



**VIT<sup>®</sup>**  
Vellore Institute of Technology  
(Deemed to be University under section 3 of UGC Act, 1956)

# INDUSTRIAL INTERNSHIP REPORT

STUDENT DETAILS	
Register No	21BAI1171
Name	GAYATRI SREERAJ
Programme	B.Tech CSE AI/ML
COMPANY DETAILS	
Name	Alamy Ltd.
Website	www.alamy.com
Address	Alamy, C-16, Gayatri Building, Technopark, Thiruvananthapuram, Kerala. PIN - 695 581
Duration of Internship: 18.10.2023 to 18.12.2023 (2 months)	



**VIT<sup>®</sup>**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

## VELLORE INSTITUTE OF TECHNOLOGY, CHENNAI

School of Computer Science and Engineering

### **Industrial Internship Approval Form (2023-2024)**

Programme Name : B.Tech CSE AI/ML

Register No : 21BAI1171

Student Name : GAYATRI SREERAJ

Company Name : Alamy Ltd.

Company Email ID : hrindia@alamy.com

Phone Number : +91 (0)471 2865200

Website Address : www.alamy.com

Company Address : Alamy, C-16, Gayatri Building, Technopark,  
Thiruvananthapuram, Kerala. PIN - 695 581

Duration : 2 months

Stipend Amount (if any) : NIL

A handwritten signature in black ink, appearing to read 'L Jeganathan'.

18.10.2023

Dr. Jeganathan L

Dr. Rama Prabha K P

Name & Signature of Proctor

Name & Signature of Professor-in-Charge

## APPROVAL:

**G** Gayatri Sreeraj 21BAI1171 <gayatri.sreeraj2021@vitstudent.ac.in>  
to Rama ▾

Dear Ma'am,

I have attached the offer letter email sent from the company.

Regards  
Gayatri

\*\*\*

One attachment • Scanned by Gmail ⓘ



**R** Rama Prabha K P <ramaprabha.kp@vit.ac.in>  
to me ▾

Approved..... Ensure that the internship is minimum for 28 days.....

Thanks and Regards,  
Dr.Rama Prabha  
Associate Professor,  
School of Computer Science & Engineering,  
VIT, Chennai.

\*\*\*

## OFFER LETTER:

**H** HR India <hrindia@alamy.com>  
to Suresh, me ▾  
Hi Gayathri

Further to our discussions, we are happy to start your internship for 2 months with us on 16<sup>th</sup> October as part of your studies. You can do internship remotely and report to Suresh Kumar M who will be your mentor.

Please confirm the acceptance so that we can initiate your account set-ups.

Thanks  
HR Team



w: <https://www.alamy.com> | t: +91 (0)471 2865200

a: Alamy, C-16, Gayatri Building, Technopark, Thiruvananthapuram, Kerala. PIN - 695 581

Follow us: [Blog](#), [Instagram](#), [LinkedIn](#), [Pinterest](#), [YouTube](#), [X](#)

# INTRODUCTION

## About Alamy Ltd.

Alamy (registered as Alamy Limited) is a British stock photography agency launched in September 1999. Its headquarters are in Milton Park, near Abingdon, Oxfordshire, United Kingdom. It has a development and operations centre at Technopark in Trivandrum, Kerala, India and a sales office in Brooklyn, New York, United States.

Alamy is an online supplier of stock images, videos, and other image material. Their content comes from agencies and independent photographers, or are collected from news archives, museums, national collections, and public domain content copied from Wikimedia Commons.

During my two-month industrial internship at Alamy, **Mr. Suresh Kumar M**, the Head of Technology at Alamy India, was my mentor for the assigned project.

## About the Project

My project involved creating a prototype for an image search engine that utilizes machine learning models, like OpenAI's CLIP (Contrastive Language-Image Pre-training). This search engine is equipped to receive inputs in both text and image formats. When provided with a keyword, the system sources images from the Alamy database by constructing URLs using the search results in XML format.

The core functionality relies on the machine learning model's capability to extract features from the input text or images. This prototype combines machine learning techniques with Alamy's extensive image collection. By conducting a similarity search between the feature vectors of the input and the feature vectors of the images sourced from Alamy, the system identifies the most relevant images within the Alamy database and displays them as search results.

## About OpenAI CLIP

CLIP stands for "Contrastive Language-Image Pre-training." It's a neural network model developed by OpenAI that learns by associating images and their descriptions in the form of text. Unlike previous models that focused solely on images or text separately, CLIP is trained to understand the relationship between images and text simultaneously.

CLIP is designed to understand semantic similarities between images and their descriptions in text, enabling it to perform various tasks. This ability makes CLIP a versatile and powerful tool for various applications in machine learning, computer vision, and natural language processing.

# PROJECT EXECUTION – WEEKLY TASKS

## Week 1: Defining the Problem Scope

This phase involves setting clear objectives for the project, exploring various image processing models, and choosing appropriate indexing and searching tools required for implementation.

### I. PROJECT OBJECTIVES

This project has the following key objectives:

1. Development of an image search engine capable of supporting diverse search criteria.
2. Integration of technology and models to provide efficient image retrieval.
3. Creation of a user-friendly interface for seamless interaction.

### II. COMPARISON OF MACHINE LEARNING MODELS

#### 1. CLIP (Contrastive Language-Image Pre-training) from OpenAI

OpenAI's CLIP (Contrastive Language-Image Pre-training) is a powerful neural network model. It's trained to understand and associate images and text together.

#### 2. ImageBind, from Meta AI

ImageBind, from Meta AI combines six modalities (images, text, audio, etc.) for cross-modal data retrieval. It allows querying in one modality and retrieving related documents from others. This facilitates building diverse applications like associating text with uploaded images.

CLIP is a more suitable model for this project.

### III. COMPARISON OF INDEXING TOOLS

1. **Elasticsearch** is a distributed search engine for storing and analyzing large data sets in real-time, suitable for various analytics and search applications, and can be hosted on an Amazon EC2 instance for scalability.

2. **Apache Solr** is an open-source search platform that facilitates efficient data indexing and retrieval, offering extensive customization options, and can also be hosted on an EC2 instance for handling search tasks.

3. **FAISS Index**, optimized for similarity searches and clustering of dense vectors, lacks direct hosting on an EC2 instance. It functions as a library and needs integration into systems for utilization due to its focus on similarity search tasks.

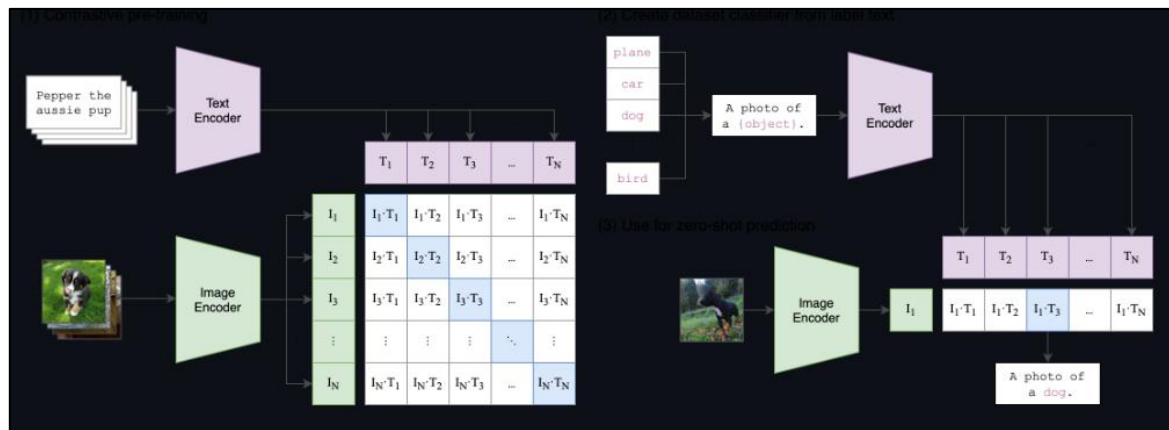
All these indexing tools are utilized in this project.

## Week 2: Exploring OpenAI CLIP and Indexing Tools

This phase involves understanding the functionalities of the OpenAI CLIP model, extracting features using CLIP.

In this project, the CLIP (Contrastive Language-Image Pretraining) model is utilized for extracting text and image features. Here are the key functionalities of the CLIP model used:

- 1. Text Feature Extraction:** The CLIP model processes text input and extracts embeddings representing the semantic features of the text. It chunks the input text into suitable sizes, processes each chunk separately, and concatenates the resulting text features.
- 2. Image Feature Extraction:** The CLIP model processes images and extracts embeddings representing the visual features of the images. It uses a Vision Transformer (ViT) backbone to encode the images.



## Week 3: Understanding Vector Similarity Methods and AWS Cloud Services

This phase involves learning and comprehending techniques such as cosine similarity, nearest neighbors, familiarizing oneself with AWS Cloud Services for potential deployment and scalability, and understanding its integration with Elasticsearch and Apache Solr.

### Methods to find similarity between vector embeddings:

- 1. Cosine similarity:** Measures the cosine of the angle between two vectors, providing a similarity score between -1 (completely dissimilar) to 1 (identical).
- 2. k-nearest neighbours (k-NN):** k-NN uses a distance metric to measure the closeness or similarity between data points.

This project utilizes k-NN method to find similarity.

### AWS Cloud Services:

In this project, an Amazon EC2 (Elastic Compute Cloud) instance is utilized to host the indexing tools such as Elasticsearch and Apache Solr. These services handle the indexing, storing, and searching of data, providing scalable solutions for data retrieval and analysis.

## Week 4: Acquiring Python Packages and Alamy Database Interaction

This phase involves exploring and obtaining necessary Python packages (like torch, transformers, Pillow, pysolr, elasticsearch, faiss-cpu, scikit-learn, smart-open), interacting with the Alamy database, and coding the retrieval process (alamy\_images.py) to fetch images using the image URLs.

The Python packages required for this project, and their respective versions are:

- torch - 2.1.1
- transformers - 4.32.1
- Pillow - 9.4.0
- pysolr - 3.9.0
- elasticsearch - 7.13.4
- faiss-cpu - 1.7.4
- scikit-learn - 1.3.0
- smart-open - 5.2.1

### alamy\_images.py

```
import requests
import xml.etree.ElementTree as ET

def fetch_alamy_image_urls(search_query):
    encoded_query = search_query.replace(' ', '%20')
    alamy_url = f"https://www.alamy.com/xml-search-
results.asp?qt={encoded_query}&pn=1&ps=1000"
    try:
        response = requests.get(alamy_url)
        if response.status_code == 200:
            xml_content = response.content
            tree = ET.ElementTree(ET.fromstring(xml_content))

            images_data = []
            image_urls = []
            root = tree.getroot()
            for image in root.findall('.//I'):
                ar_filename = image.get('AR')

                # Construct image URL using the AR identifier
                image_url =
f"https://c7.alamy.com/zooms/9/1/{ar_filename}.jpg"

                # Image data
                image_data = {
                    'ID': image.get('ID'),
                    'file_name': ar_filename,
                    'size': f"{image.get('PIX_X')} x {image.get('PIX_Y')}",
                    'type': image.get('TYPE'),
                    'caption': image.get('CAPTION'),
                    'date_taken': image.get('DATETAKEN')
                }
                images_data.append(image_data)
            # Returning the image URLs
            return image_urls, images_data
        else:
            return []
    except requests.RequestException:
        return []
```

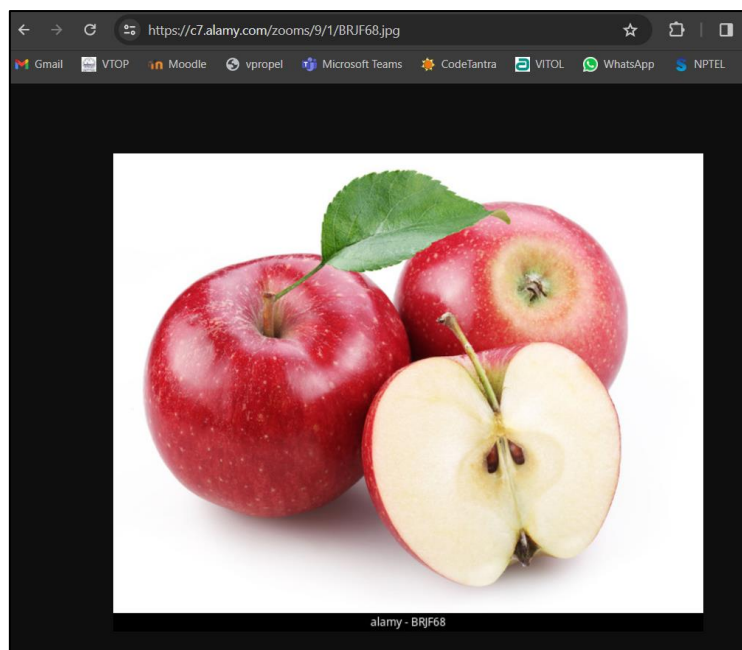
The functionality of `fetch_alamy_image_urls(search_query)` can be explained as follows:

1. Accepts a search query string.
2. [https://www.alamy.com/xml-search-results.asp?qt={encoded\\_query}&pn=1&ps=1000](https://www.alamy.com/xml-search-results.asp?qt={encoded_query}&pn=1&ps=1000)  
- This URL gives the search results from Alamy's XML search API based on the query.

For `search_query = apple`

```
▼<IMAGES>
<I ID="{B46ABC87-5BDF-4727-91FA-6C0E2E83BDB7}" AR="FJ1MB1" LI=
DATETAKEN="20091015" P="11630" T="47" FLP="0" DATACO="0" IMGSE
<I ID="{550DE376-82E0-4427-9710-4A059E84B028}" AR="E5DD6M" LI=
FLP="0" DATACO="0" IMGSEQ="72210508" pseudoid="4C11CC9A1366416
<I ID="{D6BA8B5F-3D3D-4B34-9E05-371A2FE554DF}" AR="BRJF68" LI=
background." LC="en" DATETAKEN="20100929" P="47331" T="60" FLP
asstype="uncut"/>
```

3. Sends a GET request to the constructed URL and checks for a successful response.
4. Parses the XML content received from the response.
5. Finds elements with the tag name 'I' (representing image information) in the XML tree. Iterates through these elements to extract the 'AR' attribute, which serves as the image identifier.
6. Constructs image URLs using these identifiers and a specified URL structure – [https://c7.alamy.com/zooms/9/1/{ar\\_filename}.jpg](https://c7.alamy.com/zooms/9/1/{ar_filename}.jpg)
7. Collects metadata attributes (ID, file name, size, type, caption, date taken) for each image and organizes them into dictionaries.
8. Returns two lists: `image_urls` containing the constructed image URLs and `images_data` containing dictionaries of metadata for each image.





## Week 5: Implementing CLIP Feature Extraction, Similarity Comparison and Data Processing

This stage involves coding different modules like `clip_feature_extractor.py`, `similarity.py`, `utils.py`, and `process_data.py` for data handling and analysis.

### `clip_feature_extractor.py`

```
import torch
from transformers import CLIPModel, CLIPProcessor

class CLIPFeatureExtractor:

    def __init__(self):
        model_name = "openai/clip-vit-base-patch32"
        self.model = CLIPModel.from_pretrained(model_name)
        self.processor = CLIPProcessor.from_pretrained(model_name)
        self.device = "cuda" if torch.cuda.is_available() else "cpu"
        self.model.to(self.device)

    @torch.no_grad()
    def get_text_features(self, text):
        max_chunk_length = 77
        # Maximum sequence length supported by the model

        # Chunk the input text

        text_chunks = [text[i:i + max_chunk_length] for i in range(0,
len(text), max_chunk_length)]

        # Process each text chunk separately and concatenate the results
        concatenated_text_features = []

        for chunk in text_chunks:
            inputs = self.processor(text=chunk, return_tensors="pt")
            inputs = inputs.to(self.device)
            text_features = self.model.get_text_features(**inputs)
            text_features /= text_features.norm(dim=-1, keepdim=True)
            text_features = text_features.tolist()
            concatenated_text_features.extend(text_features)

        return concatenated_text_features

    @torch.no_grad()
    def get_image_features(self, images):
        inputs = self.processor(images=images, return_tensors="pt")
        inputs = inputs.to(self.device)
        image_features = self.model.get_image_features(**inputs)
        image_features /= image_features.norm(dim=-1, keepdim=True)
        image_features = image_features.tolist()

        return image_features
```

The functionality of this code can be explained as follows:

### 1. Initialization (`__init__`):

- Loads the CLIP model and processor for feature extraction.
- Determines the device (GPU/CPU) for computation.

### 2. Text Feature Extraction (`get_text_features`):

- Splits text input into smaller chunks to fit the model's sequence length.
- Processes each chunk separately, retrieves text features, normalizes them, and concatenates the results.

### 3. Image Feature Extraction (`get_image_features`):

- Processes input images and retrieves image features.
- Normalizes them, and returns the results.

## similarity.py

```
from sklearn.neighbors import KNeighborsRegressor
import numpy as np
import torch

def calculate_similarity(embedding1, embedding2, k=2):
    embedding1 = torch.Tensor(embedding1)
    embedding2 = torch.Tensor(embedding2)
    # Reshape embeddings if needed
    if len(embedding1.shape) == 1:
        embedding1 = embedding1.unsqueeze(0)
    if len(embedding2.shape) == 1:
        embedding2 = embedding2.unsqueeze(0)
    # Ensure both embeddings have the same number of samples
    max_samples = max(embedding1.shape[0], embedding2.shape[0])
    if embedding1.shape[0] < max_samples:
        embedding1 = torch.cat([embedding1] * (max_samples //
embedding1.shape[0]) + [embedding1[:max_samples % embedding1.shape[0]]])
    elif embedding2.shape[0] < max_samples:
        embedding2 = torch.cat([embedding2] * (max_samples //
embedding2.shape[0]) + [embedding2[:max_samples % embedding2.shape[0]]])
    # Combine embeddings into a single array
    combined_embeddings = np.vstack((embedding1.detach().numpy(),
embedding2.detach().numpy()))
    # Create KNN model
    knn_model = KNeighborsRegressor(n_neighbors=k)
    # Fit the KNN model
    knn_model.fit(combined_embeddings, [0] * embedding1.shape[0] + [1] *
embedding2.shape[0])
    # Predict distances
    distances, indices = knn_model.kneighbors(combined_embeddings,
n_neighbors=k)
    # Calculate similarity score as inverse of the mean distance if mean
distance is not zero
    mean_distance = np.mean(distances)
    similarity_score = 1 / mean_distance if mean_distance != 0 else 0.0
    return similarity_score
```

The functionality of this code is as follows:

1. Converts embedding1 and embedding2 (given as input arrays) into PyTorch tensors.
2. Reshapes the tensors to ensure consistent shapes for computation.
3. Combines both embeddings into a single array (combined\_embeddings) using NumPy.
4. Initializes a KNN model (knn\_model) with a specified number of neighbors (k).
5. Uses the trained model to find the k nearest neighbors and their distances for each sample in combined\_embeddings.
6. Computes the mean distance among nearest neighbors (mean\_distance) and calculates a similarity score as the inverse of the mean distance. If the mean distance is zero, the similarity score is set to 0.0 to avoid division by zero.
7. Returns the computed similarity score.

#### utils.py

```
import smart_open
from PIL import Image

def load_image_from_url(image_url):
    with smart_open.open(image_url, "rb") as image_file:
        return pil_loader(image_file)

def pil_loader(image_file):
    with Image.open(image_file) as image:
        return image.convert("RGB")
```

The functionality of this code is as follows:

#### 1. load\_image\_from\_url(image\_url):

- Uses smart\_open.open to open the image file from the provided image\_url in read-binary ("rb") mode.
- Calls the pil\_loader function with the opened image file and returns the processed image.

#### 2. pil\_loader(image\_file):

- Opens the image file using Image.open from PIL, passing the file object (image\_file).
- Processes the image by converting it to the RGB color mode using .convert("RGB").
- Returns the processed image object.

## process\_data.py

```
from concurrent.futures import ThreadPoolExecutor
from clip_image_search.clip_feature_extractor import CLIPFeatureExtractor
from clip_image_search.similarity import calculate_similarity
from clip_image_search.utils import load_image_from_url

# Initialize CLIP feature extractor
clip_extractor = CLIPFeatureExtractor()

# Function to process image data and perform similarity ranking
def clip_process_data(search_query, image_urls, images_data):
    image_embeddings = []
    caption_embeddings = []
    list_of_urls_and_features = []

    def process_image(image_url, image_info):
        try:
            image_data = load_image_from_url(image_url)

            if image_data is not None:
                # Extract embeddings using CLIP feature extractor for the
                # current image
                image_embedding =
clip_extractor.get_image_features(image_data)
                image_embeddings.append(image_embedding)
                t = (image_url, image_embedding)
                list_of_urls_and_features.append(t)

                # Extract embeddings for image captions from images_data
                caption = image_info['caption']
                caption_embedding = clip_extractor.get_text_features(caption)
                caption_embeddings.append(caption_embedding)
            else:
                # Handle the case when image data loading fails
                print(f"Failed to load image data from URL: {image_url}")
        except Exception as e:
            # Handle exceptions related to image data loading or processing
            print(f"Error processing image from URL {image_url}: {str(e)}")

    # Use ThreadPoolExecutor to process images concurrently
    with ThreadPoolExecutor(max_workers=10) as executor:
        for image_url, image_info in zip(image_urls, images_data):
            executor.submit(process_image, image_url, image_info)

    # Get text features (embedding) for the search query using CLIP feature
    # extractor
    text_embedding = clip_extractor.get_text_features(search_query)

    # Calculate similarity scores between text embedding and image/caption
    # embeddings
    similarity_scores = []
    for img_emb, cap_emb in zip(image_embeddings, caption_embeddings):
        # Calculate similarity scores for image embeddings and caption
        # embeddings
        similarity_score_img = calculate_similarity(img_emb, text_embedding)
        similarity_score_caption = calculate_similarity(cap_emb,
text_embedding)
```

```

        # Combine the similarity scores from image and caption
        similarity_scores.append((similarity_score_img +
similarity_score_caption) / 2)

    # Rank images based on similarity scores
    ranked_images = [url for _, url in sorted(zip(similarity_scores,
image_urls), reverse=True)]
    return ranked_images, list_of_urls_and_features

```

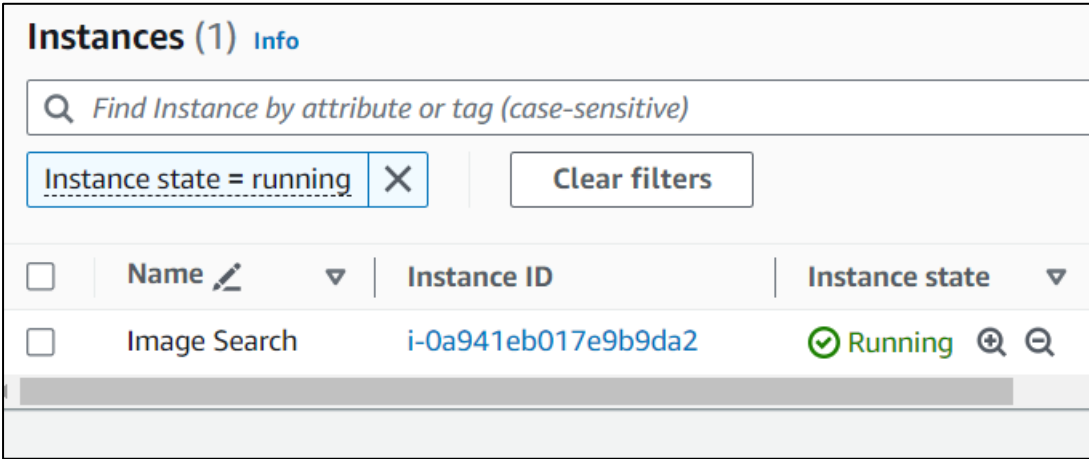
The functionality of this code can be explained as follows:

1. Initializes a CLIPFeatureExtractor object named clip\_extractor for extracting features from images and text using the CLIP model.
2. Defines a function to process image data and perform similarity ranking based on a search query.
3. Utilizes ThreadPoolExecutor to concurrently process images using multiple threads for improved efficiency (max\_workers=10). Processes individual image data asynchronously.
4. Loads image data from provided URLs using load\_image\_from\_url and handles exceptions if image loading fails.
5. Extracts embeddings (features) using CLIP for images and their associated captions.
6. Stores image URLs and their respective embeddings in list\_of\_urls\_and\_features.
7. Retrieves text features (embeddings) for the search query using the CLIP feature extractor.
8. Calculates similarity scores between the text query and the image/caption embeddings.
9. Combines similarity scores for images and captions and averages them to obtain a final similarity score for ranking.
10. Ranks images based on their similarity scores, with higher scores indicating better matches to the search query.
11. Returns a list of ranked image URLs along with their corresponding feature embeddings.

## Week 6: Launching Amazon EC2 Instance and Data Indexing

The phase involves launching an EC2 instance and implementing data indexing procedures (index\_data.py) using Elasticsearch, Apache Solr and FAISS Index for efficient search and retrieval of images.


Amazon EC2 instance:



On port 9200 of Amazon EC2 instance: Elasticsearch service

```
{
  "name" : "EC2AMAZ-6EL50FJ",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "07EfdaKOT2qp1xFoYjAsxw",
  "version" : {
    "number" : "8.11.3",
    "build_flavor" : "default",
    "build_type" : "zip",
    "build_hash" : "64cf052f3b56b1fd4449f5454cb88aca7e739d9a",
```

On port 8984 of Amazon EC2 instance: Apache Solr service



Dashboard

Logging

Security

Core Admin

Java Properties

Thread Dump

	<b>solr-spec</b>	8.11.2
	<b>solr-impl</b>	8.11.2 17dee71932c683e345508113523e
	<b>lucene-spec</b>	8.11.2
	<b>lucene-impl</b>	8.11.2 17dee71932c683e345508113523e

JVM

**Runtime**

Oracle Corporation Java HotSpot(TM) 64-Bit Server VM

**Processors**

2

**Args**

-DSTOP.KEY=solrrocks  
-DSTOP.PORT=7984  
-Diava.io.tmpdir=C:\Users\Administrator\

## index\_data.py

```
import pysolr
from elasticsearch import Elasticsearch
import numpy as np
import faiss
from scripts.alamy_images import fetch_alamy_image_urls
from clip_image_search.clip_feature_extractor import CLIPFeatureExtractor
from clip_image_search.process_data import clip_process_data

# Initialize Elasticsearch client
es = Elasticsearch([{'host': 'ec2-16-170-159-232.eu-north-1.compute.amazonaws.com', 'port': 9200}])

# Establish a connection to your Solr instance
solr = pysolr.Solr('http://ec2-16-170-159-232.eu-north-1.compute.amazonaws.com:8984/solr/features/', timeout=10)

# Indexing data into Elasticsearch
def index_data(keyword, results):
    es.index(index='image_search_index', body={'keyword': keyword, 'results': results})

def search_and_index_new_keyword(keyword):
    existing_doc = es.search(index='image_search_index', body={'query': {'match': {'keyword': keyword}}})
    if existing_doc['hits']['total']['value'] == 0:
        # Keyword doesn't exist in the index, perform a new search and index the results
        img_urls, img_data = fetch_alamy_image_urls(keyword)
        new_results, list_of_urls_and_features = clip_process_data(keyword, img_urls, img_data) # Fetch search results using process_data.py
        for t in list_of_urls_and_features:
            features_str = ';'.join(', '.join(str(f) for f in feature) for feature in t[1])
            doc = {
                'id': t[0],
                'image_features': features_str
            }
            solr.add([doc])
            index_data(keyword, new_results) # Index keyword and results
        return new_results
    else:
        # Keyword already exists in the index, fetch and display the existing results
        existing_results = existing_doc['hits']['hits'][0]['_source']['results']
        return existing_results

def convert_str_to_list(features_str):
    return [list(map(float, feature.split(','))) for feature in features_str]

# Function to normalize vectors
def normalize(v):
    norm = np.linalg.norm(v)
    if norm == 0:
        return v
    return v / norm
```

```

# Function to perform image search based on uploaded image
def perform_image_search(uploaded_image):
    # Initialize CLIP feature extractor
    clip_extractor = CLIPFeatureExtractor()

    total = solr.search('*:*', rows=0)
    # Get the total count of documents
    total_docs = total.hits

    # Extract features of the uploaded image
    uploaded_image_features =
clip_extractor.get_image_features([uploaded_image])
    uploaded_image_np = np.array(uploaded_image_features).astype('float32')

    results = solr.search('*:*', fl='id,image_features', rows = total_docs)

    image_urls = []
    image_features = []
    for result in results:
        image_urls.append(result['id'])
        image_features.append(convert_str_to_list(result['image_features']))

    # Convert features to a suitable numpy array for FAISS and normalize
    image_features_array = np.array(image_features).astype('float32')
    image_features_normalized = np.vstack([normalize(vec) for vec in
image_features_array])

    # Build an index for FAISS
    index1 = faiss.IndexFlatIP(image_features_normalized.shape[1]) # Using
Inner Product (IP) index
    index1.add(image_features_normalized)

    # Normalize the uploaded image features
    uploaded_image_normalized = normalize(uploaded_image_np[0])

    # Perform a similarity search using FAISS
    k = 40 # Number of similar images to retrieve
    uploaded_image_normalized = np.expand_dims(uploaded_image_normalized,
axis=0)
    distances, indices = index1.search(uploaded_image_normalized, k)

    similar_images_urls = []
    for i in indices[0]:
        similar_images_urls.append(image_urls[i])

    return similar_images_urls

def perform_image_search_for_keyword(uploaded_image, img_urls):
    # Initialize CLIP feature extractor
    clip_extractor = CLIPFeatureExtractor()

    # Extract features of the uploaded image
    uploaded_image_features =
clip_extractor.get_image_features([uploaded_image])
    uploaded_image_np = np.array(uploaded_image_features).astype('float32')

    l = len(img_urls)

```



```

# Fetch features for all URLs
image_features = []
for url in img_urls:
    result = solr.search('id:"{}"'.format(url), fl='image_features')
    for res in result:
        image_features.append(convert_str_to_list(res['image_features']))

# Convert features to a suitable numpy array for FAISS
image_features_array = [item for sublist in image_features for item in
sublist]
image_features_np = np.array(image_features_array).astype('float32')

# Build an index for FAISS
index2 = faiss.IndexFlatL2(image_features_np.shape[1])
index2.add(image_features_np)

# Perform a similarity search using FAISS
k = 1 # Number of similar images to retrieve
distances, indices = index2.search(uploaded_image_np, k)

similar_images_urls = []
for i in indices[0]:
    similar_images_urls.append(img_urls[i])

return similar_images_urls

```

The functionality of this code is as follows:

1. **index\_data(keyword, results)** indexes the image search results (image URLs), along with the respective keyword into Elasticsearch.

2. The **search\_and\_index\_new\_keyword(keyword)** function is called when the user input is a "Text" query. It checks if that keyword exists in the Elasticsearch index.

If the keyword already exists, it retrieves the existing results. If the keyword does not exist, it performs a search using the `fetch_alamy_image_urls(keyword)` function, and indexes the keyword and URLs in Elasticsearch.

It also computes the vector embeddings of each image and indexes it into Solr, along with the respective URL, so that the features do not need to be recomputed.

3. The **perform\_image\_search(uploaded\_image)** function is called when the user input is a "Image" query. The image embeddings in the Solr index are retrieved and indexed into a FAISS Index. The function then performs an image similarity search based on an uploaded image using FAISS. The most similar images are returned.

4. The **perform\_image\_search(uploaded\_image)** function is called when the user input is a "Text + Image" query. It retrieves similar images based on the features of both the uploaded image and the entered text using FAISS.

5. The **convert\_str\_to\_list(features\_str)** function converts a string representation of image features to a list format.

6. The **normalize(v)** function normalizes vectors by dividing them by their Euclidean norm.

## Week 7: Creating a Streamlit Application, Testing, Optimization, and Code Refinement

This phase involves developing a user-friendly Streamlit application, testing, optimizing for improved speed, and refining the codebase based on the test results.

### streamlit\_app.py

```
import streamlit as st
from PIL import Image as PILImage
from io import BytesIO
import base64
import requests
import time
from scripts.index_data import search_and_index_new_keyword
from scripts.index_data import perform_image_search
from scripts.index_data import perform_image_search_for_keyword

# Streamlit UI
def main():
    st.title("CLIP Image Search Engine")
    st.write("This search engine uses CLIP (Contrastive Language-Image Pretraining) to perform image searches based on text input, image input, or a combination of both. You can upload an image or enter text queries to find relevant images.")

    # Sidebar for user input
    st.sidebar.header("Search Options")
    search_option = st.sidebar.selectbox("Choose Search Option", ["Text Input", "Image Input", "Text + Image Input"])

    if search_option == "Text Input":
        text_query = st.sidebar.text_input("Enter Text Query")
        if st.sidebar.button("Search"):
            if len(text_query)>0:
                st.write("Performing search based on text input...")
                start_time = time.time() # Record start time
                img = search_and_index_new_keyword(text_query)
                end_time = time.time() # Record end time
                st.write(f"Time taken: {int(end_time - start_time)} seconds") # Display time taken
                # Display images on the main section
                display_images(img)
            else:
                st.warning("Please enter a text.")

    elif search_option == "Image Input":
        uploaded_file = st.sidebar.file_uploader("Upload Image", type=['png', 'jpg', 'jpeg'])
        if st.sidebar.button("Search"):
            if uploaded_file is not None:
                st.write("Performing search based on uploaded image...")
                start_time = time.time() # Record start time
                uploaded_image = PILImage.open(uploaded_file)
                img = perform_image_search(uploaded_image)
                end_time = time.time() # Record end time
```

```

        st.write(f"Time taken: {int(end_time - start_time)}
seconds") # Display time taken
        # Display images on the main section
        display_images(img)
    else:
        st.warning("Please upload an image.")

    elif search_option == "Text + Image Input":
        text_query = st.sidebar.text_input("Enter Text Query")
        uploaded_file = st.sidebar.file_uploader("Upload Image", type=['png',
'jpg', 'jpeg'])
        if st.sidebar.button("Search"):
            if (uploaded_file is not None) and (len(text_query)>0):
                st.write("Performing search based on text and image input...")
                start_time = time.time() # Record start time
                uploaded_image = PILImage.open(uploaded_file)
                img = search_and_index_new_keyword(text_query)
                result = perform_image_search_for_keyword(uploaded_image, img)
                end_time = time.time() # Record end time
                st.write(f"Time taken: {int(end_time - start_time)}
seconds") # Display time taken
                # Display images on the main section
                display_images(result)
            else:
                if len(text_query)==0:
                    st.warning("Please enter a text.")
                if uploaded_file is None:
                    st.warning("Please upload an image.")

def crop_image_caption(image, percentage):
    width, height = image.size
    crop_height = int(height * percentage)
    cropped_image = image.crop((0, 0, width, height - crop_height))
    return cropped_image

def pil_to_b64(image):
    buffered = BytesIO()
    image.save(buffered, format="PNG")
    return base64.b64encode(buffered.getvalue()).decode()

def display_images(images):
    st.header("Search Results")
    if not images:
        st.write("No results found.")
    else:
        columns = 2 # Number of columns in the grid
        image_size = 300 # Set the size of images (adjust as needed)
        caption_percentage = 0.2 # Percentage of image height to be cropped
from the bottom

        for i in range(0, len(images), columns):
            col_images = images[i:i + columns]
            col1, col2 = st.columns(2)
            for url in col_images:
                response = requests.get(url)
                if response.status_code != 200:
                    continue
                uploaded_image = PILImage.open(requests.get(url,
stream=True).raw)

```

```

        cropped_image = crop_image_caption(uploaded_image,
caption_percentage)
        cropped_b64 = pil_to_b64(cropped_image)
        img = f'<a href="{url}" target="_blank"></a>'
        col1.markdown(img, unsafe_allow_html=True)

        # Display second image in the second column
        if len(col_images) == 1:
            col2.markdown("", unsafe_allow_html=True)
        else:
            col_images.pop(0)
            url = col_images[0]
            uploaded_image = PILImage.open(requests.get(url,
stream=True).raw)
            cropped_image = crop_image_caption(uploaded_image,
caption_percentage)
            cropped_b64 = pil_to_b64(cropped_image)
            img = f'<a href="{url}" target="_blank"></a>'
            col2.markdown(img, unsafe_allow_html=True)
if __name__ == "__main__":
    main()

```

## COMPONENTS:

1. **Streamlit UI:** Constructs a Streamlit interface for users to interact with the image search engine.

2. **Search Options:**

Allows users to select different search modes:

- Text Input
- Image Input
- Text + Image Input3

3. **Search Actions:**

### Text Input:

- Takes a text query and performs an image search based on the entered text.
- Utilizes the `search_and_index_new_keyword` function.

### Image Input:

- Allows users to upload an image and performs a search based on the uploaded image.
- Uses the `perform_image_search` function.

### Text + Image Input:

- Combines text and image queries to perform a search based on both inputs.
- Utilizes `perform_image_search_for_keyword` functions.

#### 4. Image Display:

- Displays search results in a grid layout.
- Utilizes HTML image tags for display with base64 encoding for image conversion.

#### 5. Functions:

- **crop\_image\_caption(image, percentage):** Crops an image's bottom part based on the given percentage.
- **pil\_to\_b64(image):** Converts a PIL image to a base64 encoded string.
- **display\_images(images):** Renders the images in the Streamlit UI, arranging them in a grid format with clickable links to original images

### TESTING INSIGHTS:

#### 1. Error Handling for Unsuccessful Responses:

Implemented handling for cases where a URL did not provide a successful response. This helps manage situations where images cannot be retrieved or loaded, ensuring a smoother user experience by gracefully handling failed requests.

#### 2. Optimization via Reduced HTTP Requests:

Reduced the number of HTTP requests, resulting in improved speed. Minimizing the number of requests to external resources (such as remote image URLs) can significantly enhance the overall performance and responsiveness of the application.

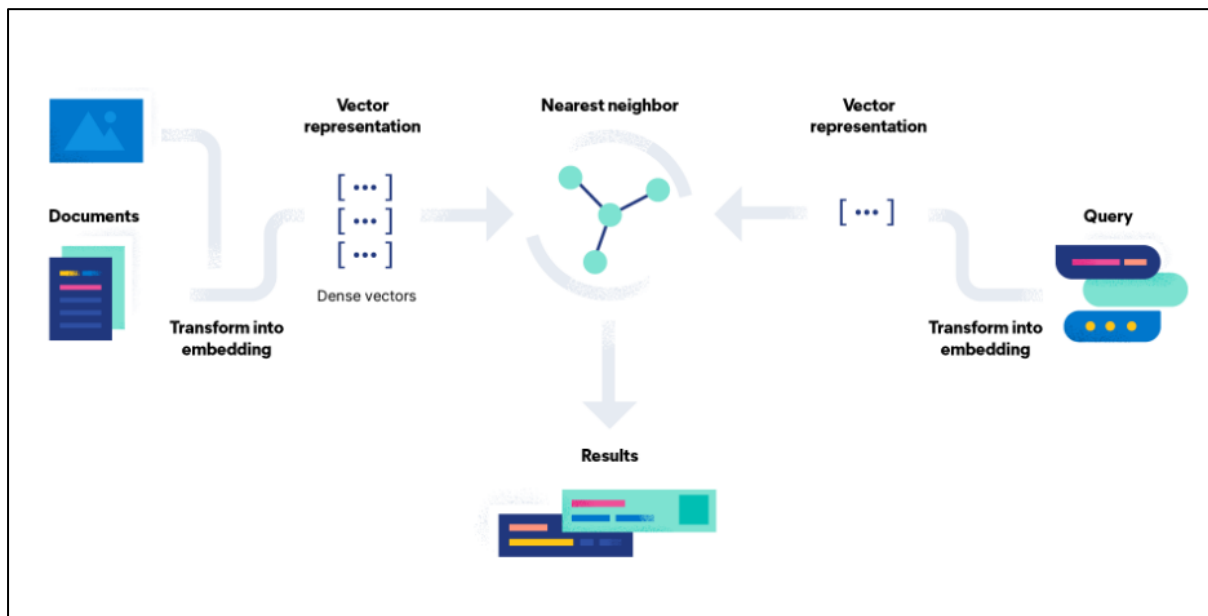
#### 3. Efficiency Enhancement through Indexing Image Embeddings:

Indexed image embeddings, which helped in improving speed as the features did not need to be recomputed for subsequent searches. Storing and reusing precomputed image features or embeddings can lead to faster retrieval times, especially in scenarios where feature extraction is computationally expensive.

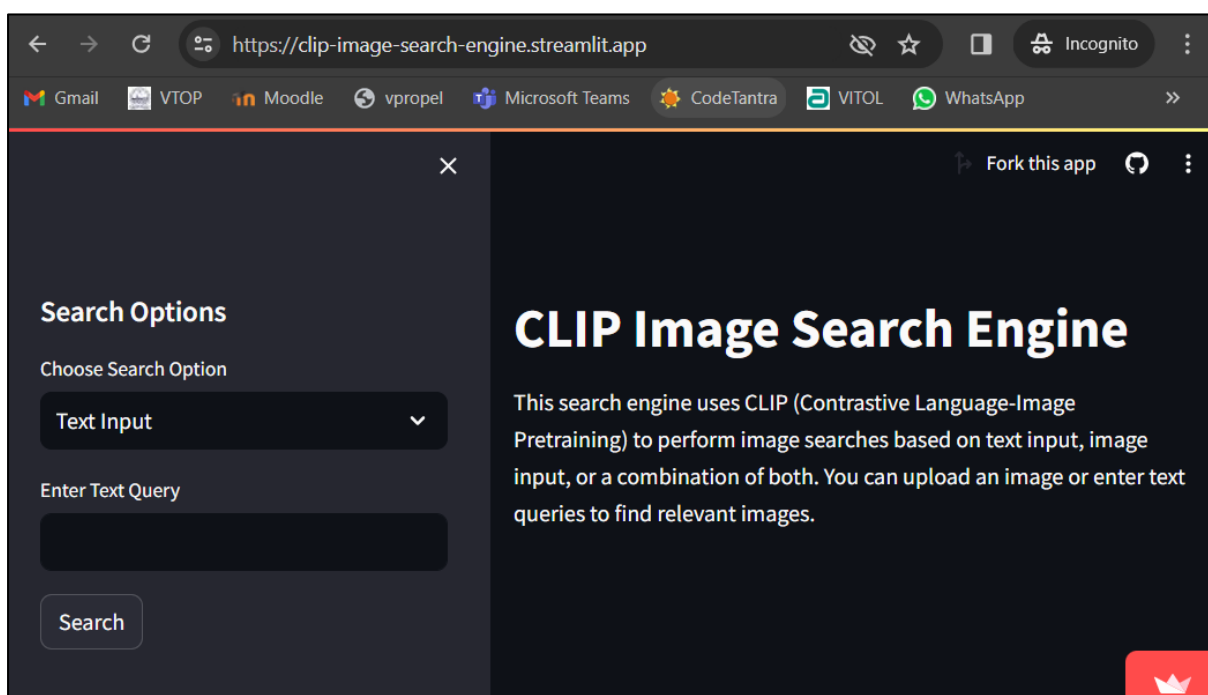
## Week 8: Final Project Demonstration and Preparing for Internship Certificate

The phase involves the final project demonstration, presenting the completed project to the mentor for evaluation, making necessary adjustments based on feedback, and preparing for the internship certificate issuance.

### PROCESS OVERVIEW:



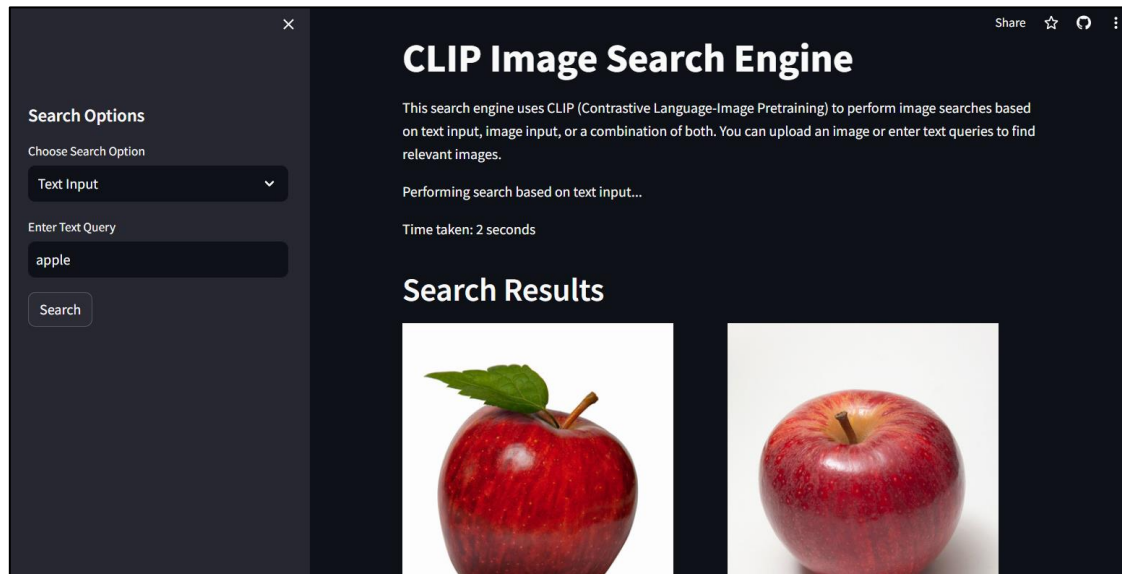
### STREAMLIT APP UI:



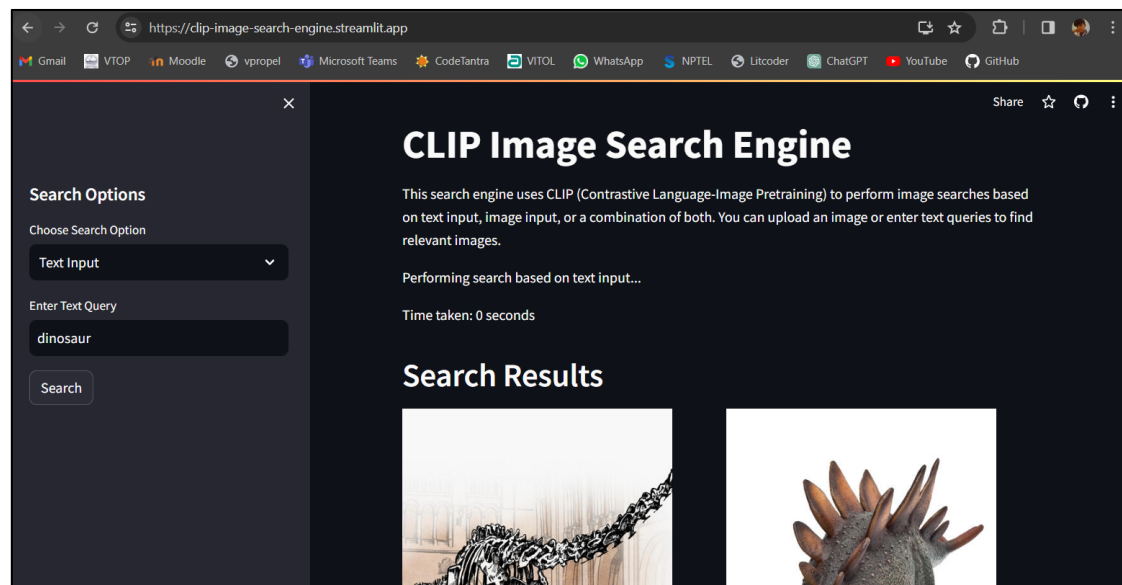
## SAMPLE INPUT-OUTPUT:

### 1. Text Input:

keyword = “apple”



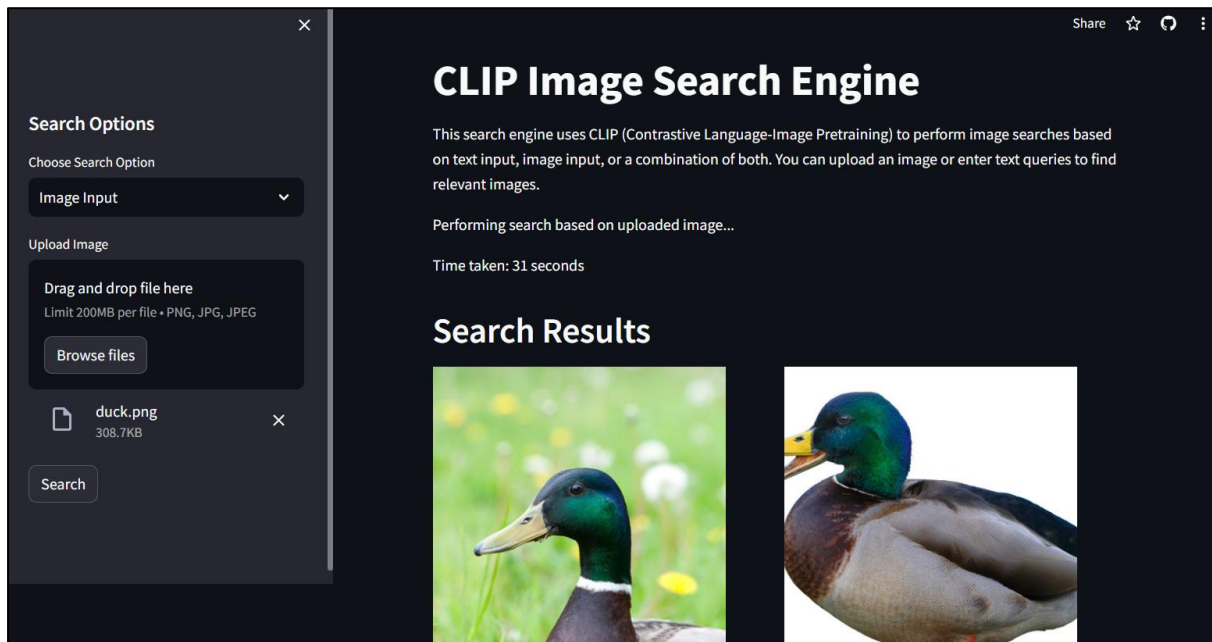
keyword = “dinosaur”



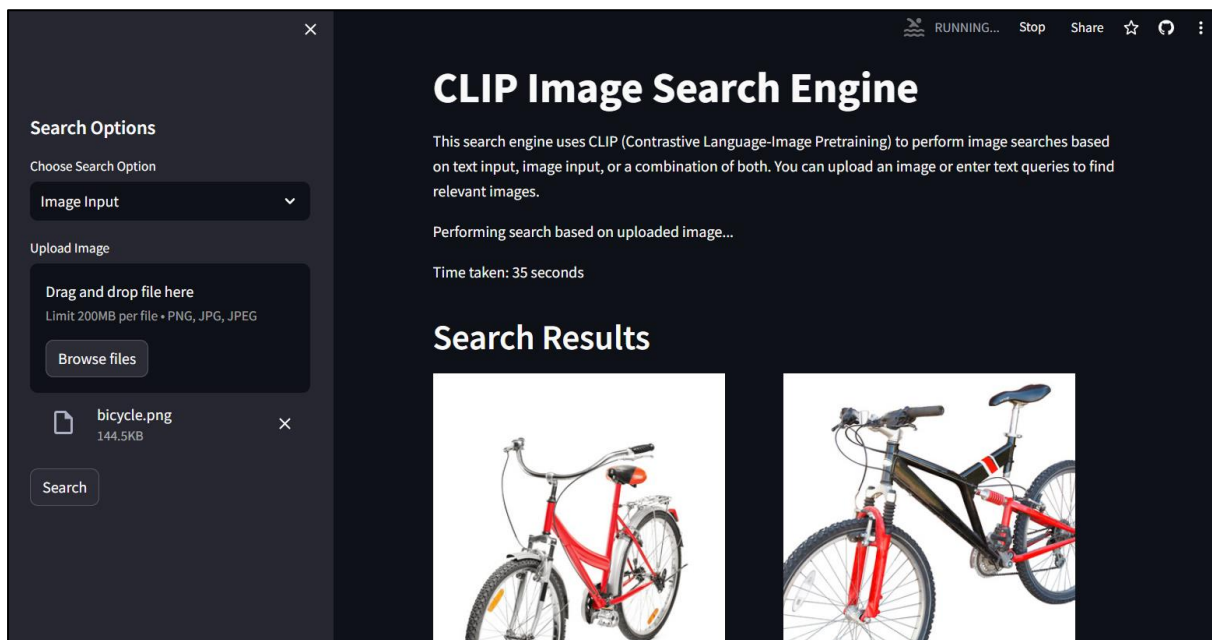
### 2. Image Input:

uploaded image = duck.png





uploaded image = bicycle.png

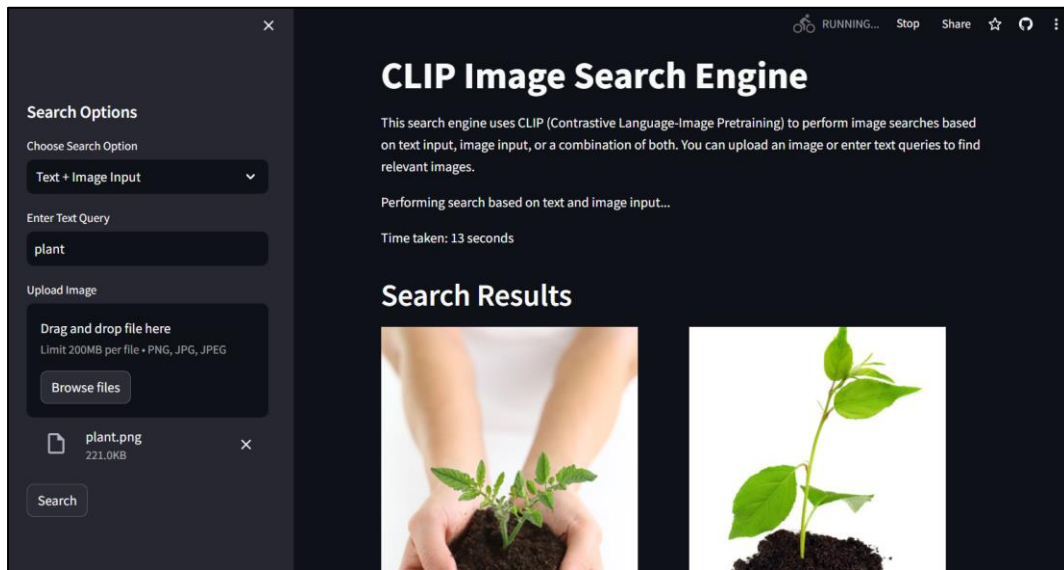




### 3. Text + Image Input:

keyword = “plant”

uploaded image = plant.png



**Github project link:**

<https://github.com/gayatrisreeraj/clip-image-search-engine>

**Streamlit app link:**

<https://clip-image-search-engine.streamlit.app>

**References:**

<https://openai.com/research/clip>

<https://github.com/openai/CLIP>

<https://www.elastic.co/blog/implement-image-similarity-search-elastic>

# CONCLUSION

The project undertaken during the industrial internship at Alamy Limited demonstrates an approach to enhance the functionality of an image search engine by integrating machine learning techniques, specifically utilizing OpenAI's CLIP model.

The objective was to create a prototype capable of conducting searches based on both text and image inputs, leveraging CLIP's ability to extract meaningful features from these different data types. By utilizing machine learning models and similarity search algorithms, the system sought to match input queries—whether in text or image format—with the most relevant images stored in Alamy's extensive image database.

Amidst this endeavor, I wish to express my deepest gratitude to Mr. Suresh Kumar M, Head of Technology at Alamy India, for his mentorship and guidance throughout the duration of this project.