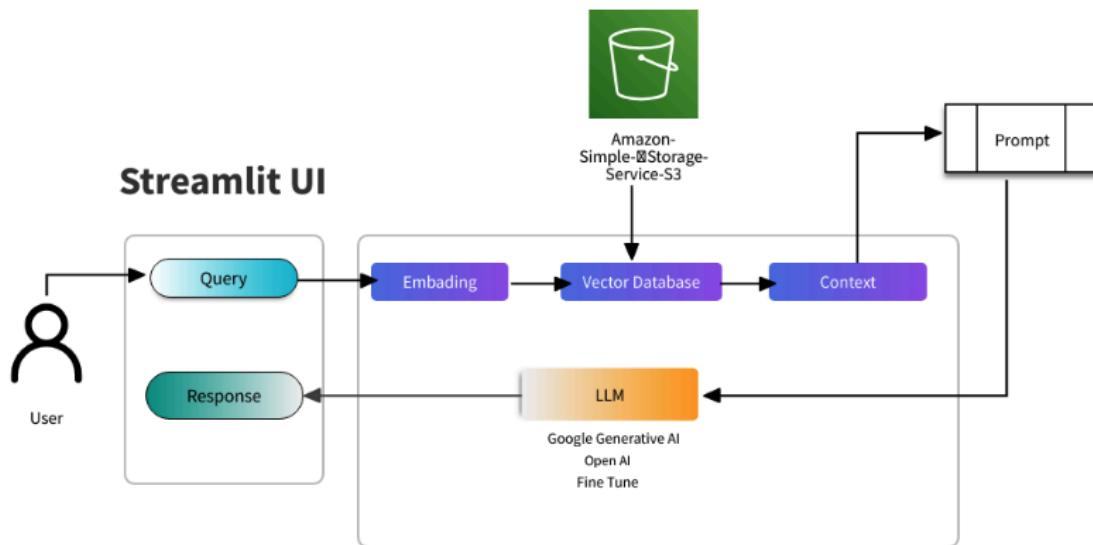
Improvements can be done on finetune:

1. Create better questions and answers from the ABO data using better prompt
2. Train for more epochs and more question-and-answer dataset

3. Question-and-answer generation model can improve performance
4. Fine Model selection can also improve on performance

Finetuning Notebook: 4. finetune_SentenceTransformers_embedding_model.ipynb

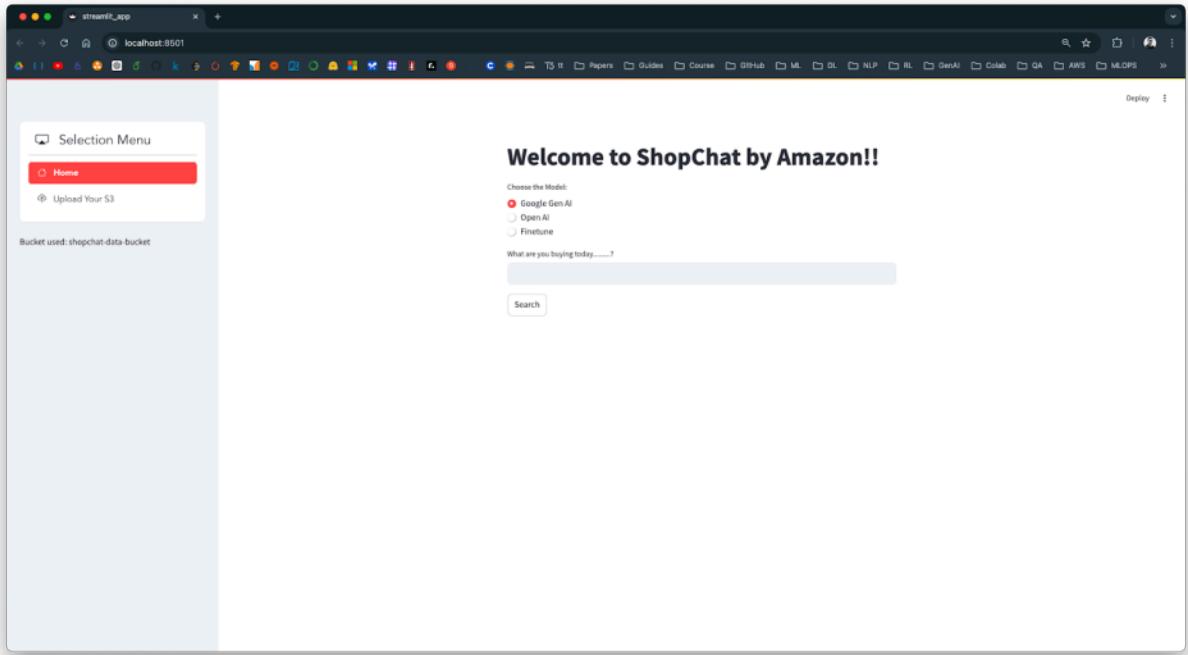
2.4 Architecture Diagram



2.5 Inference UI (Prototype)

A simple Streamlit application is created to take the user questions and provide response. There are options to select different LLM models and data source to run the application.

- User Input: Query received through Streamlit UI.
- Model Selection: Choose between Google Gen AI, OpenAI, or Finetuned models.
- Response Generation: Using selected LLM to process the query and generate recommendations.
- Image Retrieval: Display relevant images from ABO S3 using Python Pillow library
- Displaying Results: Showing recommendations and images in the Streamlit UI.



2.6 Feedback Loop

Feedback taking mechanism is created within the Streamlit app where user can like and dislike a product, share the feedback comments/suggestion. This is enabled for every item that LLM returns in respond.

We have not done anything with the feedback received from the UI, there is a scope to improve the quality LLM response by implementing feedback received from the user

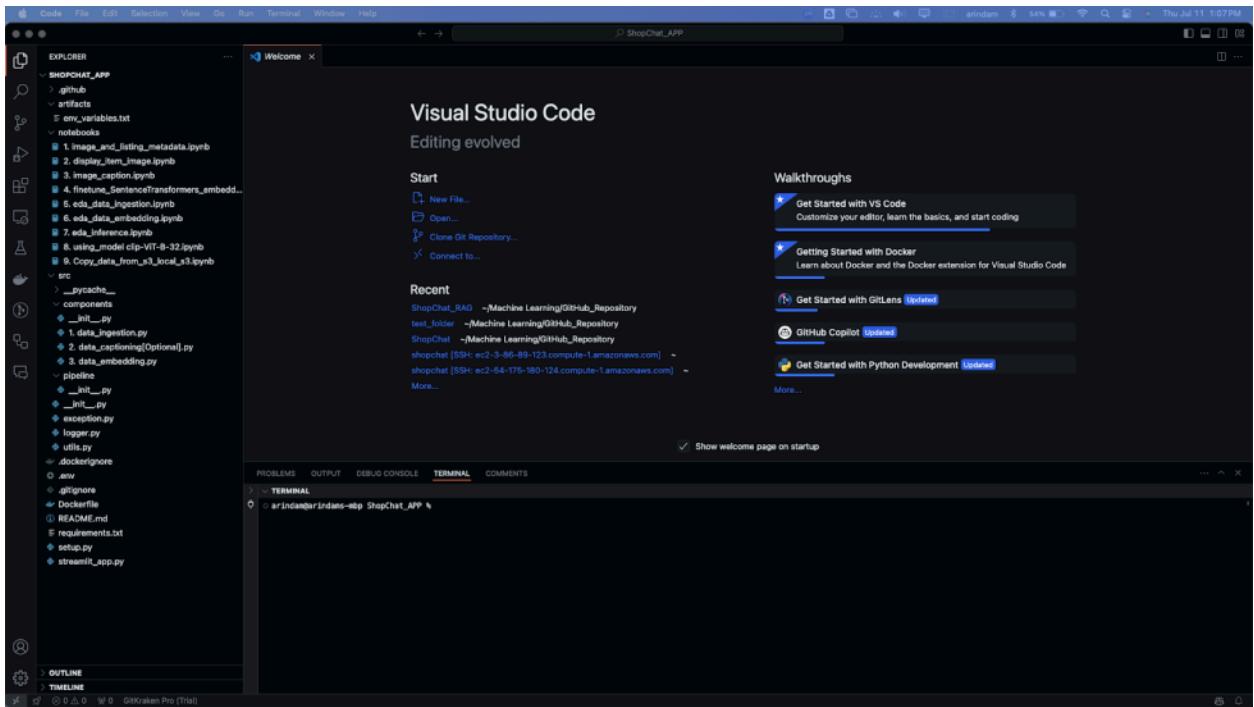
2.7 Improvements and Performance Enhancements

- Optimize Vector store and Indexing: Ensure efficient indexing and retrieval.
- Caching: Implement caching mechanisms for frequently accessed data.
- Batch Processing: Handle bulk queries to improve throughput.
- Parallel Processing: Utilize parallel processing for downloading.
- Fine tuning: Both Embedding and LLM models can give better results.
- Feedback mechanism: To retrieve right results.

3 Initial Setups

3.1 Load project in VS Code

Get the project from Github: https://github.com/arindam-29/ShopChat_APP



3.2 Create AWS user and add IAM access

Create a new user in IAM and attach the following policies directly:

- AmazonEC2ContainerRegistryFullAccess
- AmazonEC2FullAccess
- AmazonS3FullAccess

Name	Type	Used as
AmazonEC2ContainerRegistryFullAccess	AWS managed	Permissions policy
AmazonEC2FullAccess	AWS managed	Permissions policy
AmazonS3FullAccess	AWS managed	Permissions policy

3.3 Create AWS API keys for the user

Once the new user is created, click the username to go to the IAM access page. Under security credential create access keys (you can select use case as Other in next page) and download the access key

The screenshot shows the AWS IAM User Details page for a user named 'ShopChat'. The left sidebar shows navigation options like Dashboard, Access management, and Access reports. The main content area displays the user's ARN (arn:aws:iam::183651425911:user/ShopChat), which has 'Console access' disabled. It also shows the user was created on July 09, 2024, at 20:37 UTC-04:00. Two access keys are listed: 'Access key 1' (AKIASVQT2LZ5VJSGXCQ5 - Active, Never used. Created today) and 'Access key 2' (Create access key). Below this, there are sections for 'Console sign-in' (with a link to https://183651425911.signin.aws.amazon.com/console) and 'Multi-factor authentication (MFA) (0)'.

3.4 Create AWS S3 bucket

Create a new S3 bucket to save all data that will get created during execution

The screenshot shows the AWS S3 console interface. On the left, there's a sidebar with options like Buckets, Access Grants, Access Points, Object Lambda Access Points, Multi-Region Access Points, Batch Operations, IAM Access Analyzer for S3, Block Public Access settings, Storage Lens, Dashboards, Storage Lens groups, AWS Organizations settings, Feature spotlight, and AWS Marketplace for S3. The main area is titled 'Amazon S3 > Buckets'. It features a banner for 'Account snapshot - updated every 24 hours' and a 'View Storage Lens dashboard' button. Below this, there are tabs for 'General purpose buckets' (selected) and 'Directory buckets'. A search bar says 'Find buckets by name'. A table lists one bucket: 'shopchat-s3-bucket' (Name), 'US East (N. Virginia) us-east-1' (AWS Region), 'View analyzer for us-east-1' (IAM Access Analyzer), and 'July 10, 2024, 16:28:11 (UTC-04:00)' (Creation date). There are buttons for 'Copy ARN', 'Empty', 'Delete', and 'Create bucket'.

3.5 Create Google Generative API key

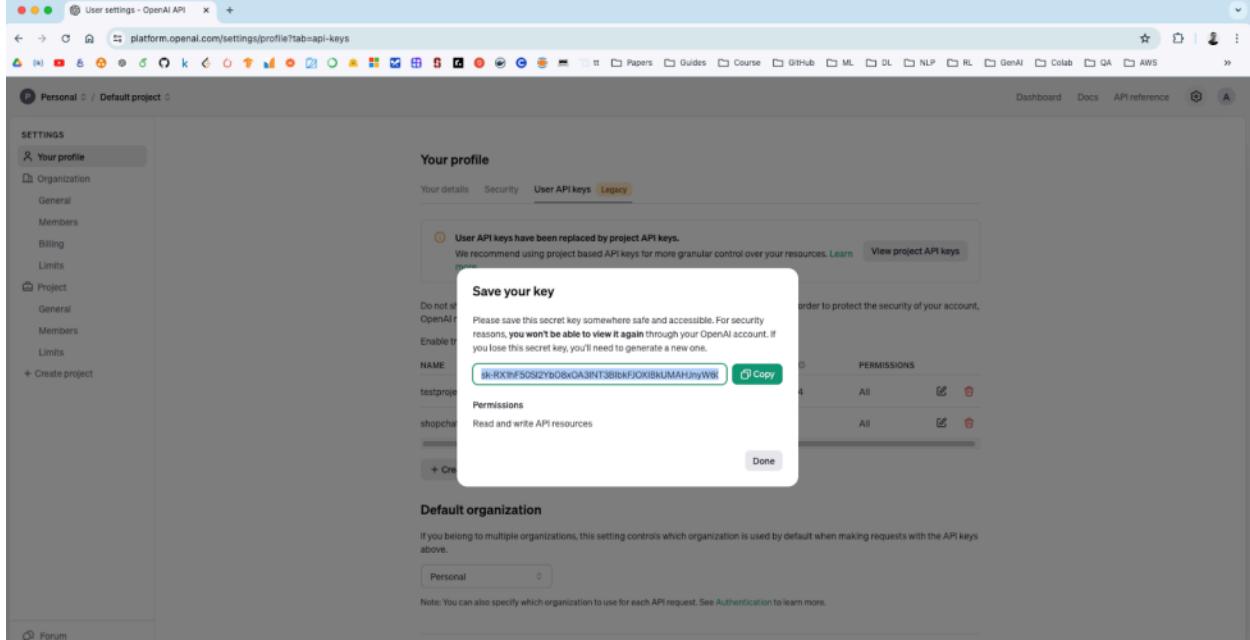
- Go to the link: <https://aistudio.google.com/app/apikey>
- login / create account with google generative AI
- Create an API key. **Save** the key, we will add them in .env file in next step

The screenshot shows the Google AI Studio interface. On the left, there's a sidebar with links like Get API key, Create new prompt, New tuned model, My library, Allow Drive access, Getting started, Documentation, Prompt gallery, Gemini cookbook, Discourse forum, and Build with Vertex AI on Google Cloud. The main area is titled 'Get API key'. It has a sub-section 'API keys' with a note about creating a new project. A modal window titled 'API key generated' shows the generated key: 'AlzaSyDWMpeJGPzcfPy9wsefEvECWevc-Y8Cm8'. Below it, there's a 'Copy' button. The page also includes sections for 'Plan', 'Free of charge', 'Set up Billing', and 'View usage data'. At the bottom, there's a code block for 'Quickly test the API by running a cURL command':

```
curl \
  -H "Content-Type: application/json" \
  -d '{"contents": [{"text": "Explain how AI works"}]}' \
  -X POST 'https://generativelanguage.googleapis.com/vbeta/models/gemini-1.5-flash-latest:generateContent?key=YOUR_API_KEY'
```

3.6 Create OpenAI API key

- Go to the link: <https://platform.openai.com/settings/profile?tab=api-keys>
- login / create account with OpenAI
- Create an API key. Save the key, we will add it in .env file in next step



3.7 Create Virtual environment and install dependencies

Setup the virtual environment for running this project. We need **Python 3.9** for this. If your machine has python 3.9 then you can directly create virtual environment or may use conda or other tools to create the environment with python 3.9. We will use python venv in this project since my machine has Python 3.9 and install dependencies (requirements.txt) using following commands

- a. `python3 -m venv venv`
- b. `source venv/bin/activate`
- c. `pip3 install -r requirements.txt`

```

requirements.txt
1 pillow==10.3.0
2 transformers==4.41.2
3 sentence_transformers==2.7.0
4 pandas==2.2.2
5 matplotlib==3.9.0
6 streamlit==1.36.0
7 streamlit_feedback==0.1.3
8 streamlit_option_menu==0.3.13
9 google-generativeai>=0.5.2
10 python-dotenv==1.0.1
11 langchain==0.2.7
12 pydub==4.2.0
13 chromadb==0.5.3
14 faiss-cpu==1.8.0
15 langchain_google_genai
16 langchain_openai==0.1.14
17 langchain_community==0.2.7
18 langchain_huggingface==0.0.3
19 langchain_groq==0.1.6
20 jg==1.7.0
21 numpy==1.26.4
22 matplotlib==3.9.0
23 s3fs==2024.6.1
24 sagemaker==2.224.4
25 llama_index_finetuning==0.1.10
26 llama_index==0.10.53
27 llama_index_embeddings_huggingface==0.2.2
28 huggingface_hub==0.23.4
29 scikit-learn==1.5.1
30 boto3==1.34.131

```

Once the requirements get installed in virtual environment, setup tool creates the Info folder about the project

```

ShopChat_APP > requirements.txt
1 pillow==10.3.0
2 transformers==4.41.2
3 sentence_transformers==2.7.0
4 pandas==2.2.2
5 matplotlib==3.9.0
6 streamlit==1.36.0
7 streamlit_feedback==0.1.3
8 streamlit_option_menu==0.3.13
9 google-generativeai>=0.5.2
10 python-dotenv==1.0.1
11 langchain==0.2.7
12 pydub==4.2.0
13 chromadb==0.5.3
14 faiss-cpu==1.8.0
15 langchain_google_genai
16 langchain_openai==0.1.14
17 langchain_community==0.2.7
18 langchain_huggingface==0.0.3
19 langchain_groq==0.1.6
20 jg==1.7.0
21 numpy==1.26.4
22 matplotlib==3.9.0
23 s3fs==2024.6.1
24 sagemaker==2.224.4
25 llama_index_finetuning==0.1.10
26 llama_index==0.10.53
27 llama_index_embeddings_huggingface==0.2.2
28 huggingface_hub==0.23.4
29 scikit-learn==1.5.1
30 boto3==1.34.131

ShopChat_APP > dependency_links.txt
ShopChat_APP > SOURCES.txt
ShopChat_APP > PKG-INFO
1 Metadata-Version: 2.1
2 Name: ShopChat
3 Version: 1.0
4 Summary: A RAG based shopping assistant
5 Home-page: UNKNOWN
6 Author: Arindam Choudhury
7 Author-email: arindam.choudhury@email.com
8 License: UNKNOWN
9 Platform: UNKNOWN
10 UNKNOWN
11 UNKNOWN
12 UNKNOWN
13 UNKNOWN
14 UNKNOWN
15 UNKNOWN
16 UNKNOWN

```

3.8 Update environment key and variables

- Go to the file `.env` and add all the keys saved from previous steps
- Add your S3 bucket name in `YOUR_S3_BUCKET_NAME` variable
- Leave all other variables unchanged as described

```

ShopChat_APP
└── .env
    └── .env
        1 export AWS_ACCESS_KEY_ID = "AKIASVOT2LZ3XX4VE6Q0A"
        2 export AWS_SECRET_ACCESS_KEY = "3RwKg1+0K/CBAcHzzC1x6g5dGR8qlx60UnrCcrlU"
        3 export AWS_DEFAULT_REGION = "us-east-1"
        4 GOOGLE_API_KEY = "AIzaSyDmPeJGPzaziFy9we8eviECWEvC-Y8Cm8"
        5 OPENAI_API_KEY = "sk-proj-duZAuIETa0YaLfxJM7cXT3BlbkFJ9BLLOnPbhN9kl087Pn"
        6 YOUR_S3_BUCKET_NAME = "shopchat-s3-bucket"
        7 ## Unchanged Variables (DO NOT CHANGE) ##
        8 SHOPCHAT_BUCKET_NAME = "shopchat-data-bucket"
        9 ARTIFACTS_BUCKET_NAME = "amazon-berkeley-objects"
        10 ARTIFACTS_FOLDER = ".../artifacts"
        11 WORKING_DIR = ".../artifacts/downloads/"
        12 EDA_FOLDER_NAME = "EDA_FILES"

```

TERMINAL

```

arindam@arindam-MacBook-Pro:~/Desktop/ShopChat_APP$ python -m venv venv
arindam@arindam-MacBook-Pro:~/Desktop/ShopChat_APP$ source venv/bin/activate
(venv) arindam@arindam-MacBook-Pro:~/Desktop/ShopChat_APP$ pip3 install -r requirements.txt
Collecting pillow==8.3.0
  Using cached pillow-8.3.0-cp39-cp39-macosx_11_0_arm64.whl (3.4 MB)

```

4 Inference ShopChat App

We will discuss three approaches here to interact with ShopChat app. To run the app you **may not** need the data loading scripts to run first. Required data is already present in another public S3 bucket and can be accessed by any AWS user using any AWS secret key and access key. However, in the app you cannot upload data from your S3 bucket until you run the data loading process described later in this document. App uses following bucket during inference [listed as `SHOPCHAT_BUCKET_NAME = "shopchat-data-bucket"` in `.env` file]

Note: This bucket will be used by default inside the app

Name	Type	Last modified	Size	Storage class
EDA_FILES/	Folder	-	-	-
FINETUNE/	Folder	-	-	-
GOOGLE_FAISS_DB/	Folder	-	-	-
OPENAI_FAISS_DB/	Folder	-	-	-
PROCESSED/	Folder	-	-	-

4.1 Run Locally

4.1.1 Run Streamlit app

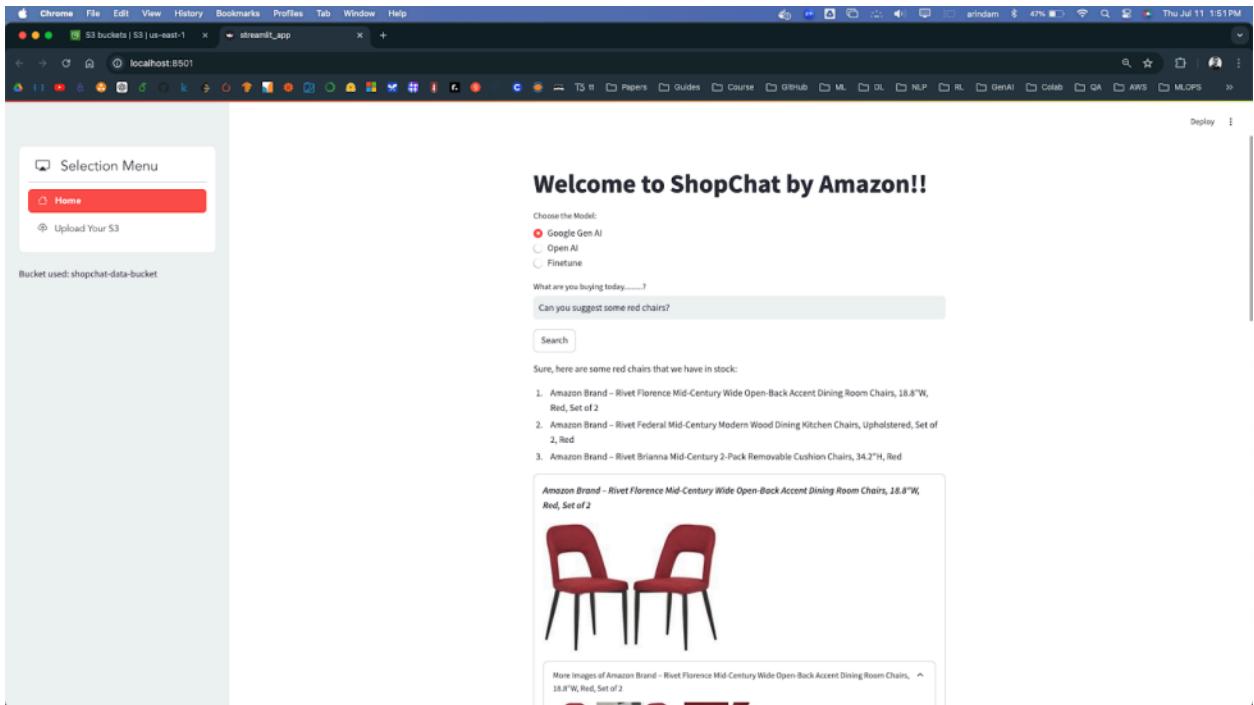
- Open a new terminal and type: **streamlit run streamlit_app.py**
- A new folder in artifacts/project will be created and necessary FAISS databases and Json dataset will be copied from project S3 bucket as mentioned above

The screenshot shows a code editor interface with the following details:

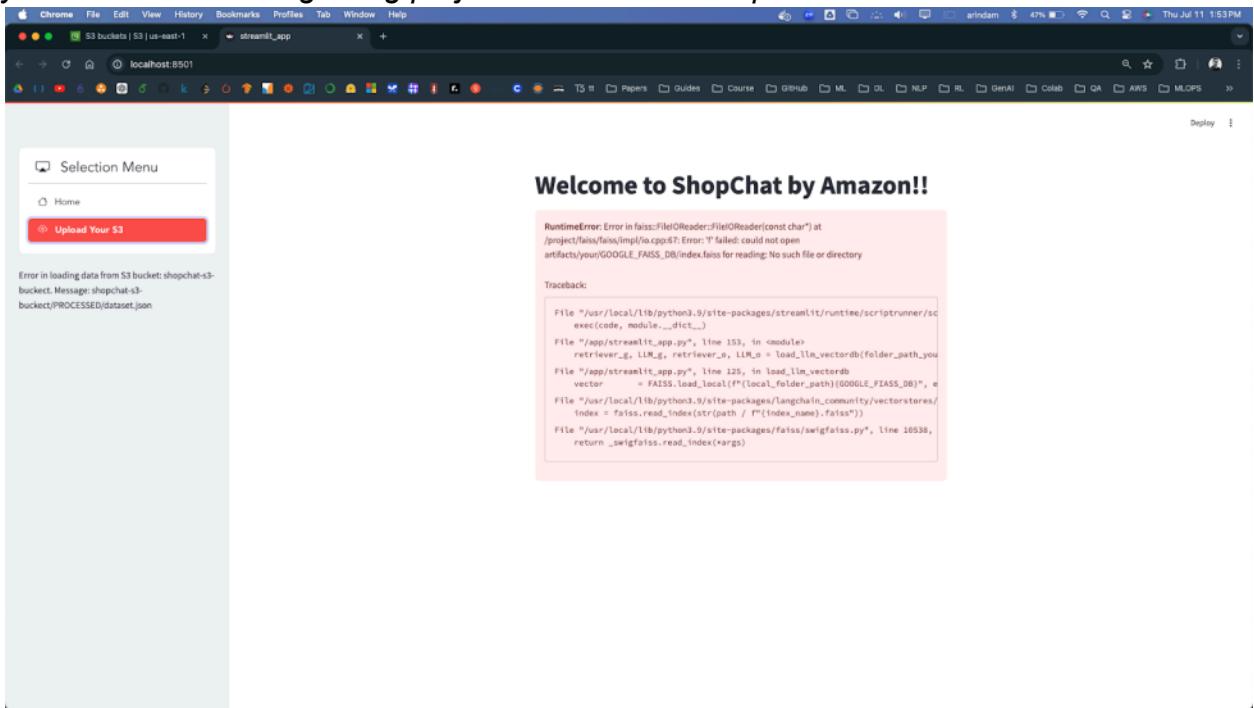
- Explorer View:** Shows the project structure under "SHOPCHAT_APP". The "artifacts" folder is highlighted with a red box.
- Code Editor:** The file "streamlit_app.py" is open, displaying Python code for a Streamlit application. The code imports various libraries like streamlit, google.generativeai, langchain, and langchain_community. It defines a function that uses these imports to handle image captioning and retrieval.
- Terminal:** The terminal window shows the command being run: `(venv) arindam@arindam-mbp ShopChat_APP % streamlit run streamlit_app.py`. Below the command, a message indicates the app is running locally at `Local URL: http://localhost:8501`.

4.1.2 Inference ShopChat app

Go to the Local URL: <http://localhost:8501> to interact with the app, ask questions and select different LLM models for your response



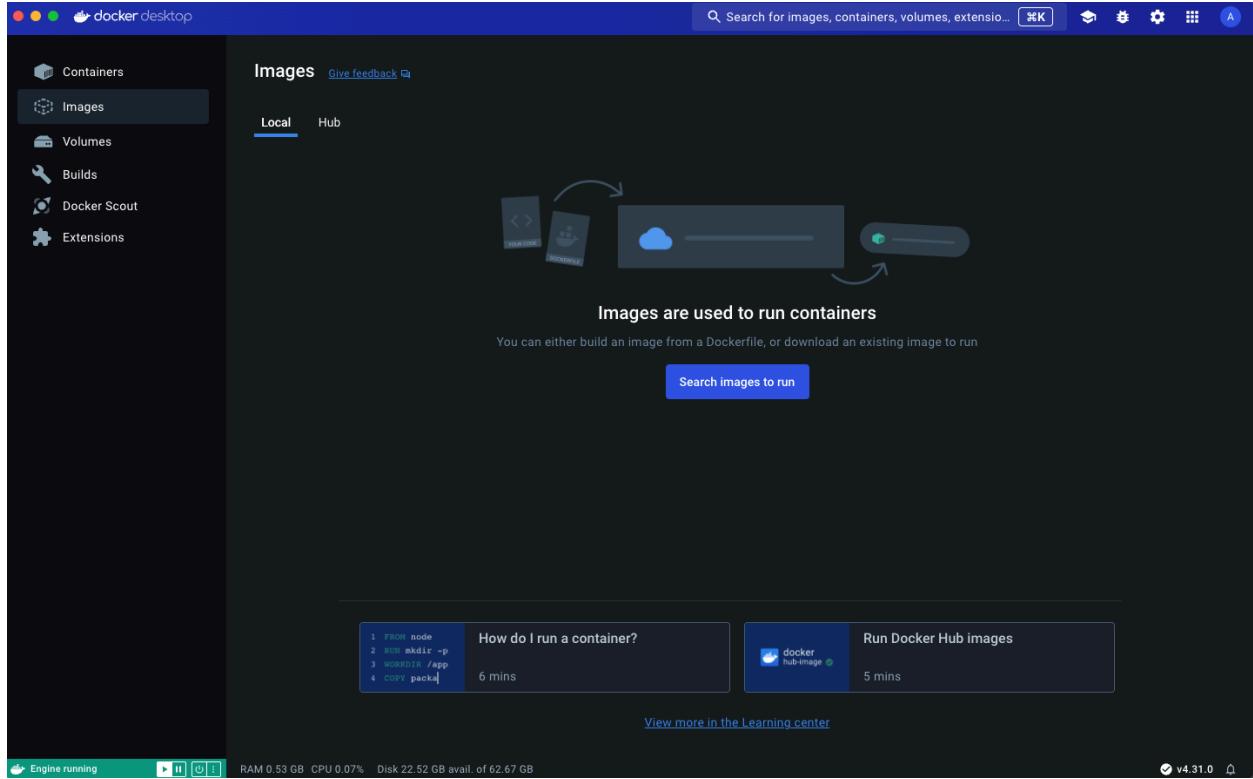
Note: If you click on Upload Your S3; this will give error as we have not created S3 yet but default running using project's S3 bucket "shopchat-data-bucket".



4.2 Run using Docker

4.2.1 Start Docker desktop

Start docker desktop client in your local system



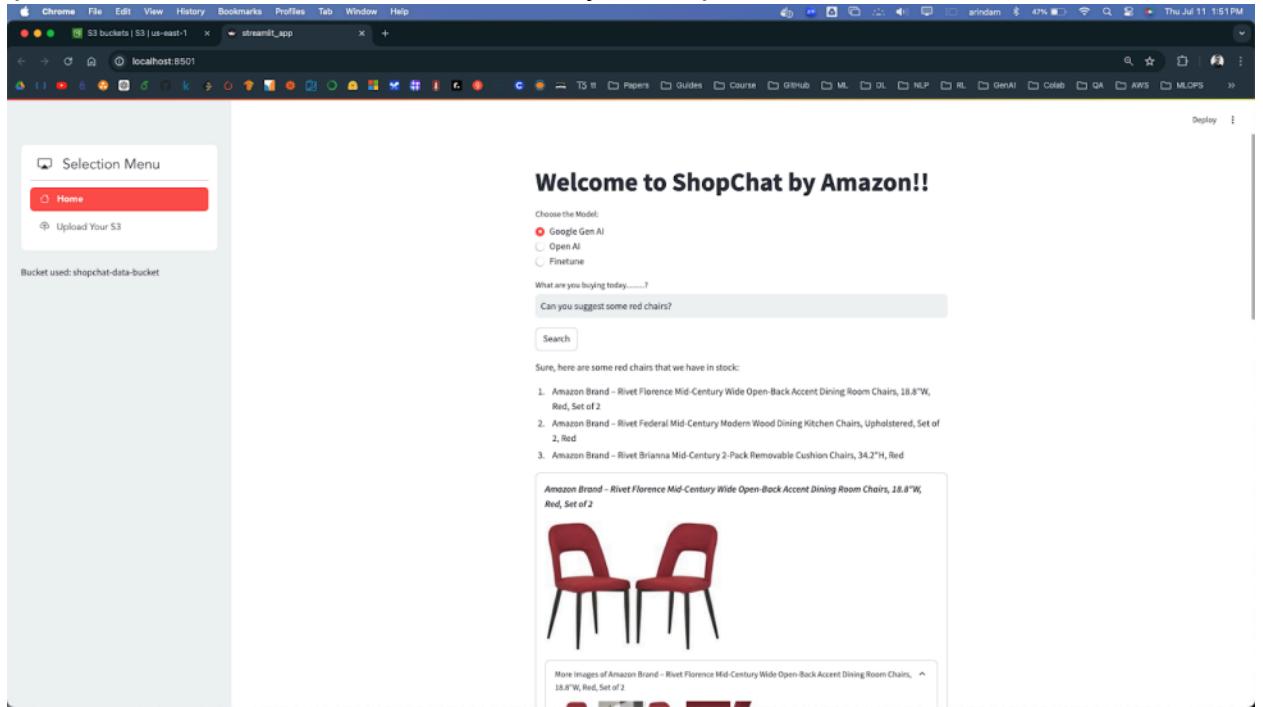
4.2.2 Build and Run Docker Image

Goto VS code and in a new terminal issue the following commands to build and run the docker image:

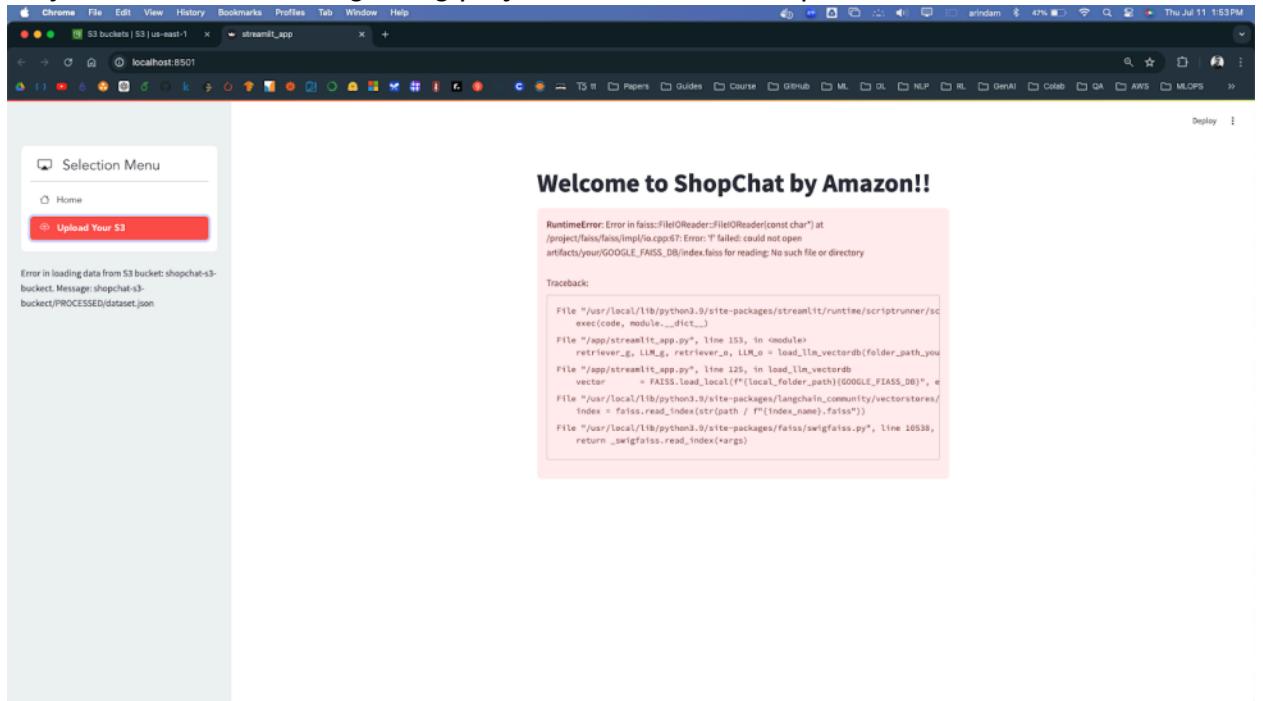
- docker build streamlit:1.0 .
 - docker run -p 8501:8501 streamlit:1.0

4.2.3 Inference ShopChat app

Go to the Local URL: <http://localhost:8501> to interact with the app to ask questions and select different models for your response



Note: If you click on Upload Your S3; this will give error as we have not created S3 yet but default running using project's S3 bucket “shopchat-data-bucket”.



5 Data Loading and Preprocessing

All created data in this process will be loaded to your S3 bucket created in earlier step [YOUR_S3_BUCKET_NAME = "shopchat-s3-bucket" as listed in .env file]

5.1 Data Ingestion

Script Name: "1.data_ingestion.py"

This script reads the data from ABO S3 bucket (amazon-berkeley-objects) <https://amazon-berkeley-objects.s3.amazonaws.com/index.html> process the data and create a Json file (dataset.json) and save it to your S3 bucket.

5.1.1 Ingestion Processes

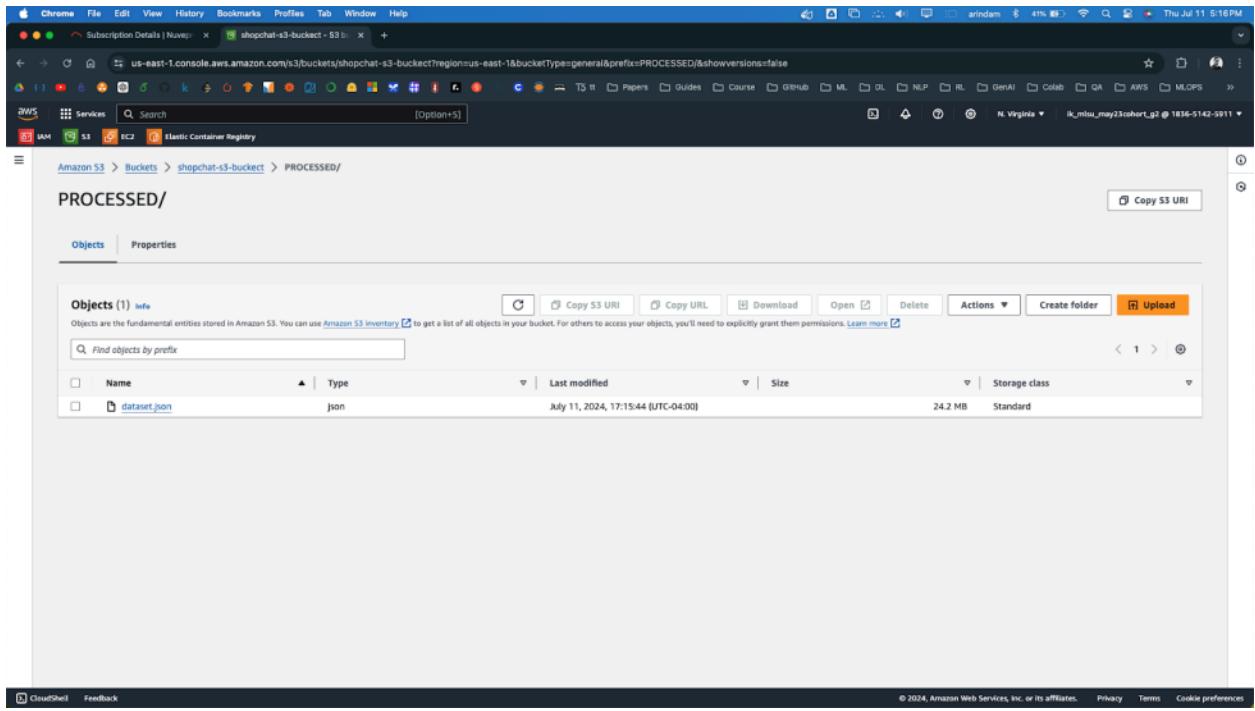
- Loads JSON files of product listings from ABO S3 bucket
- For each listings, extracts US_English metadata fields from the product listings.
- Loads image metadata from ABO S3 bucket
- Merges product listings metadata with image metadata to get the image paths for product image and additional image paths of the product
- Drops unnecessary columns and saves the processed dataset as Json file in project S3 bucket.

5.1.2 Running data ingestion

View the log file and S3 bucket. *Note: This script runs about 20-25 minutes*

```
logs > 07_11_2024_16_41_19.log > F 07_11_2024_16_41_19.log
1 [ 2024-07-11 16:41:20,323 ] 37 root - INFO - Data Ingestion - started
2 [ 2024-07-11 16:41:20,323 ] 40 root - INFO - Data Ingestion - file load
3 [ 2024-07-11 16:41:20,329 ] 567 aiobotocore.credentials - INFO - Found
4 [ 2024-07-11 16:41:39,477 ] 49 root - INFO - Data Ingestion - file loc
5 [ 2024-07-11 16:41:39,477 ] 59 root - INFO - Data Ingestion - processir
6 [ 2024-07-11 16:41:42,002 ] 77 root - INFO - Data Ingestion - processir
7 [ 2024-07-11 16:41:42,002 ] 81 root - INFO - Data Ingestion - processir
8 [ 2024-07-11 17:03:23,724 ] 98 root - INFO - Data Ingestion - processir
9 [ 2024-07-11 17:03:23,734 ] 108 root - INFO - Data Ingestion - dataset.
10 [ 2024-07-11 17:03:41,598 ] 113 root - INFO - Data Ingestion - complete
11
```

Entry in S3 in PROCESSED folder



5.2 Data Captioning [Optional Step]

Script Name: “2. data_captioning[Optional].py”

This script reads the Json dataset created in previous step from S3. For every item image captioning is done using SoTA Salesforce BLIP model.

Captioning each image takes time and is not a **Must Have feature** for embedding hence this script is made optional but vector search does perform well if added in text embedding

5.2.1 Captioning Processes

- Loads dataset JSON file created in ingestion process
- For each product, gets the image path
- Loads the images from ABO S3
- Sends the image to BLIP model for captioning
- Adds the caption back to the Json dataset.
- Saves back the Json file in project S3 bucket

5.2.2 Running data captioning

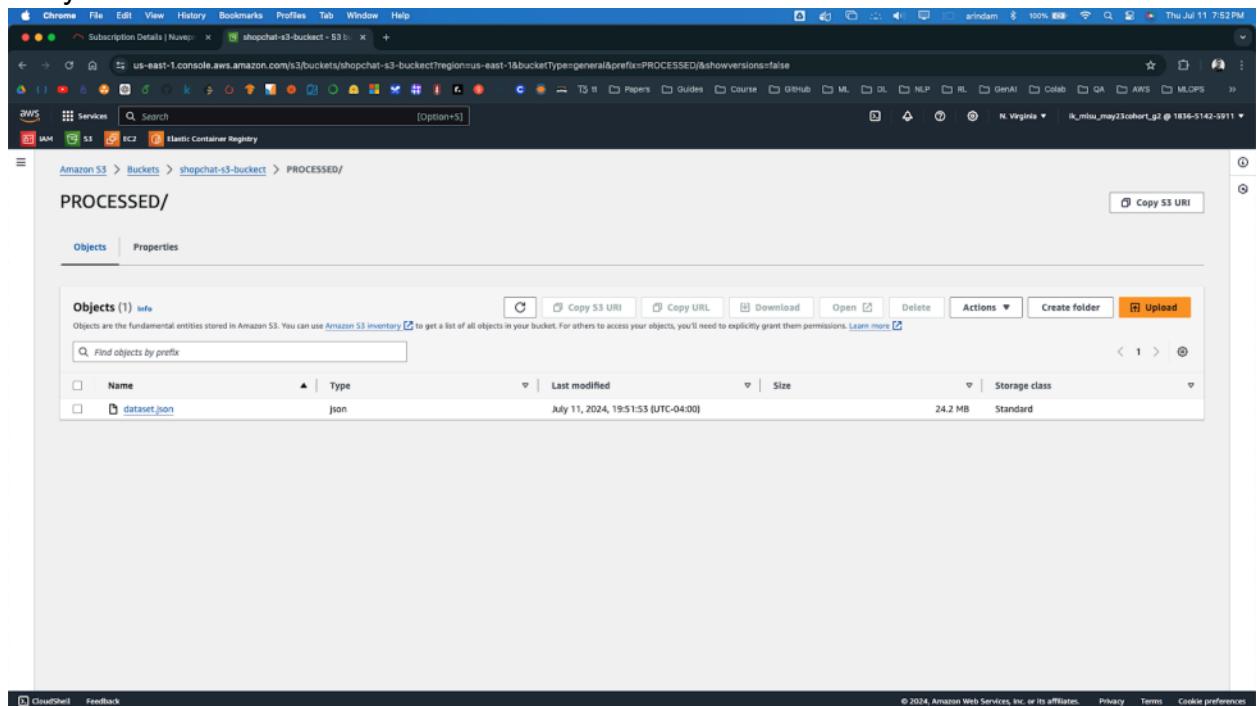
The caption is added back to the same Json dataset in your S3 bucket.

Note: This script runs for long time (more than 3 hours depending on the system config) and hence it is an optional script for this process. If used, then gives better results during inference thru LLM.

The screenshot shows a terminal window with the following content:

```
(venv) arindam@arindam-mbp ShopChat_APP % python3 src/components/2/_data_captioning[Optional].py
/users/arindam/Machine Learning/GitHub_Repository/ShopChat_APP/venv/lib/python3.9/site-packages/transformers/generation/utils.py:1188: UserWarning: Using the model-agnostic default "max_length" (>20) to control the maximum length. We recommend setting "max_new_tokens" to control the maximum length of the generation.
warnings.warn("Using the model-agnostic default "max_length" (>20) to control the maximum length. We recommend setting "max_new_tokens" to control the maximum length of the generation.", UserWarning)
```

Entry in S3 in PROCESSED folder



5.3 Data Embedding

Script Name: "3. data_embedding.py"

This script reads the Json dataset created in previous step from S3. Uses Langchain JSONLoader and Google embedding and(or) OpenAI embedding and save to FAISS vector database.

5.3.1 Embedding Processes

- Loads dataset.json file from project S3 bucket
 - Converts the Json list of objects into LangChain Documents objects using JSONLoader, which is compatible with the embedding generation.
 - Create Vector Store: Utilizes FAISS, a highly efficient library for approximate nearest neighbor search in the vector space, to create a vector store from the documents using the specified embedding model.
 - Saves the vector database to S3

Models used for Embedding are:

- GoogleGenerativeAIEmbeddings: For Google's embedding model.
 - OpenAIEMBEDDINGS: For OpenAI's model.
 - HuggingFaceEmbeddings: For a fine-tuned model loaded from S3

5.3.2 Running Google Embedding

We will be using GoogleGenerativeAIEmbedding model from Langchain google genai. Script is set to use “google” from the selection menu. *Note: This script runs about 20-25 minutes*

The screenshot shows a developer's workspace with multiple tabs open. The main editor tab displays Python code for a 'DataEmbedding' class, specifically the 'initiate_data_embedding' method. A red box highlights the line 'model = "google"'. Below the editor, the terminal window shows the command 'python3 src/components/3/\ data_embedding.py' being run, followed by its output: 'Uploaded GOOGLE_FAISS_DB/index. Tails to S3 bucket shopchat-s3-bucket'. The bottom status bar indicates the file is 0 bytes and the editor is using GitKraken Pro (Trial).

```
class DataEmbedding:
    def initiate_data_embedding(self):
        if os.path.exists(self.embedding_config):
            shutil.rmtree(self.embedding_config)
            print(f"Deleting {self.embedding_config}")
        logging.info("Data Embedding - config deleted")
    except Exception as e:
        raise CustomException(e, sys)

if __name__ == "__main__":
    #Select the model for embedding
    model = "google"

model = "openai"
#model = "finetune"
obj=DataEmbedding(model)
obj.initiate_data_embedding()
```

Entry in S3 GOOGLE_FAISS_DB folder

5.3.3 Running OpenAI Embedding

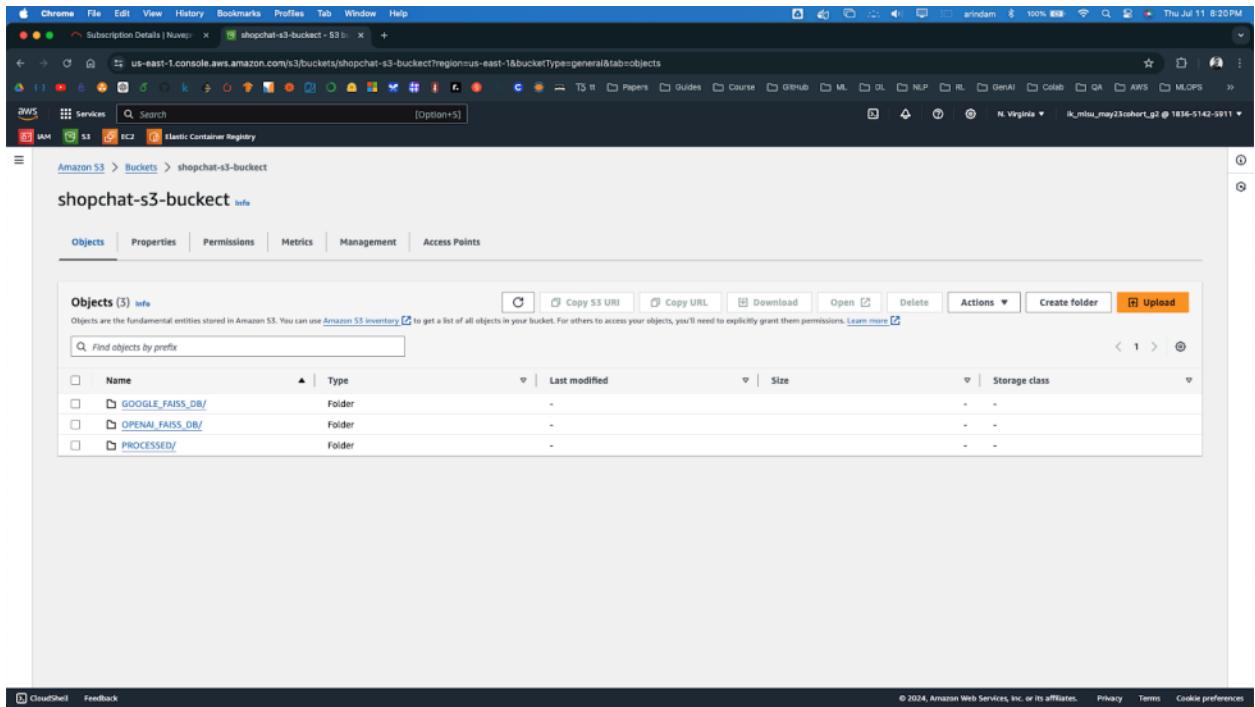
We will be using OpenAIEmbedding model from Langchain_openai. Script is set to use “openai” from the selection menu. *Note: This script runs about 20-25 minutes*

```

3. data_embedding.py
src > components > 3. data_embedding.py > ...
73 class DataEmbedding:
74     def initiate_data_embedding(self):
75         upload_directory_to_s3(self.embedding_config)
76
77         # Remove working directory
78         if os.path.exists(self.embedding_config):
79             shutil.rmtree(self.embedding_config)
80             print(f"Deleted {self.embedding_config}")
81
82         logging.info("Data Embedding - completed")
83
84     except Exception as e:
85         raise CustomException(e, sys)
86
87
88     if __name__ == "__main__":
89         # Select the model for embedding
90         #model = "google"
91         model = "openai"
92
93         #MODEL = "finetune"
94         obj=DataEmbedding(model)
95         obj.initiate_data_embedding()
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115

```

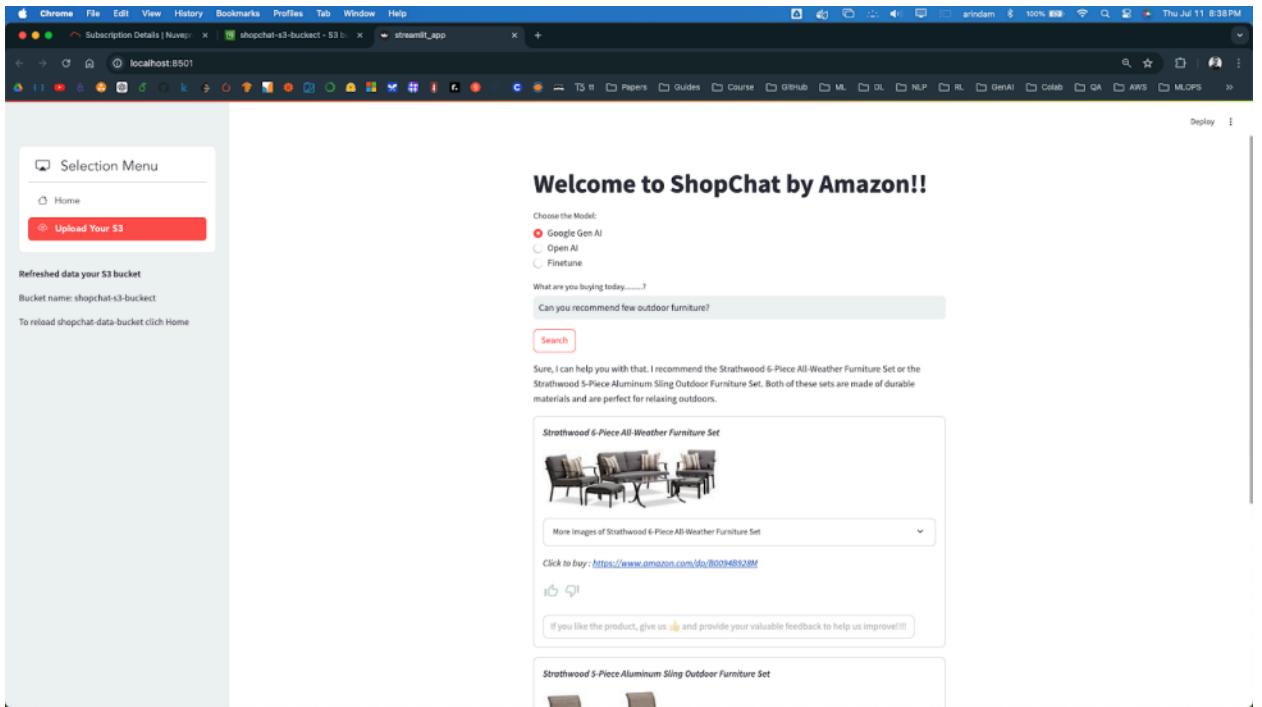
Entry in S3 OPENAI_FAISS_DB folder



6 Full inferencing after data loading

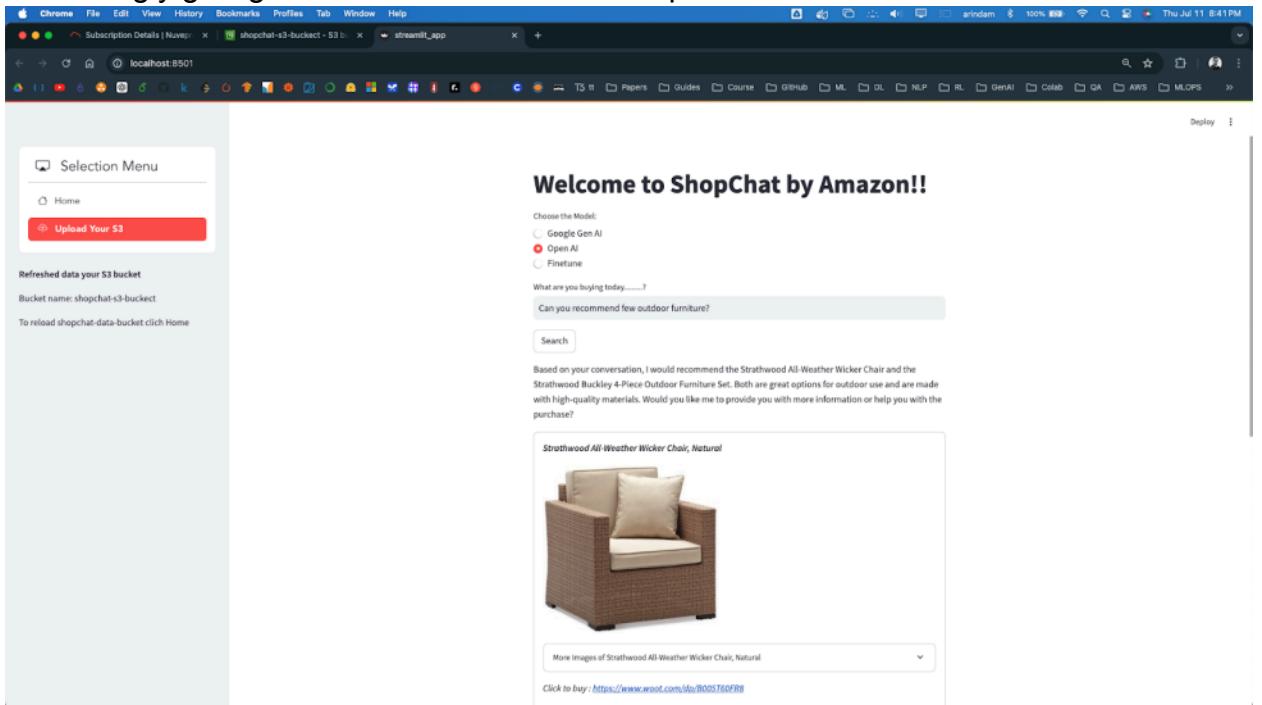
6.1 Upload Your S3

Now we can rerun the ShopChat app using any of the methods described earlier it must not give any error while uploading your S3 bucket. Now using your S3 data to give the results.



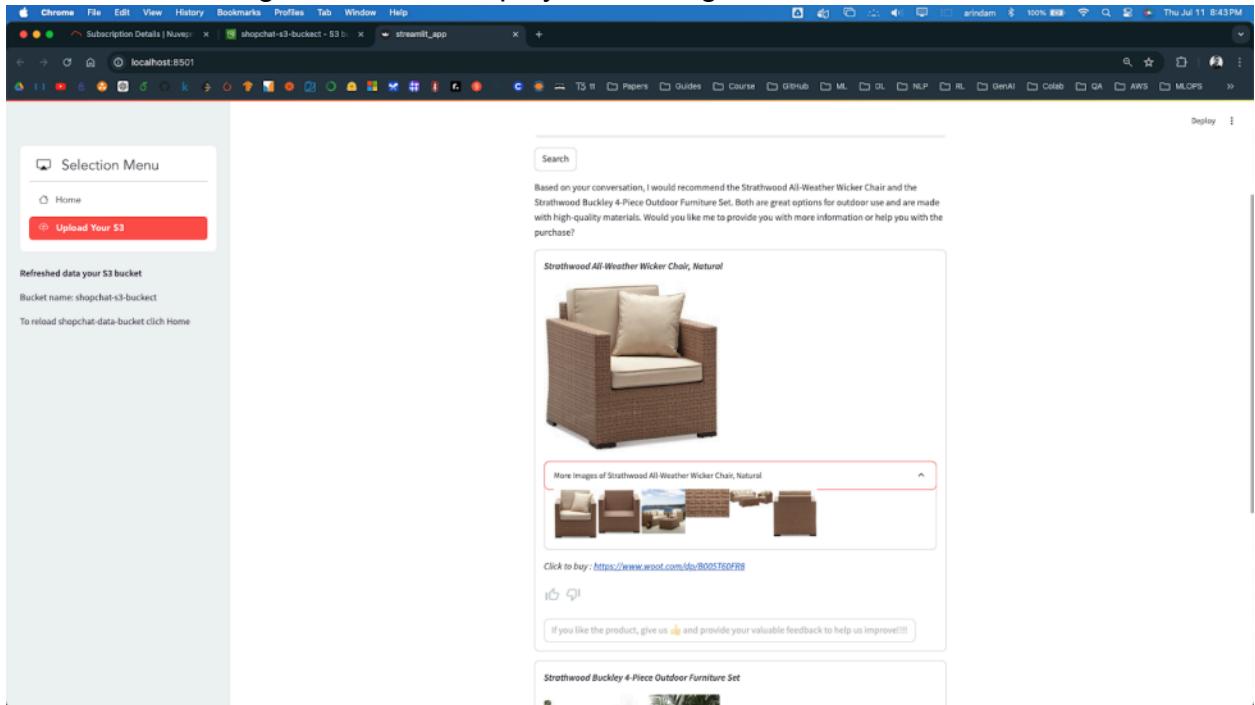
6.2 Selection of LLM model

Changing LLM is easy just select any of the listed models to use for inference. Interestingly giving different result when same question is asked 😊

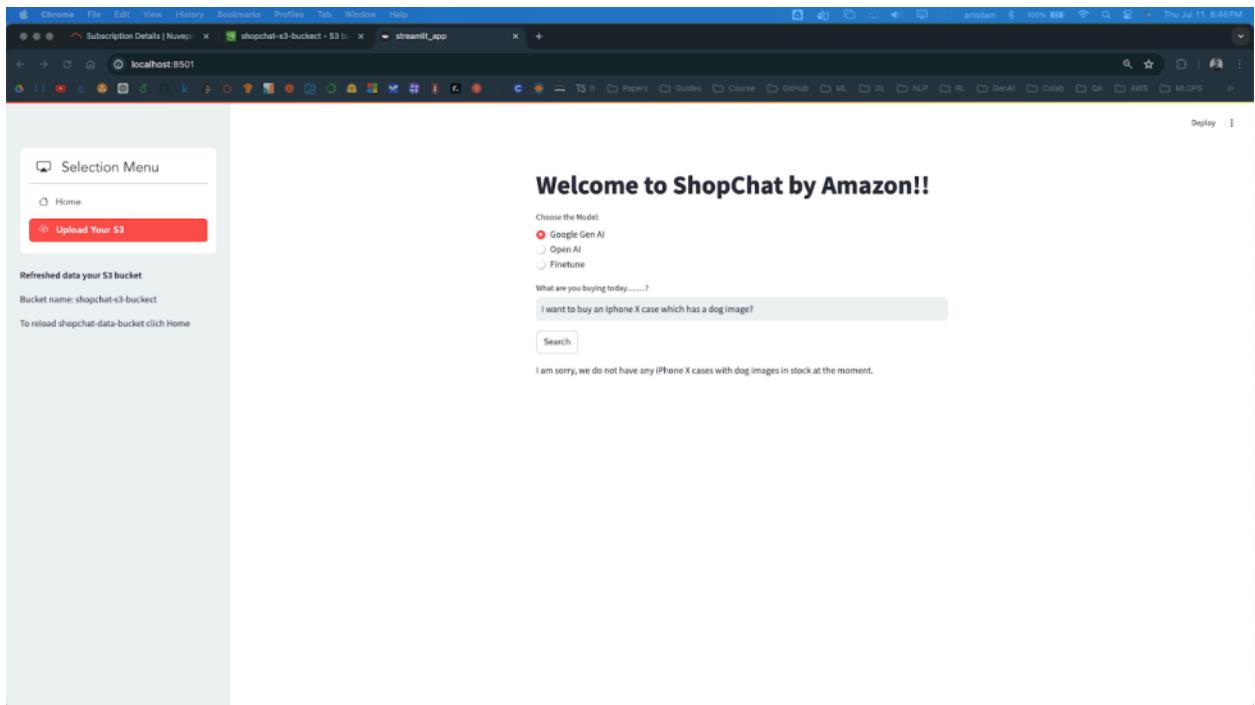


6.3 Display more images

Click on more images and it will display more images of the item if available.

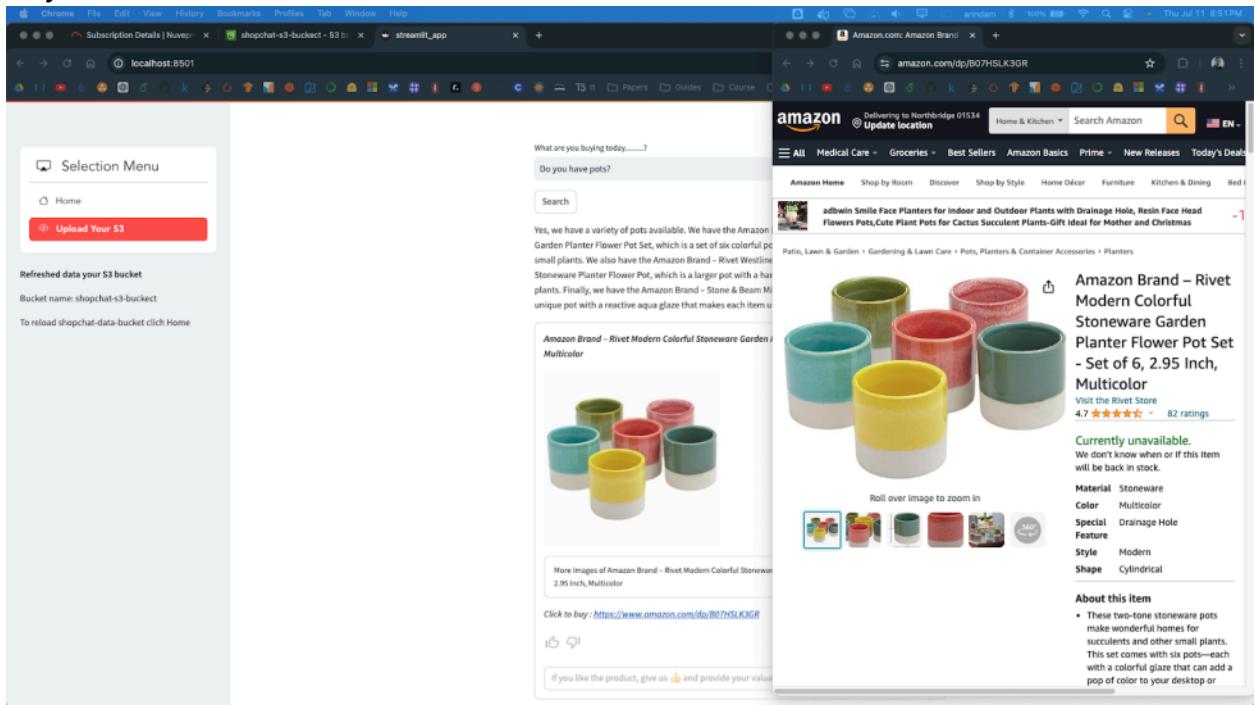


6.4 Item not found



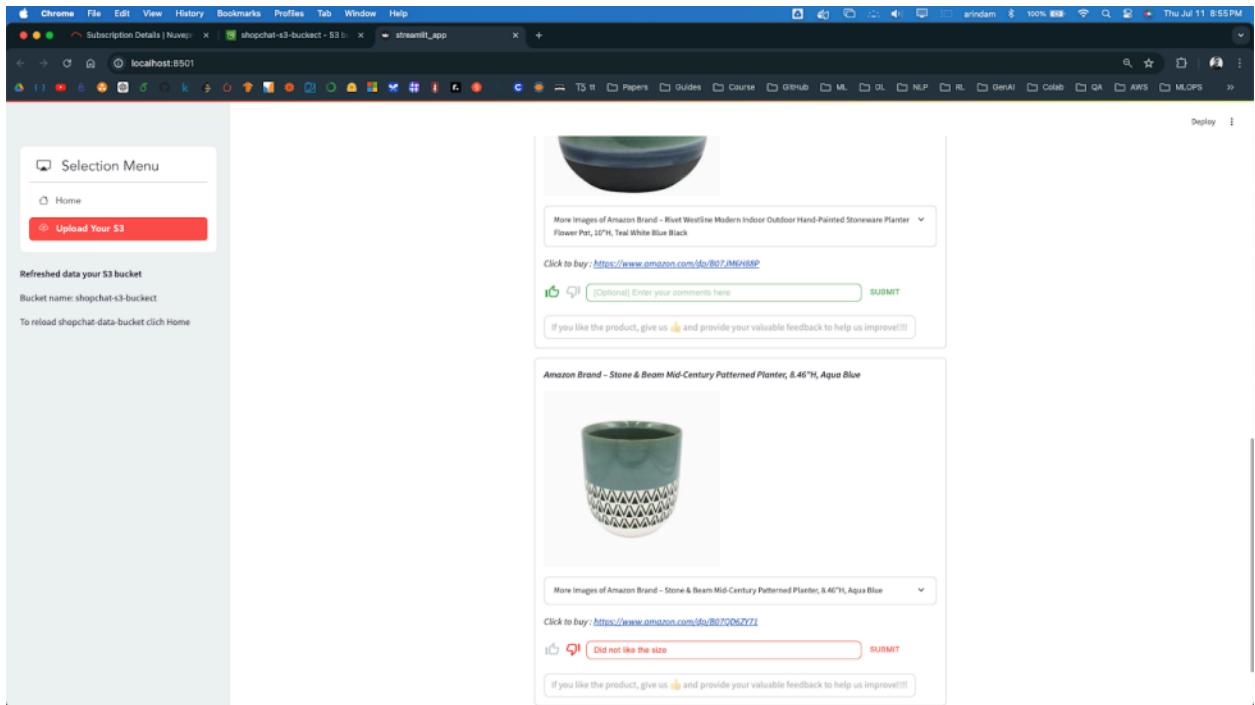
6.5 Listing Link

On clicking, link took me to the Amazon.com website for the product listing. Note – may not work for all items as warned at ABO site itself.



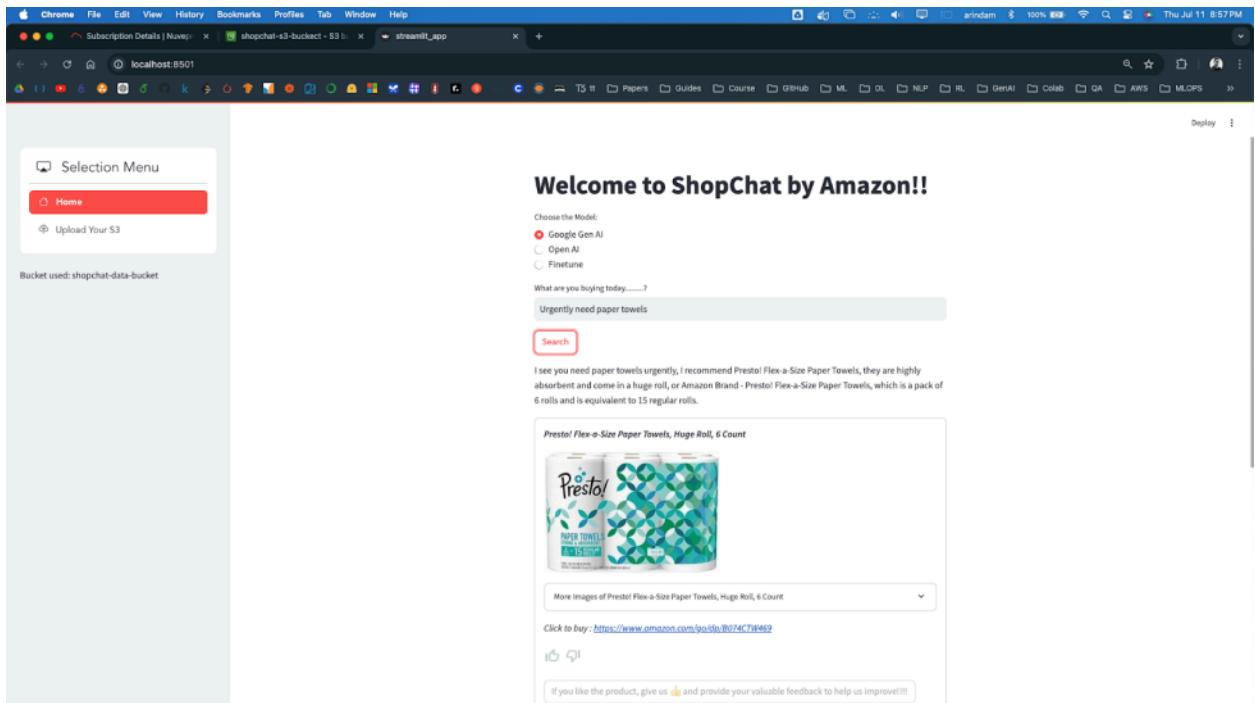
6.6 Feedback

You can like or dislike the product listing and provide your feedback



6.7 Home button

You can click Home button on the left side to refresh the data with default S3 bucket



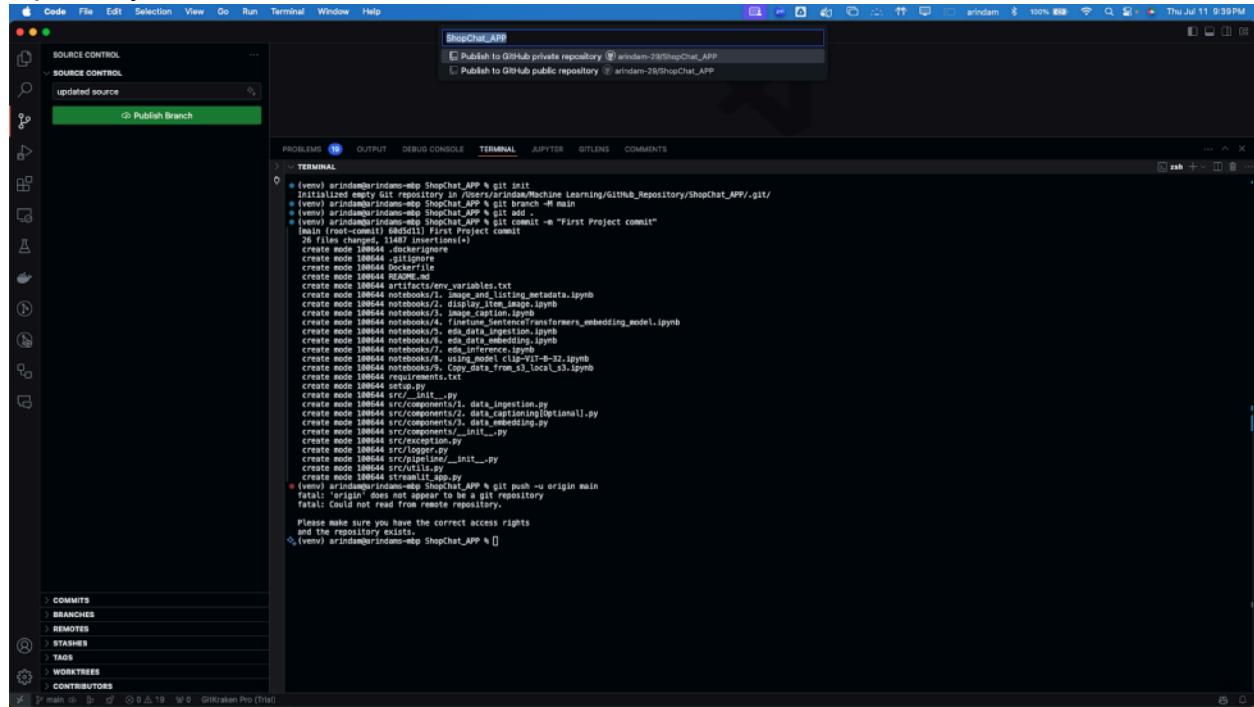
7 CICD with GitHub Action and AWS

7.1 Create Git Repository

Run Following commands on VSCode terminal within this folder, this will create a GitHub repository (ShopChat_App)

- git init [initialize the git in this folder]
- git branch -M main [changing the branch to main from master]
- git add . [add all files]
- git status [check the file status]
- git commit -m "First git setup for the project" [first commit to the project]
- git push -u origin main [push the changes to github]

Go to the VS Code and click on git icon and make sure to select private repository.



The screenshot shows the VSCode interface with the GitHub Actions workflow creation process. The terminal window displays the following command history:

```
(venv) arindam@arindam-mbp: ShopChat_APP % git init
Initialized empty Git repository in /Users/arindam/Machine Learning/GitHub_Repository/ShopChat_APP/.git/
(venv) arindam@arindam-mbp: ShopChat_APP % git branch -M main
Switched to branch 'main'
Your branch is up-to-date with 'origin/main'.
(venv) arindam@arindam-mbp: ShopChat_APP % git add .
(venv) arindam@arindam-mbp: ShopChat_APP % git commit -m "First Project commit"
[master (root-commit) 26] First Project commit
 26 files changed, 11487 insertions(+)
 create mode 100644 .dockerrcignore
 create mode 100644 Dockerfile
 create mode 100644 README.md
 create mode 100644 artifacts/env_variables.txt
 create mode 100644 notebooks/1. Image_and_listing_metadata.ipynb
 create mode 100644 notebooks/2. Data_Ingestion.ipynb
 create mode 100644 notebooks/3. Image_Caption.ipynb
 create mode 100644 notebooks/4. finetune_Sentencetransformers_embedding_model.ipynb
 create mode 100644 notebooks/5. eda_data_embedding.ipynb
 create mode 100644 notebooks/6. eda_inference.ipynb
 create mode 100644 notebooks/7. finetune_Sentencetransformers_embedding.ipynb
 create mode 100644 notebooks/8. Copy_data_From_m3_local_m3.ipynb
 create mode 100644 requirements.txt
 create mode 100644 src/_init_.py
 create mode 100644 src/components/1. data_Ingestion.py
 create mode 100644 src/components/2. data_CapturingOptional.py
 create mode 100644 src/components/3. data_embedding.py
 create mode 100644 src/components/_init__.py
 create mode 100644 src/logger.py
 create mode 100644 src/streamlit_app.py
 create mode 100644 src/utils.py
(venv) arindam@arindam-mbp: ShopChat_APP % git push -u origin main
fatal: 'origin' does not appear to be a git repository
fatal: Could not read from remote repository.
```

The status bar at the bottom indicates the current file is 'SHOPCHAT APP'.

Check in GitHub to make sure repository is created

ShopChat_APP · Private

Code Issues Pull requests Actions Projects Security Insights Settings

main 1 Branch 0 Tags Go to file Add file Code About

arindam-29 First Project commit 60d5d11 - 7 minutes ago 1 Commit

artifacts	First Project commit	7 minutes ago
notebooks	First Project commit	7 minutes ago
src	First Project commit	7 minutes ago
.dockerignore	First Project commit	7 minutes ago
.gitignore	First Project commit	7 minutes ago
Dockerfile	First Project commit	7 minutes ago
README.md	First Project commit	7 minutes ago
requirements.txt	First Project commit	7 minutes ago
setup.py	First Project commit	7 minutes ago
streamlit_app.py	First Project commit	7 minutes ago

README

No description, website, or topics provided.

Readme Activity 0 stars 1 watching 0 forks

No releases published Create a new release

No packages published Publish your first package

Languages

7.2 Create AWS ECR Repository

Log into your AWS account and search for ECR, select “private repository”, give a name to the repository (shopchat-repo) and rest all options are left default.

Amazon Elastic Container Registry

Private registry Repositories Settings

Public registry Repositories Settings

ECR public gallery Amazon ECS Amazon EKS

Getting started Documentation

Amazon ECR > Private registry > Repositories

Private repositories

Repositories (1)

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
shopchat-repo	183651425911.dkr.ecr.us-east-1.amazonaws.com/shopchat-repo	July 11, 2024, 21:50:25 (UTC-04)	Disabled	Manual	AES-256

View push commands Delete Actions Create repository

7.3 Create AWS EC2 Instance

Go to AWS account, search for EC2 and launch a new EC2 instance with following suggested configuration (can be different as well, but make sure at least 16gb of memory is selected)

The screenshot shows the 'Launch an instance' wizard in the AWS Management Console. The configuration steps are as follows:

- Name and tags:** Name is set to "ShopChat-Instance".
- Application and OS Images (Amazon Machine Image):** An AMI is selected: "Canonical, Ubuntu, 24.04 LTS (HVM) - SSD Volume Type".
- Virtual server type (instance type):** Instance type is selected as "g4dn.xlarge".
- Firewall (security group):** A new security group is created.
- Storage (volumes):** 2 volumes (1) - 133 GB are selected.
- Free tier information:** A tooltip explains the free tier benefits: 750 hours of t2.micro (or t3.micro in regions where t2.micro is unavailable) instance usage, 750 hours of public IPv4 address usage per month, 30 GB of EBS storage, 2 million IOPS, 1 GB of snapshots, and 100 GB of bandwidth to the internet.
- Quick Start:** Various AMI options are shown, including Amazon Linux, macOS, Ubuntu, Windows, Red Hat, and SUSE.
- Summary:** The summary shows the configuration details: 1 instance, Canonical, Ubuntu, 24.04 LTS (HVM) - SSD Volume Type, g4dn.xlarge, New security group, 2 volumes (1) - 133 GB, and a note about the free tier.
- Actions:** Buttons for "Cancel", "Launch instance" (highlighted in orange), and "Review commands".

The second part of the screenshot shows the "Create key pair" dialog box, which is part of the "Launch instance" process. It asks for a key pair name ("shopchat-key") and provides options for private key file format (.pem or .ppk). It also includes a warning about storing the private key securely. Buttons for "Cancel" and "Create key pair" are present.

Note: Download the new key pair, which will be needed later to log into the instance

The screenshot shows the AWS EC2 Launch Instance wizard in progress. The configuration steps are as follows:

- Instance type:** g4dn.xlarge selected. It's a Family: g4dn, 4xCPU, 16 GB Memory instance. Current generation: true. On-Demand Linux base pricing: 0.526 USD per hour. On-Demand SUSE base pricing: 0.582 USD per hour. On-Demand Windows base pricing: 0.711 USD per hour. On-Demand RHEL base pricing: 0.584 USD per hour.
- Key pair (login):** shopchat-key selected.
- Network settings:** Network: vpc-064f1b67907d7d38a, Subnet: No preference (Default subnet in any availability zone). Auto-assign public IP: Enabled.
- Summary:** Number of instances: 1. Software Image (AMI): Canonical, Ubuntu, 24.04 LTS, ami-04d81a99f5ec58529. Virtual server type (instance type): g4dn.xlarge. Firewall (security group): New security group. Storage (volumes): 2 volume(s) - 133 GB.
- Free tier information:** A tooltip indicates that the free tier includes 750 hours of t2.micro (or t3.micro in Regions where t2.micro is unavailable) instance usage on free-tier AMIs per month, 750 hours of public IPv4 address usage per month, 30 GB of EBS storage, 2 million I/Os, 1 GB of snapshots, and 100 GB of bandwidth to the internet.
- Launch Instance button:** The final step before launching the instance.

Instance is created

The screenshot shows the AWS EC2 Instances page. On the left, there's a sidebar with various navigation options like EC2 Dashboard, EC2 Global View, Events, Console-to-Code, Instances, Instance Types, Launch Templates, Spot Requests, Savings Plans, Reserved Instances, Dedicated Hosts, Capacity Reservations, Images, AMIs, AMI Catalog, Elastic Block Store, Volumes, Snapshots, Lifecycle Manager, Network & Security, Security Groups, Elastic IPs, Placement Groups, Key Pairs, IAM Role, Connect Shell, and Feedback.

The main area displays a table of instances. One instance is selected: "ShopChat-Instance" (ID: i-0041ce14aa20e9e80). The instance is listed as "Running" with an "Initializing" status check. It has a Public IPv4 DNS of "ec2-3-84-21-139.compute-1.amazonaws.com" and a Public IPv4 address of "3.84.21.139". The instance type is "g4dn.xlarge" and it's located in "us-east-1a".

A detailed view of the selected instance is shown in a modal window titled "i-0041ce14aa20e9e80 (ShopChat-Instance)". The "Details" tab is active, showing the following information:

- Instance ID:** i-0041ce14aa20e9e80 (ShopChat-Instance)
- IPv6 address:** -
- Hostname type:** IP name: ip-172-31-91-133.ec2.internal
- Answer private resource DNS name:** IPv4 (A)
- Auto-assigned IP address:** 3.84.21.139 [Public IP]
- IAM Role:** -
- Public IPv4 address:** 3.84.21.139 [open address]
- Instance state:** Running
- Private IP DNS name (IPv6 only):** ip-172-31-91-133.ec2.internal
- Instance type:** g4dn.xlarge
- VPC ID:** vpc-064f1b67907d7d38a [open address]
- Subnet ID:** -
- Private IPv4 addresses:** 172.31.91.133
- Public IPv4 DNS:** ec2-3-84-21-139.compute-1.amazonaws.com [open address]
- Elastic IP addresses:** -
- AWS Compute Optimizer finding:** User: arn:aws:ss::183651425911:federated-user/k_msu_may23cohrt_g2 is not authorized to perform: compute-optimizer:GetEnrollmentStatus on resource: * because no identity-based policy allows the compute-optimizer:GetEnrollmentStatus action
- Auto Scaling Group name:** -

Go to Connect and SSH client. Follow the instructions and Copy the Example.

The screenshot shows the "Connect to instance" dialog box for the selected instance. The dialog has tabs for "EC2 Instance Connect", "Session Manager", "SSH client", and "EC2 serial console". The "SSH client" tab is selected.

The dialog provides instructions for connecting:

- Open an SSH client.
- Locate your private key file. The key used to launch this instance is "shopchat-key.pem".
- Run this command, if necessary, to ensure your key is not publicly viewable:
chmod 400 "shopchat-key.pem"
- Connect to your instance using its Public DNS:
ssh -i "shopchat-key.pem" ubuntu@ec2-3-84-21-139.compute-1.amazonaws.com

An example command is shown in a code block:

```
ssh -i "shopchat-key.pem" ubuntu@ec2-3-84-21-139.compute-1.amazonaws.com
```

A note at the bottom states: "Note: In most cases, the guessed username is correct. However, read your AMI usage instructions to check if the AMI owner has changed the default AMI username."

Logged in to the instance using ssh client

```

ubuntu@ip-172-31-91-133: ~ (0.22s)
Welcome to Ubuntu 24.04 LTS (GNU/Linux 6.8.0-1009-aws x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Fri Jul 12 02:03:23 UTC 2024

System load: 0.0 Temperature: -0.1 °C
Usage of /: 5.4% of 28.02GB Processes: 123
Memory usage: 1% Users logged in: 0
Swap usage: 0% IPv4 address for ens5: 172.31.91.133

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

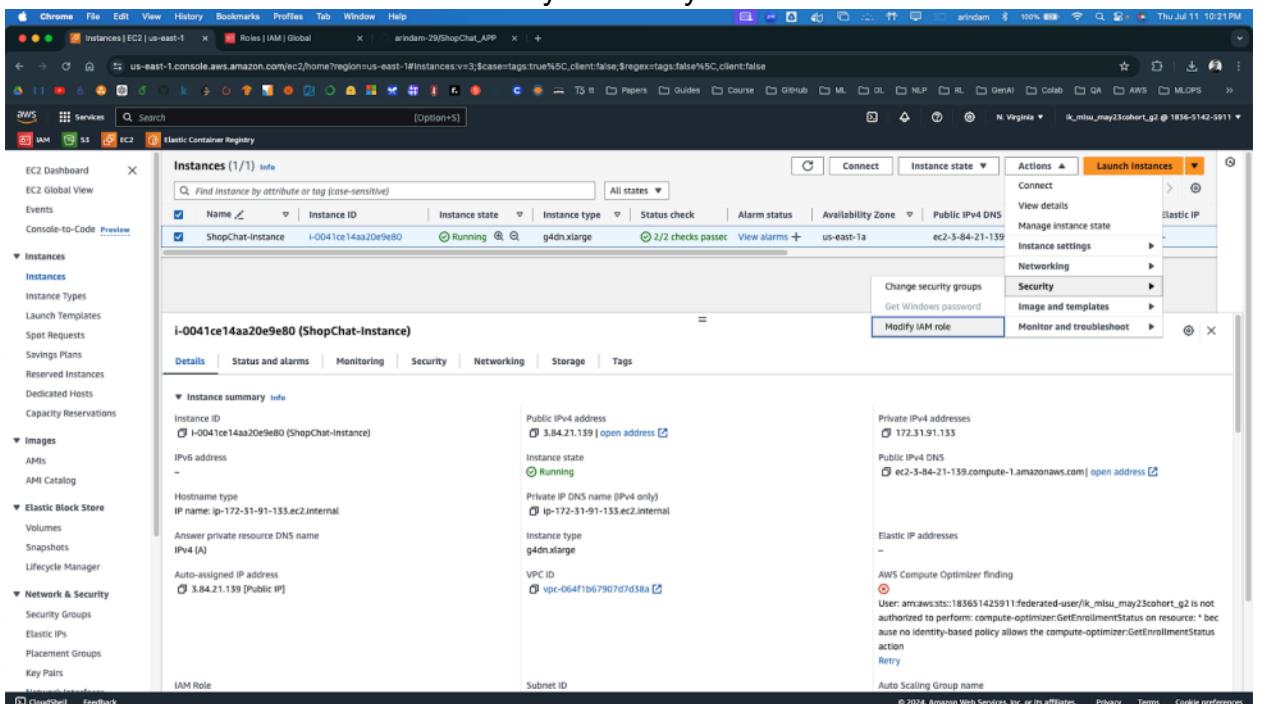
To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-91-133: ~

```

7.4 Create IAM Role and add to EC2 instance

Now we need to give necessary access to S3 and ECR to the new EC2 instance so that it can access both the resources. Go to the main page of EC2. Select the instance and click on Action -> Security -> Modify IAM role



From here select “Create new IAM role”

Modify IAM role

Attach an IAM role to your instance.

Instance ID: i-0041ce14aa20e9e80 (ShopChat-Instance)

IAM role

Select an IAM role to attach to your instance or create a new role if you haven't created any. The role you select replaces any roles that are currently attached to your instance.

Choose IAM role

⚠ If you choose No IAM Role, any IAM role that is currently attached to the instance will be removed. Are you sure you want to remove from the selected instance?

Cancel

Identity and Access Management (IAM)

Roles (10)

An IAM role is an identity you can create that has specific permissions with credentials that are valid for short durations. Roles can be assumed by entities that you trust.

Role name	Trusted entities	Last activity
AWSServiceRoleForAmazonElasticFilesystem	AWS Service: elasticfilesystem [Service-Linked Role]	Yesterday
AWSServiceRoleForAmazonSageMakerNotebooks	AWS Service: sagemaker [Service-Linked Role]	Yesterday
AWSServiceRoleForCloudTrail	AWS Service: cloudtrail [Service-Linked Role]	439 days ago
AWSServiceRoleForFMS	AWS Service: fms [Service-Linked Role]	598 days ago
AWSServiceRoleForOrganizations	AWS Service: organizations [Service-Linked Role]	305 days ago
AWSServiceRoleForServiceQuotas	AWS Service: servicequotas [Service-Linked Role]	22 days ago
AWSServiceRoleForSSO	AWS Service: sso [Service-Linked Role]	-
AWSServiceRoleForSupport	AWS Service: support [Service-Linked Role]	21 days ago
AWSServiceRoleForTrustedAdvisor	AWS Service: trustedadvisor [Service-Linked Role]	-
nl-admin-dent-del	Account: 722045817191	3 hours ago

Roles Anywhere

Authenticate your non-AWS workloads and securely provide access to AWS services.

Access AWS from your non-AWS workloads

X.509 Standard

Temporary credentials

Create role

Click on Create role and select Use Case as EC2 and click next

Select trusted entity

Trusted entity type

- AWS service
- AWS account
- SAML 2.0 federation
- Custom trust policy

Use case
Allow an AWS service like EC2, Lambda, or others to perform actions in this account.

Service or use case
EC2

Choose a use case for the specified service.
Use case

- EC2
- EC2 Role for AWS Systems Manager
- EC2 Spot Fleet Role
- EC2 - Spot Fleet Auto Scaling

Select policy permission

- AmazonEC2ContainerRegistryFullAccess
- AmazonS3FullAccess

In the next page give a Role Name

Role details

Role name
Enter a meaningful name to identify this role.
EC2_to_S3_and_ECR_access

Maximum 64 characters. Use alphanumeric and '+-_@-' characters.

Description
Add a short explanation for this role.
Allows EC2 instances to call AWS services on your behalf.

Step 1: Select trusted entities

Trust policy

```

1 - {
2 -   "Version": "2012-10-17",
3 -   "Statement": [
4 -     {
5 -       "Effect": "Allow",
6 -       "Action": [
7 -         "sts:AssumeRole"
8 -       ],
9 -       "Principal": [
10 -         {
11 -           "Service": [
12 -             "ec2.amazonaws.com"
13 -           ]
14 -         }
15 -       ]
16 -     }
17 -   ]
18 - }
  
```

Step 2: Add permissions

Click on create role

The screenshot shows the 'Create role' wizard in progress. The current step is 'Step 2: Add permissions'. It displays a JSON policy document:

```
5   "Effect": "Allow",
6   "Action": [
7     "sts:AssumeRole"
8   ],
9   "Principal": [
10     {
11       "Service": [
12         "ec2.amazonaws.com"
13       ]
14     }
15   ]
16 ]
```

Below the policy editor, there's a table titled 'Permissions policy summary' showing two attached policies:

Policy name	Type	Attached as
AmazonEC2ContainerRegistryFullAccess	AWS managed	Permissions policy
AmazonS3FullAccess	AWS managed	Permissions policy

At the bottom of the screen, there are 'Cancel', 'Previous', and 'Create role' buttons.

Go back to Modify IMA role page, refresh the IAM role and select the one you have created and click Update IAM role

The screenshot shows the 'Modify IAM role' dialog. The 'Instance ID' dropdown is set to 'i-0041ce14aa20e9e80 [ShopChat-Instance]'. The 'IAM role' dropdown is set to 'EC2_to_S3_and_ECR_access'. At the bottom right, there is a large orange 'Update IAM role' button.

IAM role is now added to EC2 instance

7.5 Install Docker into EC2 instance

Run the following commands one by one in EC2 ssh client to install and run docker:

- sudo apt-get update -y
- sudo apt-get upgrade
- curl -fsSL https://get.docker.com -o get-docker.sh
- sudo sh get-docker.sh
- sudo usermod -aG docker ubuntu
- newgrp docker

Docker is now installed in EC2

The screenshot shows a terminal window titled "Ubuntu@ip-172-31-91-133 ~" with the following command history and output:

```
sudo sh get-docker.sh
API version: 1.46
Go version: go1.21.11
Git commit: 7d4bcdb
Built: Sat Jun 29 00:02:23 2024
OS/Arch: linux/amd64
Context: default

Server: Docker Engine - Community
Engine:
  Version: 27.0.3
  Min Version: 1.46 (minimum version 1.24)
  Go version: go1.21.11
  Git commit: 662f78c
  Built: Sat Jun 29 00:02:23 2024
  OS/Arch: linux/amd64
  Experimental: false
containerd:
  Version: 1.7.18
  GitCommit: ae7191c4f5e67bb4d5ae76a6b735f29cc25774e
runc:
  Version: 1.7.18
  GitCommit: v1.1.13-0-g58aa020
docker-init:
  Version: 0.19.0
  GitCommit: de4bad0

=====
To run Docker as a non-privileged user, consider setting up the
Docker daemon in rootless mode for your user:
  dockerd-rootless-setup.sh install
Visit https://docs.docker.com/go/rootless/ to learn about rootless mode.

To run the Docker daemon as a fully privileged service, but granting non-root
users access, refer to https://docs.docker.com/go/daemon-access/
WARNING: Access to the remote API on a privileged Docker daemon is equivalent
to root access on the host. Refer to the 'Docker daemon attack surface'
documentation for details: https://docs.docker.com/go/attack-surface/
=====

ubuntu@ip-172-31-91-133 ~ (8:143s)
sudo usermod -aG docker ubuntu
ubuntu@ip-172-31-91-133 ~
newgrp docker
ubuntu@ip-172-31-91-133:~$
```

7.6 Add Secrets to GitHub Actions

Add following secrets. Got to GitHub repository -> Settings -> Securities -> Secrets and Variable -> Actions -> **Click on New repository secret**

- GOOGLE_API_KEY =
- OPENAI_API_KEY =
- AWS_ACCESS_KEY_ID =
- AWS_SECRET_ACCESS_KEY =
- YOUR_S3_BUCKET_NAME = **shopchat-s3-bucket**
- ECR_REPOSITORY_NAME = **shopchat-repo**
- AWS_ECR_LOGIN_URI = **183651425911.dkr.ecr.us-east-1.amazonaws.com**
- AWS_REGION = **us-east-1**
- SHOPCHAT_BUCKET_NAME = **shopchat-data-bucket**
- ABO_BUCKET_NAME = **amazon-berkeley-objects**
- ARTIFACTS_FOLDER = **..../artifacts/**
- WORKING_DIR = **..../artifacts/downloads/**

- **EDA_FOLDER_NAME = EDA_FILES**

Note: ECR name and login URI can be found on ECR repository page:

Repository name	URI	Created at	Tag immutability	Scan frequency	Encryption type
shopchat-repo	183651425911.dkr.ecr.us-east-1.amazonaws.com/shopchat-repo	July 11, 2024, 21:50:25 (UTC-04)	Disabled	Manual	AES-256

Secrets added

Name	Last updated
ABO_BUCKET_NAME	2 minutes ago
ARTIFACTS_FOLDER	1 minute ago
AWS_ACCESS_KEY_ID	4 minutes ago
AWS_ECR_LOGIN_URI	now
AWS_REGION	3 minutes ago
AWS_SECRET_ACCESS_KEY	3 minutes ago
ECR_REPOSITORY_NAME	now
EDA_FOLDER_NAME	1 minute ago
GOOGLE_API_KEY	6 minutes ago
OPENAI_API_KEY	6 minutes ago
SHOPCHAT_BUCKET_NAME	2 minutes ago
WORKING_DIR	1 minute ago
YOUR_S3_BUCKET_NAME	2 minutes ago

7.7 Add Runner

We will create a self-hosted runner to trigger the deployment pipeline. Got to GitHub repository -> Settings -> Actions -> Runners -> Click on **New self-hosted runner** and select **Linux as Runner image**. Copy all the commands and paste them in EC2 ssh client

The screenshot shows the GitHub Actions Runner configuration page. The 'Runner image' is set to 'Linux' and 'Architecture' to 'x64'. The 'Download' section contains the following shell script:

```
# Create a folder
$ mkdir actions-runner && cd actions-runner

# Download the latest runner package
$ curl -o actions-runner-linux-x64-2.317.0.tar.gz -L
https://github.com/actions/runner/releases/download/v2.317.0/actions-runner-linux-x64-
2.317.0.tar.gz

# Optional: Validate the hash
$ echo "fb833d210ff8fc628af4f5475a5457c380353f901877d51cc01a17b2a91161d" actions-runner-linux-x64-
2.317.0.tar.gz | shasum -a 256 -c

# Extract the installer
$ tar xzf ./actions-runner-linux-x64-2.317.0.tar.gz
```

The 'Configure' section contains the following shell script:

```
# Create the runner and start the configuration experience
$ ./config.sh --url https://github.com/arindam-29/ShopChat_APP --token
BBF3Q3ZTFDRTFYENAPM3VEDGSCW3W

# Last step, run it!
$ ./run.sh
```

After pasting `./config.sh --url https://github.com/arindam-29/ShopChat_APP --token BBF3Q3ZTFDRTFYENAPM3VEDGSCW3W` (first from the configure) use following:

- Enter the name of the runner group to add this runner to: [press Enter for Default] -> Hit Enter for Default
- Enter the name of runner: [press Enter for ip-172-31-91-133] -> Use **“self-hosted”**
- This runner will have the following labels: 'self-hosted', 'Linux', 'X64'
- Enter any additional labels (ex. label-1,label-2): [press Enter to skip] -> Hit Enter to skip
- Enter name of work folder: [press Enter for _work] -> Hit Enter

```

Warp File Edit View Tab Blocks Window Help
ubuntu@ip-172-31-91-133: ~ + 
newgrp docker
ubuntu@ip-172-31-91-133:~$ mkdir actions-runner && cd actions-runner
ubuntu@ip-172-31-91-133:~/actions-runner$ curl -o actions-runner-linux-x64-2.317.0.tar.gz -L https://github.com/actions/runner/releases/download/v2.317.0/actions-runner-linux-x64-2.317.0.tar.gz
tar.gz
% Total    % Received % Xferd  Average Speed   Time   Time  Current
          Dload Upload Total Spent   Left Speed
 0     0    0     0    0     0      0 --:--:-- --:--:-- --:--:-- 0
100 170M  100 170M    0      0 440M
ubuntu@ip-172-31-91-133:~/actions-runner$ curl -o 9e803d210df8c6020af4f4745a457d300353f9d01877d51cc01a17b2a91161d actions-runner-linux-x64-2.317.0.tar.gz | shasum -a 256 -c
actions-runner-linux-x64-2.317.0.tar.gz: OK
ubuntu@ip-172-31-91-133:~/actions-runners$ tar xzf ./actions-runner-linux-x64-2.317.0.tar.gz
ubuntu@ip-172-31-91-133:~/actions-runners$ ./config.sh --url https://github.com/arindam-29/ShopChat_APP --token BBF3Q3ZTFDRTFYENAPM3VEDGSCW3W

```

Self-hosted runner registration

Authentication

✓ Connected to GitHub

Runner Registration

Enter the name of the runner group to add this runner to: [press Enter for Default]

Enter the name of runner: [press Enter for ip=172-31-91-133] self-hosted

This runner will have the following labels: 'self-hosted', 'Linux', 'X64'

Enter any additional labels (ex. label-1,label-2): [press Enter to skip]

✓ Runner successfully added

✓ Runner connection is good

Runner settings

Enter name of work folder: [press Enter for _work]

✓ Settings Saved.

ubuntu@ip-172-31-91-133:~/actions-runner\$./run.sh

✓ Connected to GitHub

Current runner version: '2.317.0'

2024-07-12 03:28:48Z: Listening for jobs

Self-hosted runner is now Connected to GitHub and listening for Jobs. You can check the status in GitHub repository -> Settings -> Actions -> Runners. It must be in **Idle** status

Chrome File Edit View Bookmarks Profiles Tab Window Help arindam 100% Thu Jul 11 11:27PM

Home EC2 us-east-1 x Elastic Container Registry x Runners · arindam-29/ShopChat_APP x

github.com/arindam-29/ShopChat_APP/settings/actions/runners

arindam-29 / ShopChat_APP

Code Issues Pull requests Actions Projects Security Insights Settings

New self-hosted runner

Runners	Status
self-hosted	idle

General

Access

Collaborators

Code and automation

- Branches
- Tags
- Rules
- Actions**
- General
- Runners
- Webhooks
- Codestyles
- Pages

Security

- Code security and analysis
- Deploy keys
- Secrets and variables

Integrations

- GitHub Apps
- Email notifications

© 2024 GitHub, Inc. Terms Privacy Security Status Docs Contact Manage cookies Do not share my personal information

7.8 Add Workflow yml to GitHub repository

Goto the VSCode folder and open **.gitignore** file. Comment out the line **.github/*** and save it.

```

You, 18 seconds ago | author (You)
1 ## YOU WILL BE ASKED TO DELETE/REMOVE BELOW LINE (.github/*) FROM THIS FILE AND SAVE BEFORE RUNNING AWS CI/CD PIPELINE ####
2
3 a.github/*
4
5 ## ONLY DELETE/REMOVE ABOVE LINE (.github/*). DO NOT CHANGE ANYTHING ELSE IN THIS FILE ####
6 ##########
7 # Byte-compiled / optimized / DLL files
8 __pycache__/
9 *.py[co]
10 *$py.class
11 # Folder -arindam
12 artifacts/project/*
13 artifacts/your/*
14 artifacts/downloads/*
15 artifacts/GOOGLER_FAISS_DB/*
16 artifacts/OPENAI_FAISS_DB/*
17 artifacts/FINETUNE_FAISS_DB/*
18 # C extensions
19 *.so
20
21 # Distribution / packaging
22 .Python
23 build/
24 develop-eggs/
25 dist/
26 downloads/
27 eggs/
28 .eggs/
29 lib/
30 lib64/
31 parts/

```

In the terminal, issue to follow git commands to add the change in GitHub repository

- git add .
- git commit -m “adding aws.yml file to repository”
- git push -u origin main

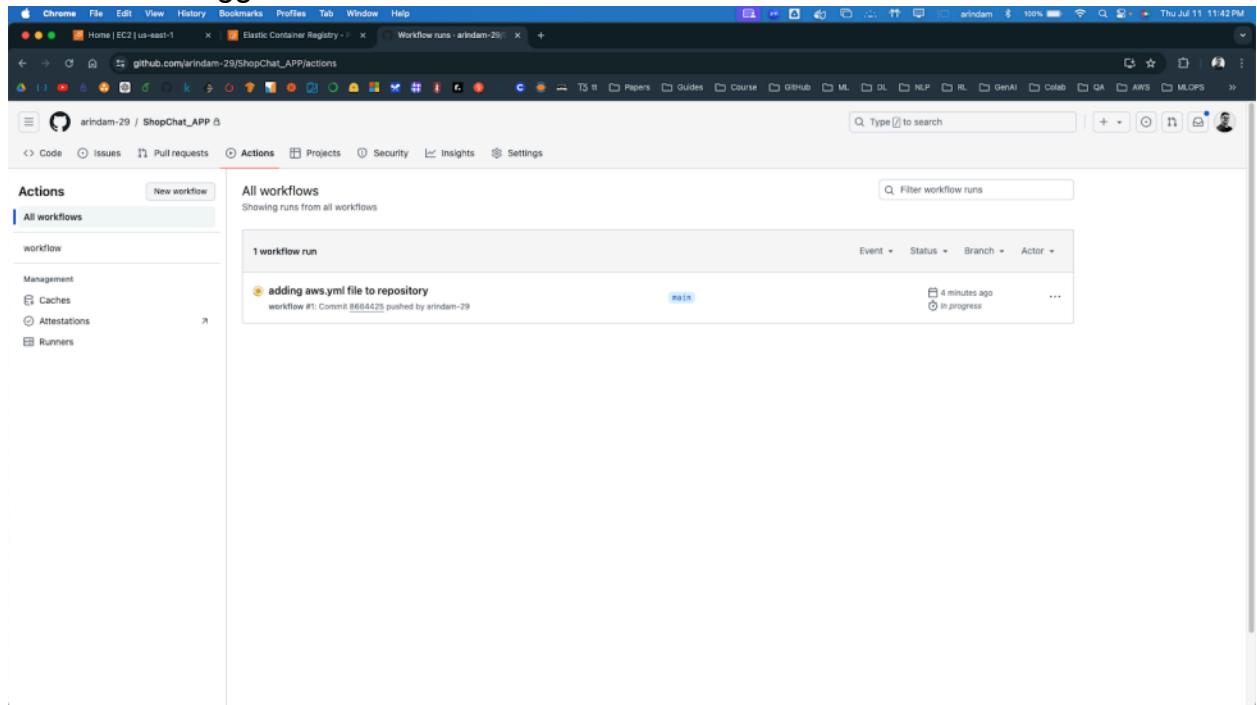
```

You, 20 seconds ago | In 3, Col 11 | Spaces: 4 | UTM-B | LF | Ignore
1 [main 8664425] adding aws.yml file to repository
1 file changed, 112 insertions(+), 0 deletions(-)
2
3 [main 8664425] nothing to commit, working tree clean
4
5 [main 8664425] arindam@arindams-mbp ShopChat_APP % git push -u origin main
6 Counting objects: 100% (12/12), done.
7 Delta compression using up to 14 threads
8 Writing objects: 100% (8/8), 1.75 KiB | 1.75 MiB/s, done.
9 Total 8 (delta 3), reused 0 (delta 0), pack-reused 0
10 To https://github.com/arindam-29/ShopChat_APP.git
11 ! [new branch] main set up to track origin/main.
12
13 [main 8664425] main -> main
14
15 [main 8664425] arindam@arindams-mbp ShopChat_APP %

```

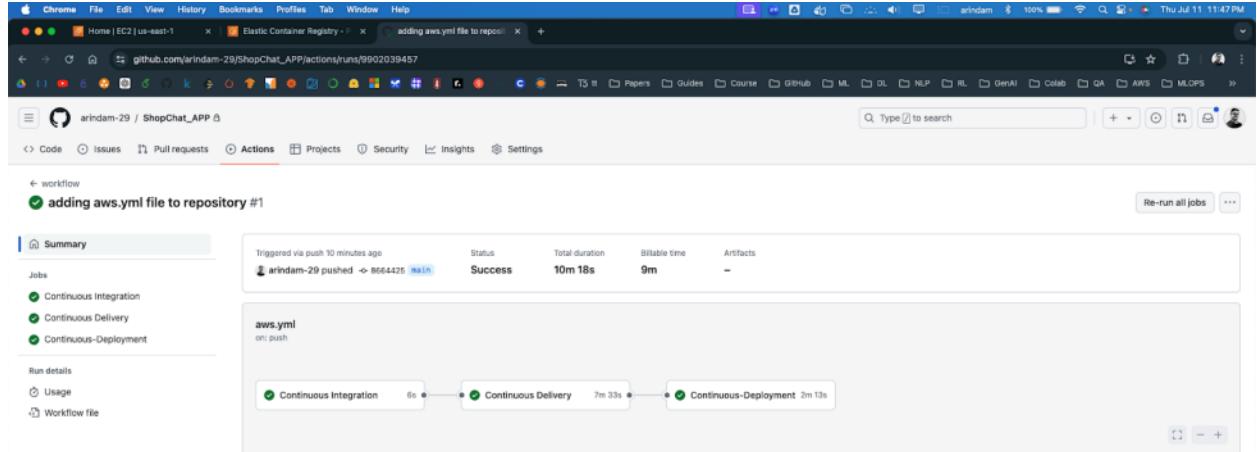
7.9 CICD Pipeline Triggers

Once the aws.yml file is pushed; Go to GitHub repository -> Actions. You can see the workflow triggers.



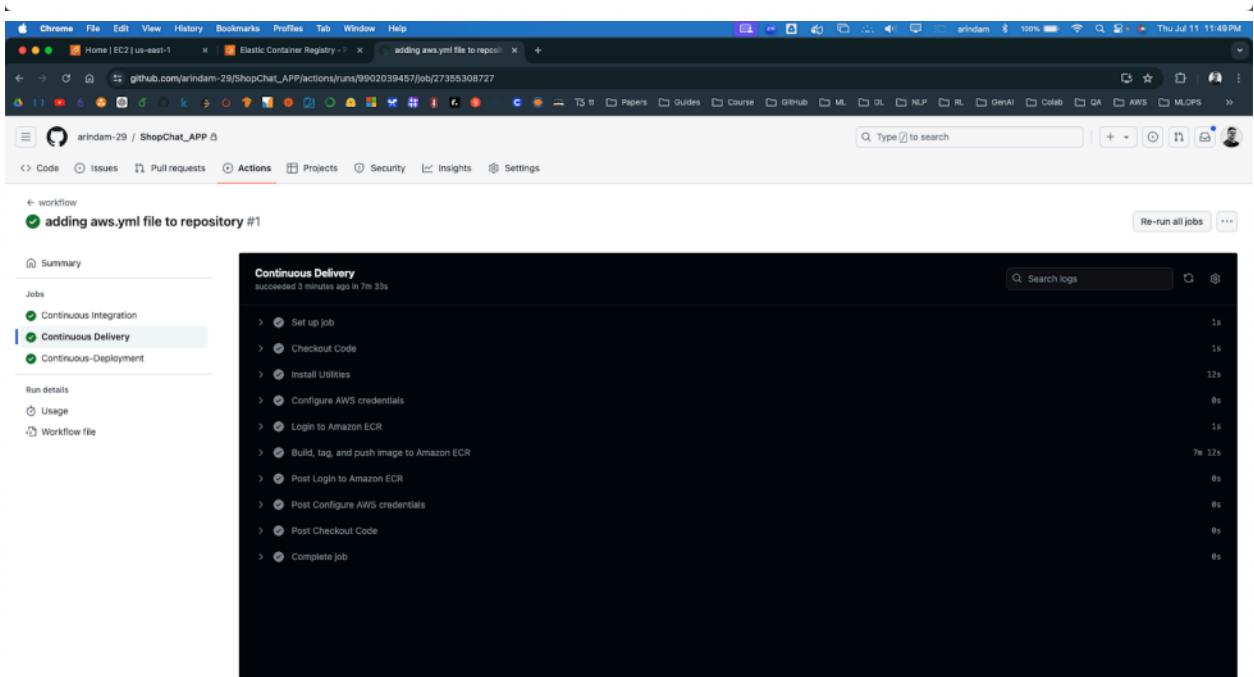
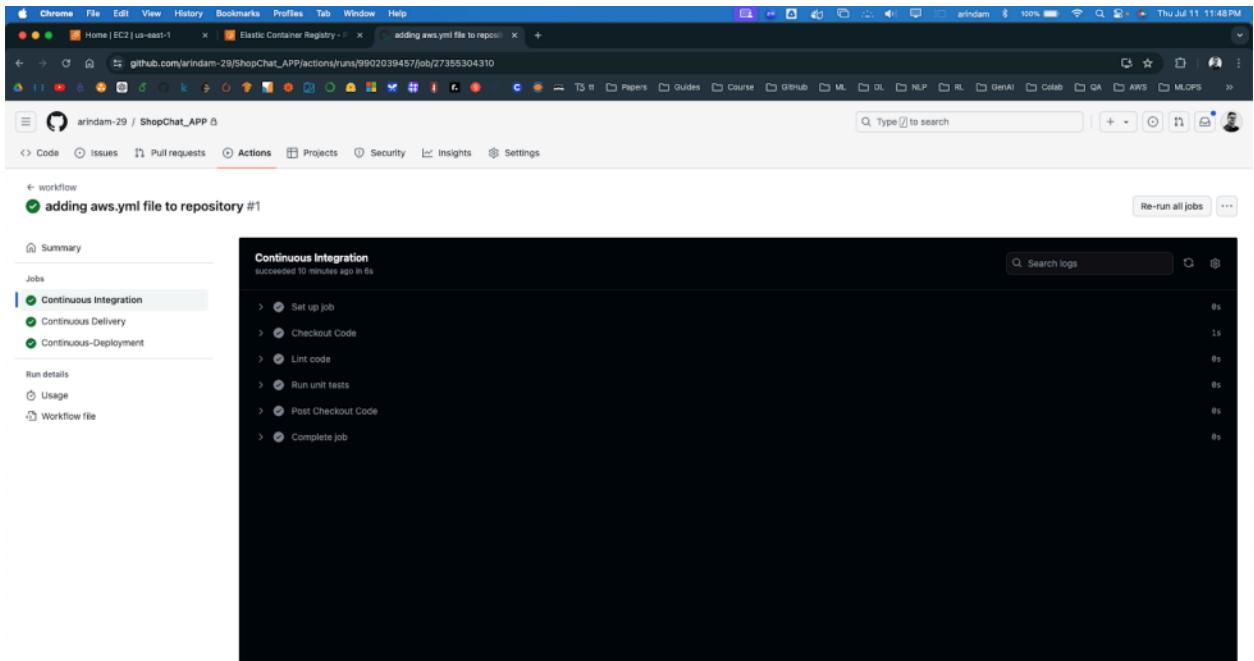
The screenshot shows the GitHub Actions interface for a repository named 'ShopChat_APP'. On the left, there's a sidebar with 'Actions' selected. The main area displays 'All workflows' with one entry: 'adding aws.yml file to repository'. This job was triggered by a commit (commit #8664426) pushed by arindam-29. The status is 'In progress' and it was triggered 4 minutes ago. There are dropdown menus for 'Event', 'Status', 'Branch', and 'Actor' at the top right of the list.

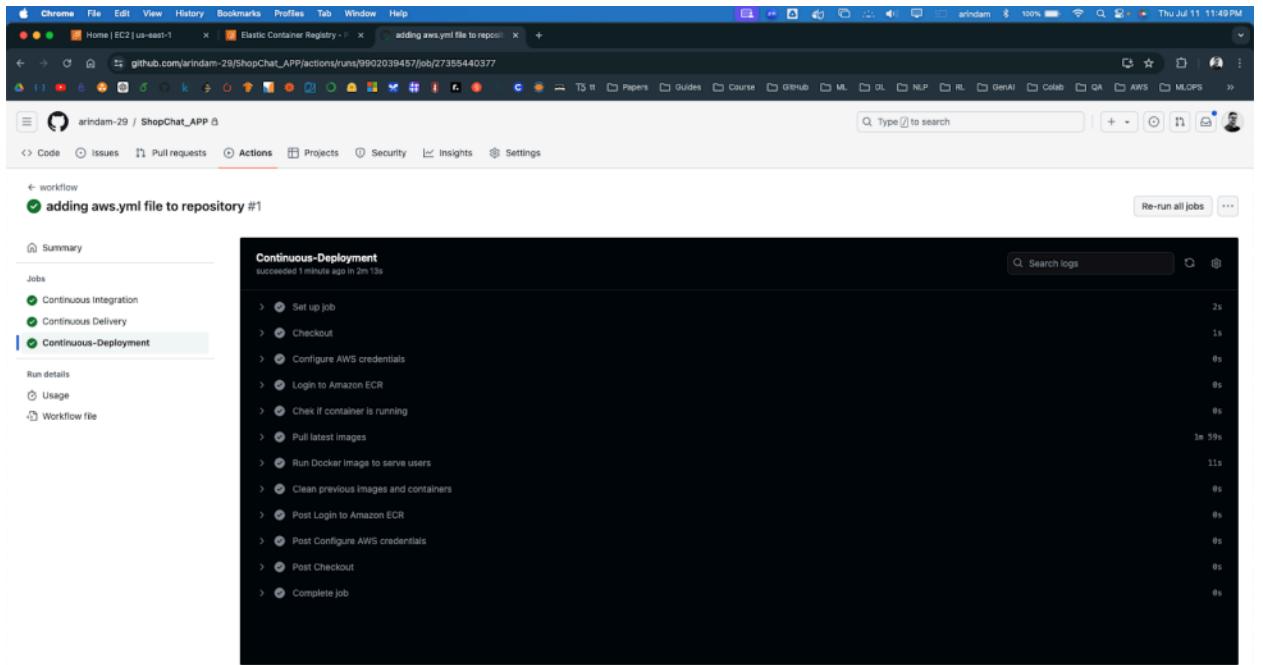
Click on the job. You can see the details of the pipeline



This screenshot shows the detailed view of the 'adding aws.yml file to repository' job. The summary table indicates the job was triggered via push 10 minutes ago, the status is 'Success', total duration was 10m 18s, billable time was 9m, and there were no artifacts. Below the summary, the 'aws.yml' configuration is shown, which includes a 'Continuous Integration' step triggered on 'push'. A timeline diagram shows the flow from 'Continuous Integration' (0s) to 'Continuous Delivery' (7m 33s) and finally 'Continuous-Deployment' (2m 13s).

Also see the steps in each pipeline





Our docker image is now running EC2 instance

7.10 Open port 8501 in EC2

Go to the AWS EC2 instance. Select our instance, go to Security

Name	Security group rule ID	Port range	Protocol	Source	Security groups	Description
-	sgr-0d2bb82637ee9599c	80	TCP	0.0.0.0/0	launch-wizard-1	-
-	sgr-0ca0b5277f4dc48	443	TCP	0.0.0.0/0	launch-wizard-1	-
-	sgr-0177dc36684ec2b903	22	TCP	0.0.0.0/0	launch-wizard-1	-

Click on Security Group -> click on Edit Inbound rules

The screenshot shows the AWS EC2 Security Groups page. The security group 'sg-01f51fc8e00d60af6 - launch-wizard-1' is selected. The 'Inbound rules' tab is active, displaying four rules:

Name	Security group rule ID	IP version	Type	Protocol	Port range	Source	Description
-	sgr-0d2b882637ee9599c	IPv4	HTTP	TCP	80	0.0.0.0/0	-
-	sgr-0cfa0b5277f4dca48	IPv4	HTTPS	TCP	443	0.0.0.0/0	-
-	sgr-0177dc36684e2ba03	IPv4	SSH	TCP	22	0.0.0.0/0	-

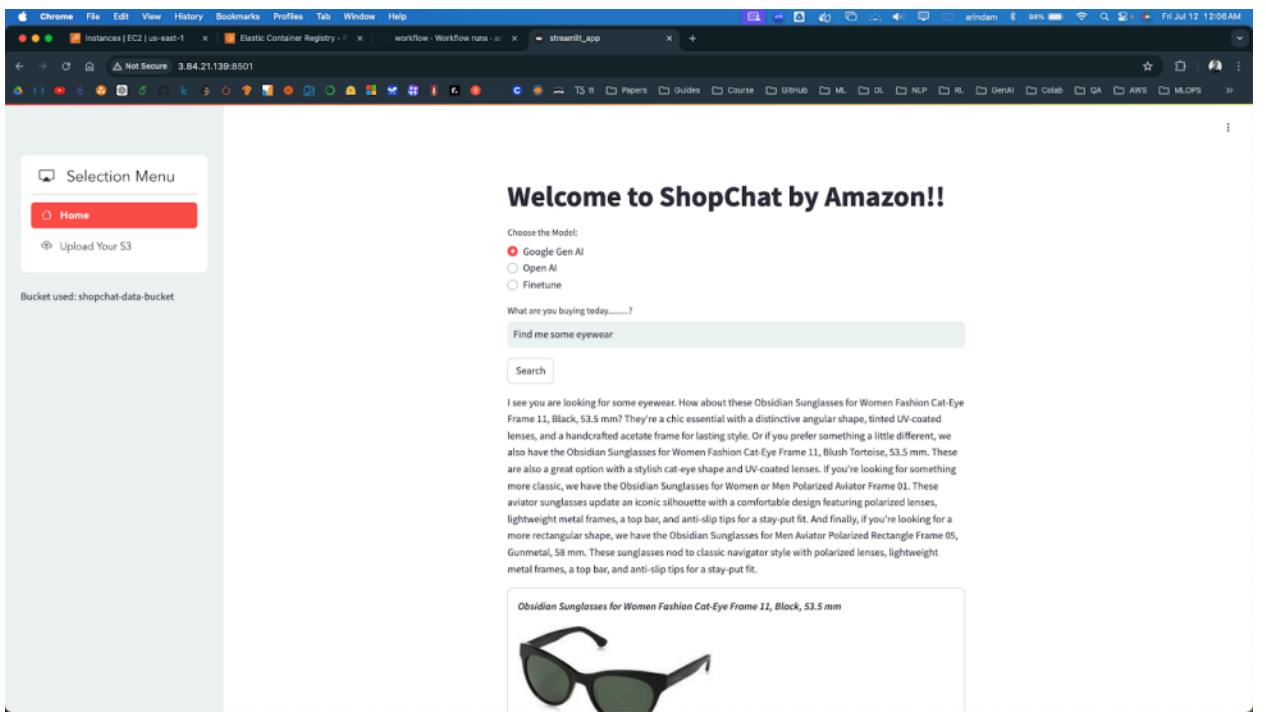
For the first security group rule ID, update the Type to “All traffic” and click on Save rules. Note: lets open all ports here and not only 8501

The screenshot shows the 'Edit inbound rules' dialog for the 'sg-01f51fc8e00d60af6 - launch-wizard-1' security group. The 'Type' dropdown is set to 'All traffic'. The 'Source' dropdown shows '0.0.0.0/0' with a warning message: 'Rules with source of 0.0.0.0/0 or ::/0 allow all traffic.' The 'Description' field is empty. At the bottom right, there are 'Preview changes' and 'Save rules' buttons.

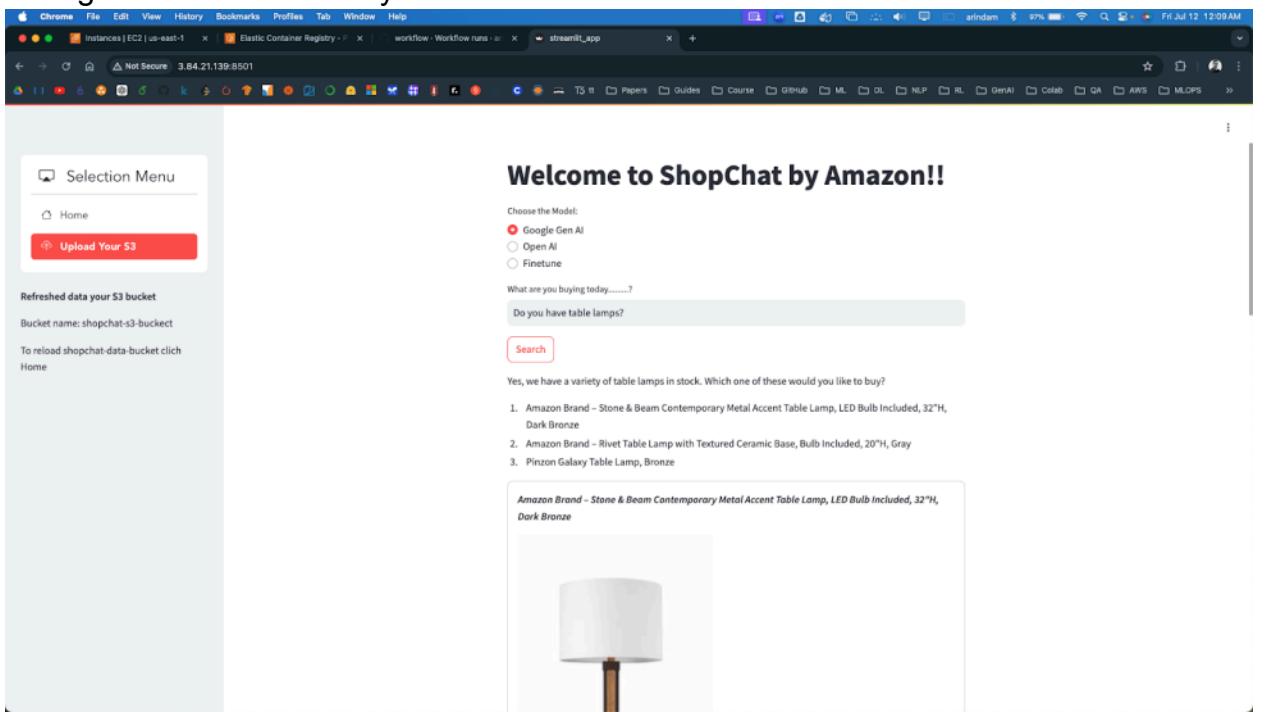
7.11 ShopChat App inference from AWS using public URL

Go to the AWS EC2 instance. Select Public IPv4 address

Goto the Public URL and start interacting with ShopChat App 😊
URL for this instance with port number-> 3.84.21.139:8501

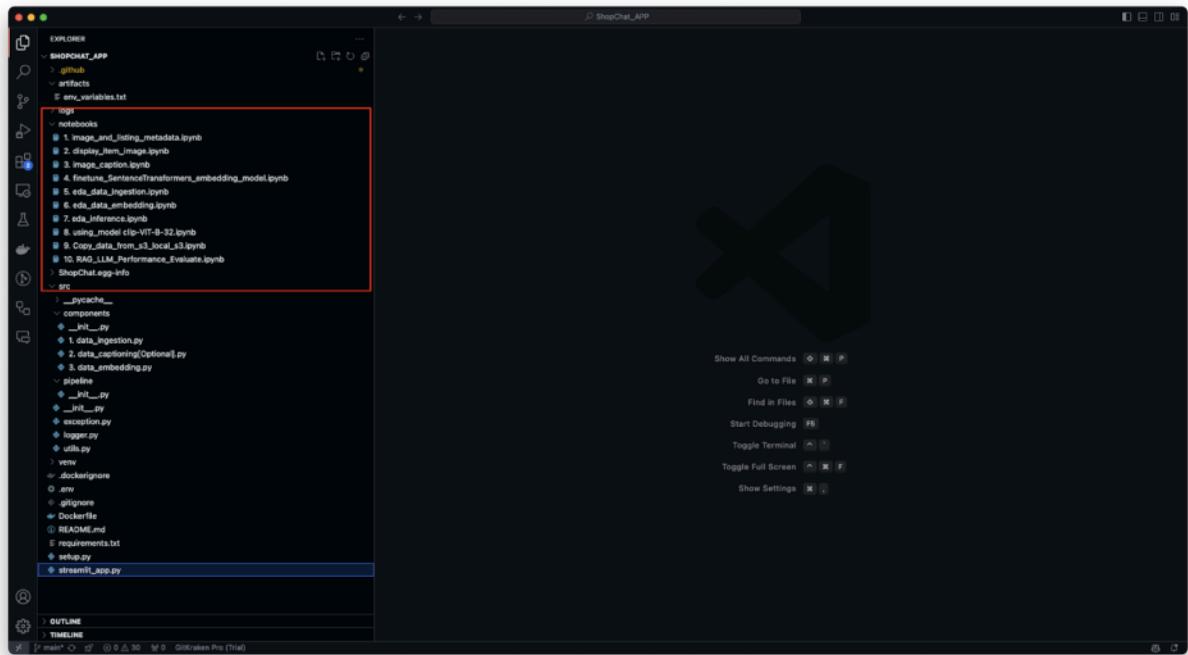


Changed the S3 bucket to your S3 bucket

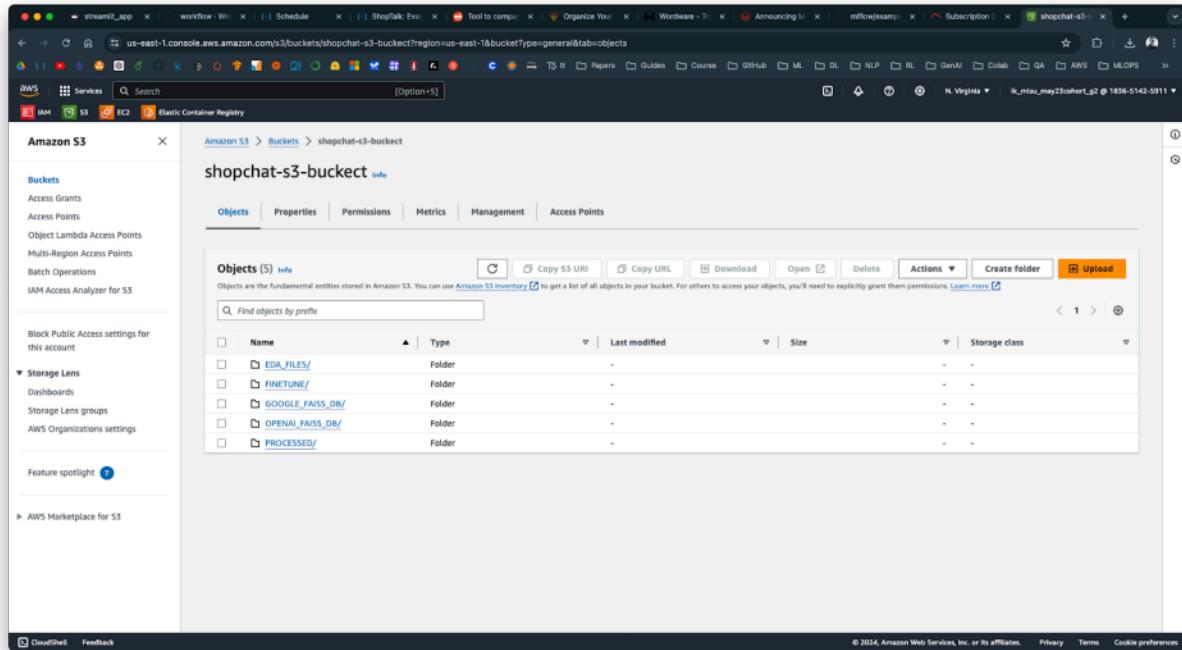


8 EDA and Data Preparation

All EDA notebooks can be found under Notebook folder. The detailed data preparation, cleaning transformation are described in each notebooks



On running the Notebook files EDA and Finetune folders are created in your S3 bucket



9 Supporting scripts and files

9.1 env_variables.txt

This document under artifacts gives the detailed secret keys to be added in GitHub action

```

env_variables.txt
artifacts > env_variables.txt
You, 13 hours ago | 1 author (You)
1 # Add below keys to Github - Actions secrets and variables
2 ### Note: All keys without and quotes
3
4 GOOGLE_API_KEY = [get it from, (login required): https://aisudio.google.com/app/apikey]
5 OPENAI_API_KEY = [get it from, (login required): https://platform.openai.com/settings/profile?tab=api-keys]
6 AWS_ACCESS_KEY_ID = [get it from, (AWS login required) IAM -> Users -> security credentials -> Create access key ]
7 AWS_SECRET_ACCESS_KEY = [Same as AWS_ACCESS_KEY_ID]
8 AWS_REGION = "us-east-1" [Use this US East (N. Virginia)]
9 YOUR_S3_BUCKET_NAME = [login to AWS account and create a S3 bucket for storing all the data]
10 ECR_REPOSITORY_NAME = [login to AWS and create a ECR repository, add the name of the repository]
11 AWS_ECR_LOGIN_URI = [provide the ECR repository login uri here. E.g. "xxxxxxxxxx.dkr.ecr.us-east-1.amazonaws.com"]
12
13 ### Unchanged Variables
14 SHOPCHAT_BUCKET_NAME = "shopchat-data-bucket" [DO NOT change this, S3 bucket for the Project default]
15 ABO_BUCKET_NAME = amazon-berkeley-objects [DO NOT change this, S3 bucket for Amazon Berkeley Objects]
16 ARTIFACTS_FOLDER = "../artifacts/" [DO NOT change this, data artifacts folder used within the project]
17 WORKING_DIR = "../artifacts/downloads/" [DO NOT change this, temporary folder used within the project]
18 EDA_FOLDER_NAME = "EDA_FILES"
19
20
21

```

This screenshot shows the GitHub Actions secrets and variables configuration in Dillenkraken Pro. It displays a file named 'env_variables.txt' containing various environment variables and their values. The variables include API keys for Google, OpenAI, and AWS, along with specific bucket and repository names for the project.

9.2 exception.py

This custom exception scripts raises exception if occurs during the running of any scripts

```

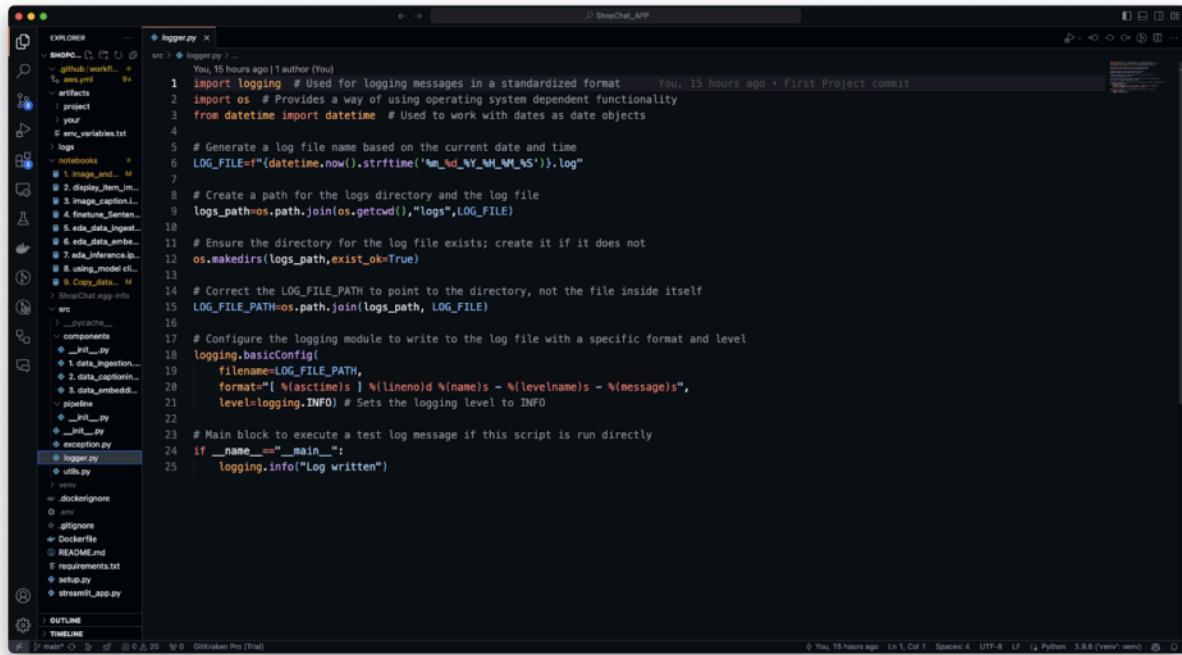
exception.py
src > exception.py
You, 15 hours ago | 1 author (You)
1 # create custom error message handler
2
3 import sys # Used for accessing system-specific parameters and functions
4 from src.logger import logging # Import custom logging module for logging
5
6 # Define a function to extract error message details
7 def error_message_details(error, error_details=sys):
8     # Extract traceback information
9     ...exc_tb=error_details.exc_info()
10    # Retrieve the filename where the error occurred
11    file_name=exc_tb.tb_frame.f_code.co_filename #get the error file name name; look at the python custom exception handling doc in the internet
12    # Format the error message with file name, line number, and error message
13    error_message="Error occurred in script {} line number {} error message {}".format(file_name, exc_tb.tb_lineno, str(error))
14    return error_message
15
16 # Define a custom exception class
You, 15 hours ago | 1 author (You)
17 class CustomException(Exception):
18     def __init__(self, error_message, error_details):
19         super().__init__(error_message)
20         # Store the detailed error message
21         self.error_message=error_message,error_details=error_details
22
23     def __str__(self):
24         # Return the detailed error message when the exception is printed
25         return self.error_message
26
27 # Testing
28 # if __name__=="__main__":
29 #     try:
30 #         a=1/0
31 #     except Exception as e:
32 #         logging.info("Division by zero")
33 #         raise CustomException(e, sys)

```

This screenshot shows the 'exception.py' file in Dillenkraken Pro. The code defines a custom exception class called 'CustomException' that inherits from Python's built-in 'Exception'. It includes methods to handle and format error messages, including the file name, line number, and the original error message. A testing block at the bottom demonstrates how the exception can be raised and caught.

9.3 logging.py

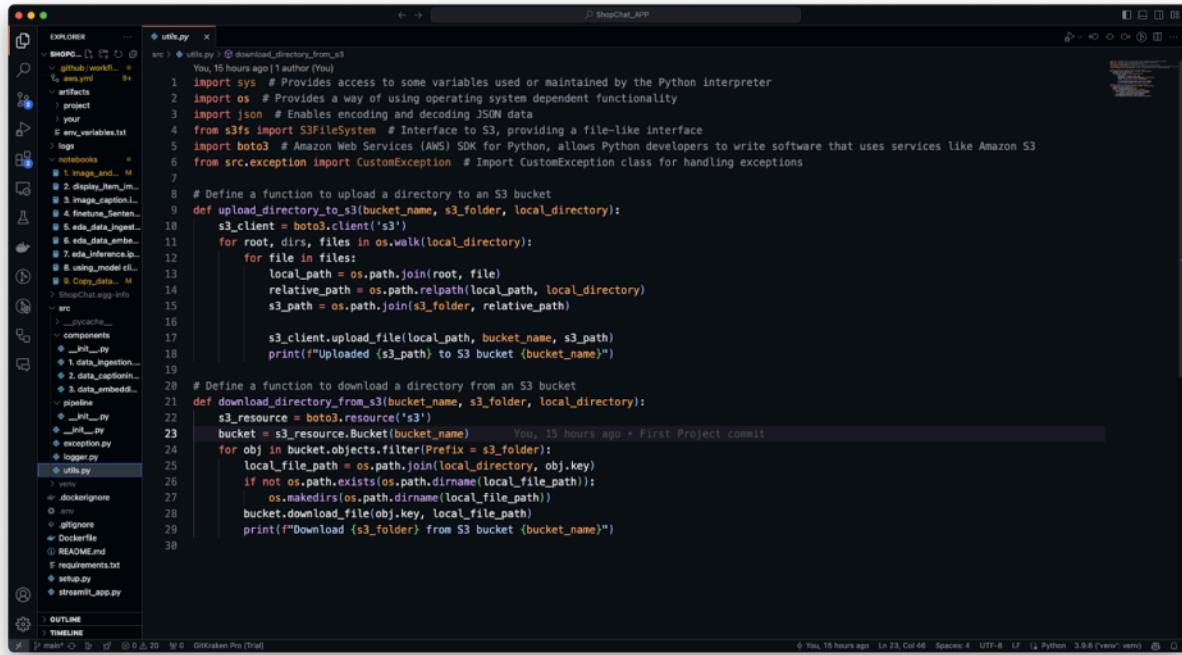
This is to log the info in log files



```
You, 15 hours ago | 1 author (You)
1 import logging # Used for logging messages in a standardized format
2 import os # Provides a way of using operating system dependent functionality
3 from datetime import datetime # Used to work with dates as date objects
4
5 # Generate a log file name based on the current date and time
6 LOG_FILE=f"(datetime.now()).strftime('%m_%d_%Y_%H_%M_%S').log"
7
8 # Create a path for the logs directory and the log file
9 logs_path=os.path.join(os.getcwd(),"logs",LOG_FILE)
10
11 # Ensure the directory for the log file exists; create it if it does not
12 os.makedirs(logs_path,exist_ok=True)
13
14 # Correct the LOG_FILE_PATH to point to the directory, not the file inside itself
15 LOG_FILE_PATH=os.path.join(logs_path, LOG_FILE)
16
17 # Configure the logging module to write to the log file with a specific format and level
18 logging.basicConfig(
19     filename=LOG_FILE_PATH,
20     format=f"[%(asctime)s] %(lineno)d %(name)s - %(message)s",
21     level=logging.INFO) # Sets the logging level to INFO
22
23 # Main block to execute a test log message if this script is run directly
24 if __name__=="__main__":
25     logging.info("Log written")
```

9.4 utils.py

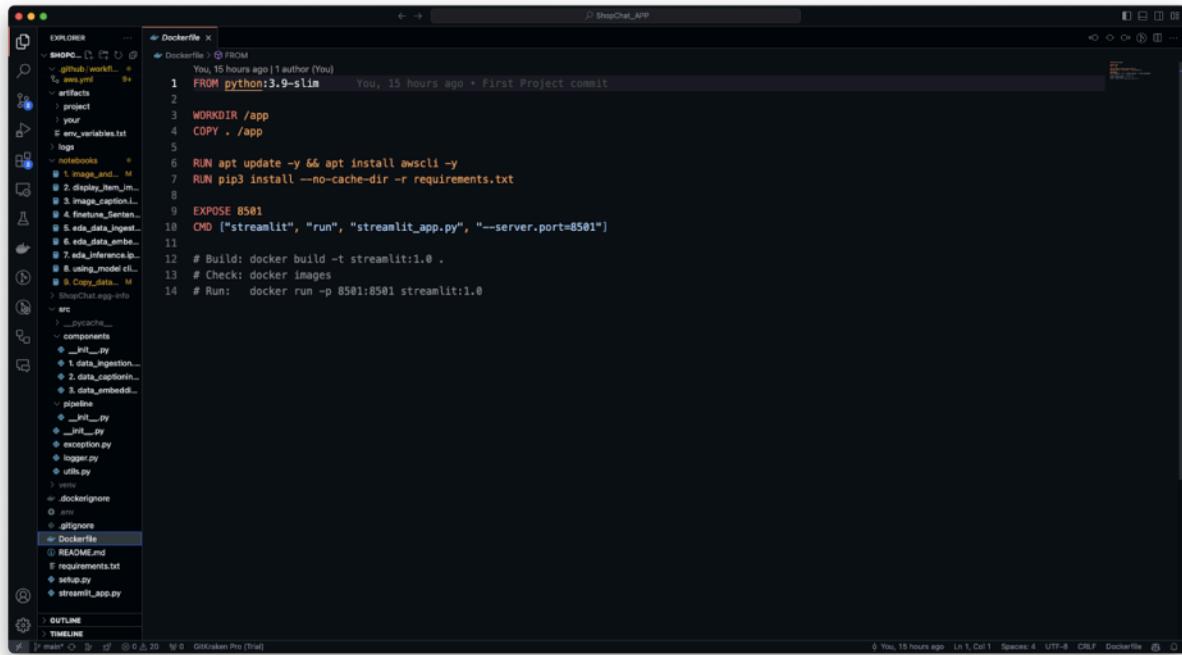
Utility script. There are two commonly used functions defined here. Loading and downloading folders from S3



```
You, 15 hours ago | 1 author (You)
1 import sys # Provides access to some variables used or maintained by the Python interpreter
2 import os # Provides a way of using operating system dependent functionality
3 import json # Enables encoding and decoding JSON data
4 from s3fs import S3FileSystem # Interface to S3, providing a file-like interface
5 import boto3 # Amazon Web Services (AWS) SDK for Python, allows Python developers to write software that uses services like Amazon S3
6 from src.exception import CustomException # Import CustomException class for handling exceptions
7
8 # Define a function to upload a directory to an S3 bucket
9 def upload_directory_to_s3(bucket_name, s3_folder, local_directory):
10     s3_client = boto3.client('s3')
11     for root, dirs, files in os.walk(local_directory):
12         for file in files:
13             local_path = os.path.join(root, file)
14             relative_path = os.path.relpath(local_path, local_directory)
15             s3_path = os.path.join(s3_folder, relative_path)
16
17             s3_client.upload_file(local_path, bucket_name, s3_path)
18             print(f"Uploaded {s3_path} to S3 bucket {bucket_name}")
19
20 # Define a function to download a directory from an S3 bucket
21 def download_directory_from_s3(bucket_name, s3_folder, local_directory):
22     s3_resource = boto3.resource('s3')
23     bucket = s3_resource.Bucket(bucket_name)
24     for obj in bucket.objects.filter(Prefix = s3_folder):
25         local_file_path = os.path.join(local_directory, obj.key)
26         if not os.path.exists(os.path.dirname(local_file_path)):
27             os.makedirs(os.path.dirname(local_file_path))
28         bucket.download_file(obj.key, local_file_path)
29     print(f"Download {s3_folder} from S3 bucket {bucket_name}")
```

9.5 Dockerfile

Scripts to containerize the application to create and run docker image

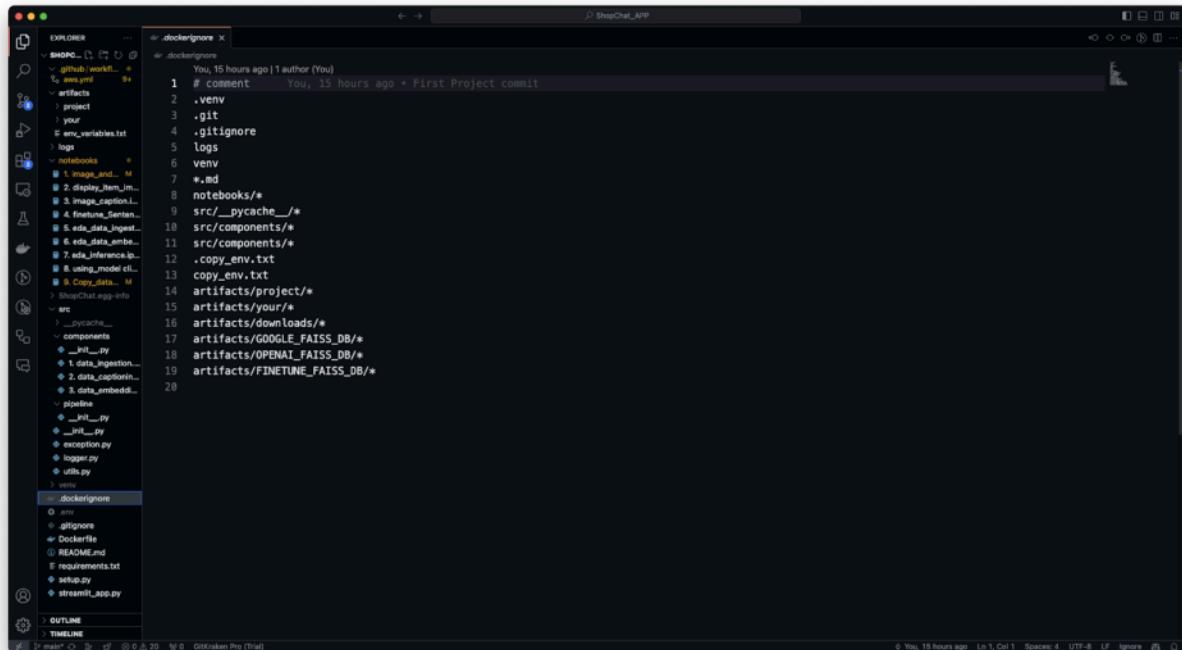


A screenshot of a code editor showing a Dockerfile in a GitHub repository named "ShopChat_APP". The Dockerfile contains the following code:

```
FROM python:3.9-slim
WORKDIR /app
COPY . /app
RUN apt update -y && apt install awscli -y
RUN pip3 install --no-cache-dir -r requirements.txt
EXPOSE 8501
CMD ["streamlit", "run", "streamlit_app.py", "--server.port=8501"]
# Build: docker build -t streamlit:1.0 .
# Check: docker images
# Run: docker run -p 8501:8501 streamlit:1.0
```

9.6 .dockerignore

Contains the list of the files and folders to be ignored while building docker image

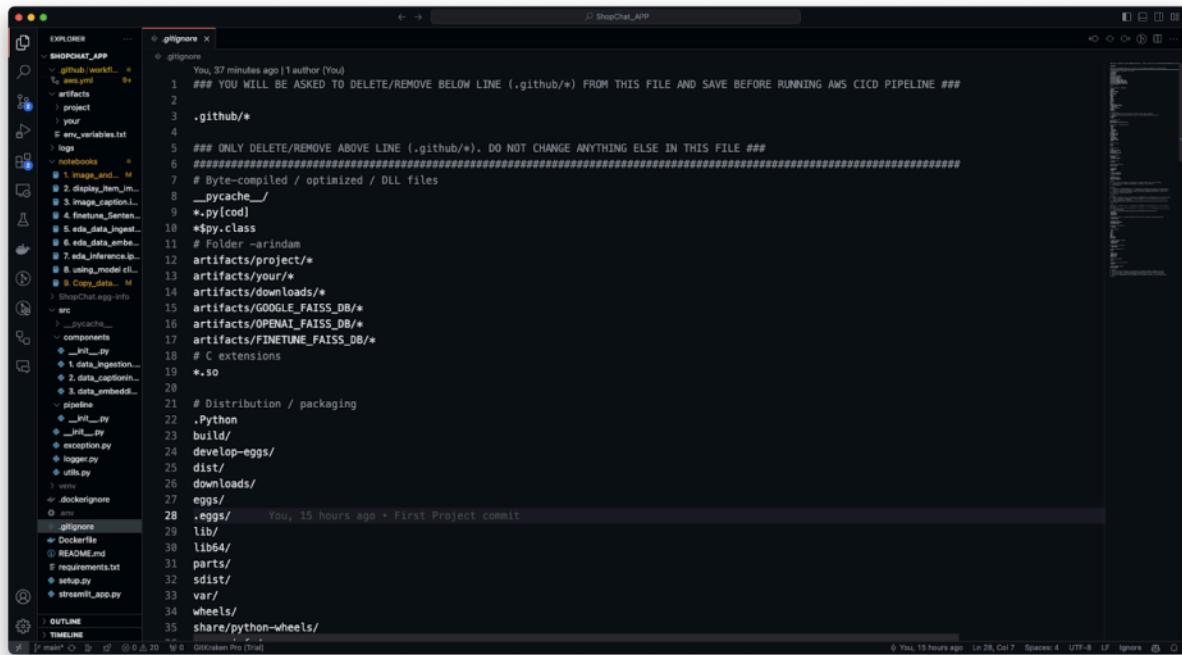


A screenshot of a code editor showing a .dockerignore file in a GitHub repository named "ShopChat_APP". The .dockerignore file contains the following ignore patterns:

```
# comment
.venv
.git
.dockerignore
.logs
.venv
*.md
notebooks/*
src/_pycache_/*
src/components/*
src/components/*
.copy_env.txt
copy_env.txt
artifacts/project/*
artifacts/your/*
artifacts/downloads/*
artifacts/GOOGLE_FAISS_DB/*
artifacts/OPENAI_FAISS_DB/*
artifacts/FINETUNE_FAISS_DB/*
```

9.7 .gitignore

Contains the list of the files and folders to be ignored in GitHub repository



The screenshot shows a code editor window with the file `.gitignore` open. The file contains the following content:

```
You, 57 minutes ago | 1 author (You)
1  ### YOU WILL BE ASKED TO DELETE/REMOVE BELOW LINE (.github/*) FROM THIS FILE AND SAVE BEFORE RUNNING AWS CICD PIPELINE ##
2
3  .github/*
4
5  ##### ONLY DELETE/REMOVE ABOVE LINE (.github/*). DO NOT CHANGE ANYTHING ELSE IN THIS FILE #####
6  #####
7  # Byte-compiled / optimized / DLL files
8  __pycache__/
9  *.py[cod]
10 *$py.class
11 # Folder -arindam
12 artifacts/project/*
13 artifacts/your/*
14 artifacts/downloads/*
15 artifacts/GOOGLE_FAISS_DB/*
16 artifacts/OPENAI_FAISS_DB/*
17 artifacts/FINETUNE_FAISS_DB/*
18 # C extensions
19 *.so
20
21 # Distribution / packaging
22 .Python
23 build/
24 develop-eggs/
25 dist/
26 downloads/
27 eggs/
28 .eggs/      You, 15 hours ago • First Project commit
29 lib/
30 lib64/
31 parts/
32 sdist/
33 var/
34 wheels/
35 share/python-wheels/
```

9.8 setup.py

This Python script is designed for setting up a Python package using setuptools. It includes a function to read and process a list of requirements from a `requirements.txt` file, and it uses this list in the package setup configuration

The screenshot shows a code editor interface with two tabs open: `setup.py` and `requirements.txt`. The `setup.py` file contains the following code:

```

1  from typing import List
2
3  HYPEN_E_DOT = '-e .' # This is the last line in requirements.txt file which triggers setup.py; we must remove this from the list
4
5  def get_requirements(file_path:str) -> List[str]:
6      """
7          this function will return the list of requirements needed for this project
8      """
9      requirements = []
10     with open(file_path) as file_obj:
11         requirements = file_obj.readlines()
12     requirements = [req.replace("\n", "") for req in requirements]
13
14     if HYPEN_E_DOT in requirements:
15         requirements.remove(HYPEN_E_DOT)
16
17     return requirements
18
19 ## Setup and User Info:
20
21 setup(
22     name='ShopChat',
23     description='A RAG based shopping assistance to help buyer on buying product from Amazon [Using ABO dataset]',
24     version='1.0',
25     author='Arindam Choudhury',
26     author_email='arindam.choudhury@gmail.com',
27     url='',
28     packages=find_packages(),
29     install_requires=get_requirements('requirements.txt'))

```

The `requirements.txt` file contains the following dependencies:

```

1 pillow==10.3.0
2 transformers==4.41.2
3 sentence_transformers==2.7.0
4 pandas==2.2.2
5 matplotlib==3.9.0
6 streamlit==1.36.0
7 streamlit_feedback==0.1.3
8 streamlit_option_menu==0.3.13
9 google-generativeai==0.5.2
10 python-dotenv==1.0.1
11 langchain==0.2.7
12 pydantic==4.2.0
13 chromadb==0.5.3
14 faiss-cpu==1.6.0
15 langchain_google_genai
16 langchain_openai==0.1.14
17 langchain_community==0.2.7
18 langchain_huggingface==0.0.3
19 langchain_groq==0.1.6
20 jq==1.7.0
21 numpy==1.26.4
22 matplotlib==3.9.0
23 s3fs==2024.6.1
24 sagemaker==2.22.4
25 llama_index_finetuning==0.1.18
26 llama_index==0.10.53
27 llama_index embeddings-huggingface==0.2.2
28 huggingface_hub==0.23.4
29 scikit-learn==1.5.1
30 boto3==1.34.131
31 -e .

```

9.9 requirements.txt

List of Python packages and dependencies to run the project

The screenshot shows a code editor interface with the `requirements.txt` file open. The file contains the following dependencies:

```

1 pillow==10.3.0
2 transformers==4.41.2
3 sentence_transformers==2.7.0
4 pandas==2.2.2
5 matplotlib==3.9.0
6 streamlit==1.36.0
7 streamlit_feedback==0.1.3
8 streamlit_option_menu==0.3.13
9 google-generativeai==0.5.2
10 python-dotenv==1.0.1
11 langchain==0.2.7
12 pydantic==4.2.0
13 chromadb==0.5.3
14 faiss-cpu==1.6.0
15 langchain_google_genai
16 langchain_openai==0.1.14
17 langchain_community==0.2.7
18 langchain_huggingface==0.0.3
19 langchain_groq==0.1.6
20 jq==1.7.0
21 numpy==1.26.4
22 matplotlib==3.9.0
23 s3fs==2024.6.1
24 sagemaker==2.22.4
25 llama_index_finetuning==0.1.18
26 llama_index==0.10.53
27 llama_index embeddings-huggingface==0.2.2
28 huggingface_hub==0.23.4
29 scikit-learn==1.5.1
30 boto3==1.34.131
31 -e .

```

10 Observations and Issues

There are many rooms for improvements here, given the time permits. We could have added other feature such as 3D, 360-degree view, geometry, material etc. but here we have taken only the images to create this app.

Better finetuned model, hyper parameter tuning, embedding finetuning etc. also could have been implemented if would have gotten more time.

Throughput could have been reduced with proper design of UI with better logical flow and data handling mechanism.

10.1 Finetune Embedding and LLM models

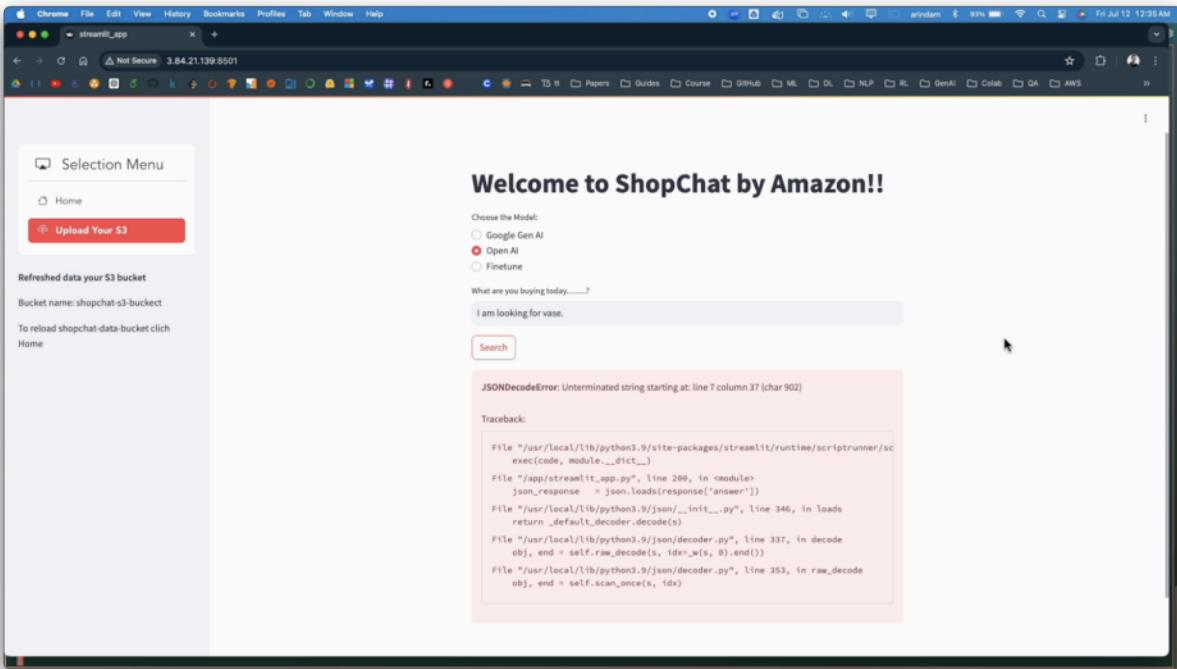
Synthetic questions generated by OpenAI model were not so buyer friendly and hence the finetuned model did not perform well on inference.

1. Synthetic questions generated by OpenAI model were not so buyer friendly and hence the finetuned model did not perform well on inference
2. Use human annotator to create better question-and-answer dataset
3. Create better questions and answers from the ABO data using better prompt
4. Try other Question-and-answer generation model to improve performance
5. Train for more epochs and more question-and-answer dataset
6. Try other Fine Model which can also improve on performance

10.2 Inference with OpenAI LLM model

Sometimes, we were facing issues with OpenAI model while inferencing, it cannot create the response in proper Json format due to the response context length.

Using proper prompting and hyper parameters tuning we can control the response and avoid getting the error. One of the examples shown below. Also added in working model video demo



10.3 Response Thruput

Both these LLM provides fast response and response time is captured and displayed in terminal while running the app. The overall response time is faster because of loading and unloading the full vector databases and operation time within Streamlit app and can be reduced by creating proper REST interface API for inference

10.4 Bucket selection

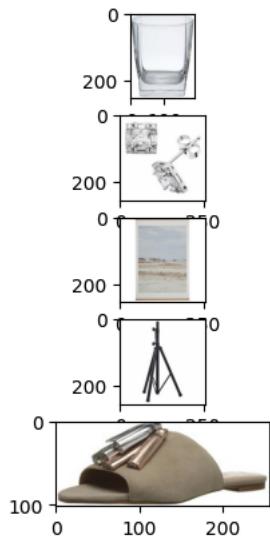
In the selection menu, after we change the S3 location, Streamlit loads the entire page including saving of models and vector database which loads fresh every time we change the bucket. This can also be avoided by creating proper REST interface API for inference

10.5 Image Caption

Captions generated by the Saleforce BLIP model is not trained to generate caption on the products. Hence the by adding caption to the dataset and doing embedding did not improve response performance. Few example of image captions are shown below:

	item_id	item_name_in_en_us	image_caption
0	B0896LJNLH	AmazonBasics Serene 16-Piece Old Fashioned and...	a close up of a glass of water on a white back...
1	B07HCR1LSQ	[Find] Amazon Collection Platinum Plated Sterl...	a pair of earrings with a square cut diamond a...
2	B075DQBBJZ	Arizona Desert Sand Horizon Photo with Wood Ha...	a white and brown photograph of a desert lands...
3	B07S74D9T7	AmazonBasics Adjustable Speaker Stand - 3.8 to...	a black tripod speaker stands on a white backg...
4	B01N27SMXC	Amazon Brand - The Fix Women's Foley Tassel Sl...	a close up of a pair of sandals with a metal b...

for the images:



11 Challenges

11.1 Data Handling

Handling large volumes of product data from multiple sources was challenging. Ensuring consistency and accuracy in the dataset required robust preprocessing and cleaning mechanisms and processing power.

11.2 Language and Localization

Filtering data to ensure language consistency (e.g., retaining only English text) was crucial and challenging, given the multilingual nature of some data fields.

11.3 Integration of AI Models

Integrating different AI models for generating embeddings and ensuring they operate efficiently under varying loads was complex.

11.4 Scalability and Performance

Ensuring the system could scale effectively to handle real-time user queries without latency was a significant technical hurdle.

11.5 User Experience Design

Designing a chat interface that was both intuitive and effective at guiding users through the product search process required iterative design and testing.

12 Future Improvements

12.1 Model Enhancement

Upgrading to more advanced versions of AI models like GPT-4 or other latest models for better understanding and generating more contextually relevant responses.

12.2 Multi-language Support

Expanding the system to support multiple languages would make the platform more accessible to a global audience.

12.3 Personalization

Integrating user preferences and purchase history to tailor recommendations more closely to individual users.

12.4 Visual Search Capabilities

Implementing image recognition and visual search functionalities could enhance the user experience by allowing queries based on images rather than text.

12.5 Analytics and Feedback Loop

Incorporating analytics to track user satisfaction and system performance, using this data to refine AI responses and improve the overall system.