Image Captioning and Sentiment Analysis App deployment with AWS EC2

B. TECH SEM – VI Cloud Computing Lab Project
Dept. of Computer Science & Engineering

By

Hardik Inani	**20BCP012**
Yash Prajapati	**20BCP021**
Mihir Sheth	**20BCP002**

Under the Supervision Of

Dr Manish Paliwal

SCHOOL OF TECHNOLOGY
PANDIT DEENDAYAL ENERGY UNIVERSITY
GANDHINAGAR, GUJARAT, INDIA

May 2023

ABSTRACT

As the volume of multimedia content on the internet continues to grow exponentially, the need for efficient tools to interpret and process this data has become increasingly important. Image captioning and sentiment analysis are essential techniques that facilitate the extraction of meaningful information from visual and textual content. This project report presents a detailed strategy for deploying an image captioning and sentiment analysis app on the Amazon Web Services (AWS) Elastic Compute Cloud (EC2) platform.

The developed app employs cutting-edge deep learning models for image captioning, such as the Show and Tell model, and sentiment analysis, including the BERT-based model. By incorporating these advanced models, the app is capable of generating accurate and contextually relevant image captions while concurrently determining the sentiment present in text content. Integrating these models into a single app streamlines the processing of multimedia data, enriching user experience and enhancing content analysis efficiency.

For deployment, the app utilizes the AWS EC2 platform, which delivers scalable and adaptable computing resources. Through the use of AWS EC2 instances, the app effectively manages fluctuating workloads and meets diverse user needs without sacrificing performance or incurring excessive costs. In addition, the integration of AWS services such as Amazon S3 for data storage and AWS Lambda for serverless computing ensures seamless and reliable data management throughout the app's operations.

To evaluate the app's effectiveness, we conducted a series of tests using the Flickr30k dataset for image captioning and the IMDb dataset for sentiment analysis. The results demonstrate the app's ability to produce high-quality image captions and accurately identify sentiment in textual content. Moreover, the app's performance remains consistent across various EC2 instance types, highlighting the flexibility and scalability offered by the AWS platform.

In conclusion, this project report showcases the successful implementation of an image captioning and sentiment analysis app on the AWS EC2 platform. By combining state-of-the-art deep learning models with AWS's robust infrastructure, the app delivers outstanding performance and adaptability for multimedia content processing. The presented approach offers valuable insights for researchers and practitioners aiming to develop and deploy similar applications in a cloud environment and lays the foundation for future advancements in multimedia analysis.

INDEX

Developing Machine Learning Model

1. Flickr 30k Image Captioning Model

Download the Flick30k dataset from Kaggle

Create a Google Collaboratory file and perform the following code:

```
# -*- coding: utf-8 -*-
"""image-captioning.ipynb

Automatically generated by Colaboratory.

Original file is located at

https://colab.research.google.com/drive/1dN67i3ZiwD8R63kiqWynL3pdvm7UHXn3

# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python
# For example, here's several helpful packages to load in

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the "../input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
# Any results you write to the current directory are saved as output.
"""

import tensorflow as tf
from keras.preprocessing.sequence import pad_sequences
from keras.preprocessing.text import Tokenizer
from keras.models import Model
from keras.layers import Flatten, Dense, LSTM, Dropout, Embedding, Activation
from keras.layers import concatenate, BatchNormalization, Input
from keras.layers.merge import add
from keras.utils import to_categorical
from keras.applications.inception_v3 import InceptionV3, preprocess_input
from keras.utils import plot_model

import matplotlib.pyplot as plt
import cv2
import string
import time
print("Running.....")

token_path = '/kaggle/input/flickr8k/flickr_data/Flickr_Data/Flickr_TextData/Flickr8k.token.txt'
text = open(token_path, 'r', encoding = 'utf-8').read()
print(text[:500])
```

```
def load_description(text):
    mapping = dict()
    for line in text.split("\n"):
        token = line.split("\t")
        if len(line) < 2:
            continue
        img_id = token[0].split('.')[0]
        img_des = token[1]
        if img_id not in mapping:
            mapping[img_id] = list()
        mapping[img_id].append(img_des)
    return mapping

descriptions = load_description(text)
print("Number of items: " + str(len(descriptions)))

descriptions['1000268201_693b08cb0e']

def clean_description(desc):
    for key, des_list in desc.items():
        for i in range(len(des_list)):
            caption = des_list[i]
            caption = [ch for ch in caption if ch not in
string.punctuation]
            caption = ''.join(caption)
            caption = caption.split(' ')
            caption = [word.lower() for word in caption if len(word)>1 and
word.isalpha()]
            caption = ' '.join(caption)
            des_list[i] = caption

clean_description(descriptions)
descriptions['1000268201_693b08cb0e']

def to_vocab(desc):
    words = set()
    for key in desc.keys():
        for line in desc[key]:
            words.update(line.split())
    return words
vocab = to_vocab(descriptions)
len(vocab)

import glob
images = '/kaggle/input/flickr8k/flickr_data/Flickr_Data/Images/'
# Create a list of all image names in the directory
img = glob.glob(images + '*.jpg')
len(img)

train_path =
'/kaggle/input/flickr8k/flickr_data/Flickr_Data/Flickr_TextData/Flickr_8k.t
rainImages.txt'
train_images = open(train_path, 'r', encoding = 'utf-8').read().split("\n")
train_img = []

for im in img:
    if(im[len(images):] in train_images):
        train_img.append(im)

test_path =
```

```
'/kaggle/input/flickr8k/flickr_data/Flickr_Data/Flickr_TextData/Flickr_8k.t
estImages.txt'
test_images = open(test_path, 'r', encoding = 'utf-8').read().split("\n")
test_img = []

for im in img:
    if(im[len(images): ] in test_images):
        test_img.append(im)
len(test_img)

#load descriptions of train and test set separately
def load_clean_descriptions(des, dataset):
    dataset_des = dict()
    for key, des_list in des.items():
        if key+'.jpg' in dataset:
            if key not in dataset_des:
                dataset_des[key] = list()
            for line in des_list:
                desc = 'startseq ' + line + ' endseq'
                dataset_des[key].append(desc)
    return dataset_des

train_descriptions = load_clean_descriptions(descriptions, train_images)
print('Descriptions: train=%d' % len(train_descriptions))

train_descriptions['1000268201_693b08cb0e']

from keras.preprocessing.image import load_img, img_to_array
def preprocess_img(img_path):
    #inception v3 excepts img in 299*299
    img = load_img(img_path, target_size = (299, 299))
    x = img_to_array(img)
    # Add one more dimension
    x = np.expand_dims(x, axis = 0)
    x = preprocess_input(x)
    return x

base_model = InceptionV3(weights = 'imagenet')
base_model.summary()

model = Model(base_model.input, base_model.layers[-2].output)

#function to encode an image into a vector using inception v3
def encode(image):
    image = preprocess_img(image)
    vec = model.predict(image)
    vec = np.reshape(vec, (vec.shape[1]))
    return vec

#run the encode function on all train images
start = time.time()
encoding_train = {}
for img in train_img:
    encoding_train[img[len(images):]] = encode(img)
print("Time Taken is: " + str(time.time() - start))

#Encode all the test images
start = time.time()
encoding_test = {}
for img in test_img:
    encoding_test[img[len(images):]] = encode(img)
```

```
print("Time taken is: " + str(time.time() - start))

train_features = encoding_train
test_features = encoding_test
print("Train image encodings: " + str(len(train_features)))
print("Test image encodings: " + str(len(test_features)))

train_features['1000268201_693b08cb0e.jpg'].shape

#list of all training captions
all_train_captions = []
for key, val in train_descriptions.items():
    for caption in val:
        all_train_captions.append(caption)
len(all_train_captions)

#onsider only words which occur atleast 10 times
vocabulary = vocab
threshold = 10
word_counts = {}
for cap in all_train_captions:
    for word in cap.split(' '):
        word_counts[word] = word_counts.get(word, 0) + 1

vocab = [word for word in word_counts if word_counts[word] >= threshold]
print("Unique words: " + str(len(word_counts)))
print("our Vocabulary: " + str(len(vocab)))

#word mapping to integers
ixtoword = {}
wordtoix = {}

ix = 1
for word in vocab:
    wordtoix[word] = ix
    ixtoword[ix] = word
    ix += 1

vocab_size = len(ixtoword) + 1  #1 for appended zeros
vocab_size

#find the maximum length of a description in a dataset
max_length = max(len(des.split()) for des in all_train_captions)
max_length

#since there are almost 30000 descriptions to process we will use
datagenerator
X1, X2, y = list(), list(), list()
for key, des_list in train_descriptions.items():
    pic = train_features[key + '.jpg']
    for cap in des_list:
        seq = [wordtoix[word] for word in cap.split(' ') if word in
wordtoix]
        for i in range(1, len(seq)):
            in_seq, out_seq = seq[:i], seq[i]
            in_seq = pad_sequences([in_seq], maxlen = max_length)[0]
            out_seq = to_categorical([out_seq], num_classes =
vocab_size)[0]
            #store
            X1.append(pic)
            X2.append(in_seq)
```

```
            y.append(out_seq)

X2 = np.array(X2)
X1 = np.array(X1)
y = np.array(y)
print(X1.shape)

#load glove vectors for embedding layer
embeddings_index = {}
glove = open('/kaggle/input/glove-global-vectors-for-word-
representation/glove.6B.200d.txt', 'r', encoding = 'utf-8').read()
for line in glove.split("\n"):
    values = line.split(" ")
    word = values[0]
    indices = np.asarray(values[1: ], dtype = 'float32')
    embeddings_index[word] = indices
print('Total word vectors: ' + str(len(embeddings_index)))

emb_dim = 200
emb_matrix = np.zeros((vocab_size, emb_dim))
for word, i in wordtoix.items():
    emb_vec = embeddings_index.get(word)
    if emb_vec is not None:
        emb_matrix[i] = emb_vec
emb_matrix.shape

# define the model
ip1 = Input(shape = (2048, ))
fe1 = Dropout(0.2)(ip1)
fe2 = Dense(256, activation = 'relu')(fe1)
ip2 = Input(shape = (max_length, ))
se1 = Embedding(vocab_size, emb_dim, mask_zero = True)(ip2)
se2 = Dropout(0.2)(se1)
se3 = LSTM(256)(se2)
decoder1 = add([fe2, se3])
decoder2 = Dense(256, activation = 'relu')(decoder1)
outputs = Dense(vocab_size, activation = 'softmax')(decoder2)
model = Model(inputs = [ip1, ip2], outputs = outputs)
model.summary()

model.layers[2]

model.layers[2].set_weights([emb_matrix])
model.layers[2].trainable = False
model.compile(loss = 'categorical_crossentropy', optimizer = 'adam')
plot_model(model, to_file = 'model.png', show_shapes = True,
show_layer_names = True)

for i in range(30):
    model.fit([X1, X2], y, epochs = 1, batch_size = 256)
    if(i%2 == 0):
        model.save_weights("image-caption-weights" + str(i) + ".h5")

def greedy_search(pic):
    start = 'startseq'
    for i in range(max_length):
        seq = [wordtoix[word] for word in start.split() if word in
wordtoix]
        seq = pad_sequences([seq], maxlen = max_length)
        yhat = model.predict([pic, seq])
        yhat = np.argmax(yhat)
```

```
        word = ixtoword[yhat]
        start += ' ' + word
        if word == 'endseq':
            break
    final = start.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final

pic = list(encoding_test.keys())[250]
img = encoding_test[pic].reshape(1, 2048)
x = plt.imread(images + pic)
plt.imshow(x)
plt.show()
print(greedy_search(img))

pic = list(encoding_test.keys())[570]
img = encoding_test[pic].reshape(1, 2048)
x = plt.imread(images + pic)
plt.imshow(x)
plt.show()
print(greedy_search(img))

model.save("my_model.h5")

#train it for some more time
model.fit([X1, X2], y, epochs = 1, batch_size = 64)
model.save("my_model_"+str(i)+".h5")

pic = list(encoding_test.keys())[888]
img = encoding_test[pic].reshape(1, 2048)
x = plt.imread(images + pic)
plt.imshow(x)
plt.show()
print(greedy_search(img))

model.save("my-cap.h
```

2. Sentiment Analysis Using Imdb Dataset

Create a Google Collaboratory file and perform the following code:

```
# -*- coding: utf-8 -*-
"""Untitled0.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1WPIV-
sePAA1kOth4s0D4toQTd597Sqtj
"""

from google.colab import drive
drive.mount('/content/drive')

import numpy as np
import tensorflow as tf
from tensorflow.keras import layers, models, preprocessing
```

```
from tensorflow.keras.datasets import imdb

# Load the IMDB dataset
(train_data, train_labels), (test_data, test_labels) =
imdb.load_data(num_words=10000)

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

x_train = vectorize_sequences(train_data)
x_test = vectorize_sequences(test_data)

y_train = np.asarray(train_labels).astype("float32")
y_test = np.asarray(test_labels).astype("float32")

from tensorflow.keras import layers, models, regularizers, optimizers

model = models.Sequential()
model.add(layers.Dense(128, activation="relu",
kernel_regularizer=regularizers.l2(0.001), input_shape=(10000,)))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(64, activation="relu",
kernel_regularizer=regularizers.l2(0.001)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(64, activation="relu",
kernel_regularizer=regularizers.l2(0.001)))
model.add(layers.Dropout(0.3))
model.add(layers.Dense(32, activation="relu",
kernel_regularizer=regularizers.l2(0.001)))
model.add(layers.Dropout(0.2))
model.add(layers.Dense(1, activation="sigmoid"))

# Set a custom learning rate
learning_rate = 0.0005
optimizer = optimizers.RMSprop(lr=learning_rate)

model.compile(optimizer=optimizer,
              loss="binary_crossentropy",
              metrics=["accuracy"])

from tensorflow.keras.callbacks import ModelCheckpoint
checkpoint_filepath = "/content/drive/MyDrive/sentiment_analysis_model.h5"
checkpoint_callback = ModelCheckpoint(
    filepath=checkpoint_filepath,
    monitor="val_loss",
    save_best_only=True,
    mode="min",
    verbose=1
)

model.fit(x_train, y_train, epochs=10, batch_size=512,
validation_data=(x_test, y_test), callbacks=[checkpoint_callback])

from tensorflow.keras.models import load_model
from tensorflow.keras.datasets import imdb

model = load_model("/content/drive/MyDrive/sentiment_analysis_model.h5")
word_index = imdb.get_word_index()
```

Developing Flask Application

Create/Add following files in Cloud-Project Directory :

static/uploads

templates/index.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>AI Demo</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f7f7f7;
      margin: 0;
      padding: 0;
    }
    .container {
      max-width: 800px;
      margin: 0 auto;
      padding: 2rem;
      background-color: #ffffff;
      box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    }
    h1 {
      text-align: center;
      color: #333;
      margin-bottom: 2rem;
    }
    h3 {
      margin-top: 1.5rem;
      color: #333;
    }
    .btn {
      border: 2px solid #333;
      color: #333;
      background-color: white;
      padding: 8px 20px;
      border-radius: 8px;
      font-size: 20px;
      font-weight: bold;
      cursor: pointer;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>AI Demo</h1>
    <form action="/" method="post">
      <label for="model">Choose a model:</label>
      <select name="model" id="model">
        <option value="image_captioning">Image Captioning</option>
        <option value="sentiment_analysis">Sentiment Analysis</option>
      </select>
      <input type="submit" value="Select" class="btn">
```

```
    </form>
    {% if selected_model == "image_captioning" %}
    <h3>Image Captioning</h3>
    <form action="/upload" method="post" enctype="multipart/form-data">
      <label for="file">Choose an image:</label>
      <input type="file" name="file" id="file" required>
      <input type="submit" value="Submit" class="btn">
    </form>
    {% elif selected_model == "sentiment_analysis" %}
    <h3>Sentiment Analysis</h3>
    <form action="/sentiment_analysis" method="post">
      <label for="text">Enter text:</label>
      <textarea name="text" id="text" cols="40" rows="5"
required></textarea>
      <input type="submit" value="Analyze Sentiment" class="btn">
    </form>
    {% endif %}
  </div>
</body>
</html>
```

templates/results.html

```
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <title>AI Demo Results</title>
  <style>
    body {
      font-family: Arial, sans-serif;
      background-color: #f7f7f7;
      margin: 0;
      padding: 0;
    }
    .container {
      max-width: 800px;
      margin: 0 auto;
      padding: 2rem;
      background-color: #ffffff;
      box-shadow: 0 2px 5px rgba(0, 0, 0, 0.1);
    }
    h1 {
      text-align: center;
      color: #333;
      margin-bottom: 2rem;
    }
    h3 {
      margin-top: 1.5rem;
      color: #333;
    }
    .result {
      font-size: 18px;
      font-weight: bold;
      color: #333;
      margin-bottom: 1rem;
    }
    .btn {
      border: 2px solid #333;
      color: #333;
```

```
      background-color: white;
      padding: 8px 20px;
      border-radius: 8px;
      font-size: 20px;
      font-weight: bold;
      cursor: pointer;
      display: block;
      margin: 0 auto;
      margin-top: 2rem;
    }
    .image-container {
      display: flex;
      justify-content: center;
      align-items: center;
      margin-bottom: 1rem;
    }
  </style>
</head>
<body>
  <div class="container">
    <h1>AI Demo Results</h1>
    {% if selected_model == "image_captioning" %}
    <h3>Image Captioning</h3>
    <div class="image-container">
      <img src="{{ url_for('display_image', filename=filename) }}"
alt="Uploaded image" style="width:100%; max-width: 600px;">
    </div>
    <p class="result"><strong>Greedy Search:</strong> {{ caption_greedy
}}</p>
    <p class="result"><strong>Beam Search (k=3):</strong> {{ caption_beam_3
}}</p>
    <p class="result"><strong>Beam Search (k=5):</strong> {{ caption_beam_5
}}</p>
    {% elif selected_model == "sentiment_analysis" %}
    <h3>Sentiment Analysis</h3>
    <p class="result">Sentiment: {{ sentiment }}</p>
    {% endif %}
    <a href="{{ url_for('index') }}"><button class="btn">Back</button></a>
  </div>
</body>
</html>
```

caption_generator.py

```
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.sequence import pad_sequences
from tensorflow.keras.models import Model
from tensorflow.keras.applications.inception_v3 import InceptionV3,
preprocess_input
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from pickle import load

model = load_model('new-model-1.h5')
pickle_in = open("wordtoix.pkl", "rb")
wordtoix = load(pickle_in)
pickle_in = open("ixtoword.pkl", "rb")
ixtoword = load(pickle_in)
max_length = 74
```

```
# The rest of the functions and code from the Tkinter version (excluding
the GUI parts)

def generate_caption(file_path):
    enc = encode(file_path)
    image = enc.reshape(1, 2048)
    caption_greedy = greedy_search(image)
    caption_beam_3 = beam_search(image)
    caption_beam_5 = beam_search(image, 5)
    return caption_greedy, caption_beam_3, caption_beam_5


def encode(image_path):
    image = load_img(image_path, target_size=(299, 299))
    image = img_to_array(image)
    image = np.expand_dims(image, axis=0)
    image = preprocess_input(image)

    inception_v3 = InceptionV3(weights='imagenet')
    inception_v3 = Model(inception_v3.input, inception_v3.layers[-
2].output)

    features = inception_v3.predict(image)
    return features


def greedy_search(photo):
    in_text = 'startseq'
    for i in range(max_length):
        sequence = [wordtoix[w] for w in in_text.split() if w in wordtoix]
        sequence = pad_sequences([sequence], maxlen=max_length)
        yhat = model.predict([photo, sequence], verbose=0)
        yhat = np.argmax(yhat)
        word = ixtoword[yhat]
        in_text += ' ' + word
        if word == 'endseq':
            break
    final = in_text.split()
    final = final[1:-1]
    final = ' '.join(final)
    return final


def beam_search(image, beam_index=3):
    start = [wordtoix["startseq"]]
    start_word = [[start, 0.0]]
    while len(start_word[0][0]) < max_length:
        temp = []
        for s in start_word:
            par_caps = pad_sequences([s[0]], maxlen=max_length,
padding='post')
            preds = model.predict([image, par_caps], verbose=0)
            word_preds = np.argsort(preds[0])[-beam_index:]
            for w in word_preds:
                next_cap, prob = s[0][:], s[1]
                next_cap.append(w)
                prob += preds[0][w]
                temp.append([next_cap, prob])
        start_word = temp
        start_word = sorted(start_word, reverse=False, key=lambda l: l[1])
```

```
        start_word = start_word[-beam_index:]

    start_word = start_word[-1][0]
    intermediate_caption = [ixtoword[i] for i in start_word]
    final_caption = []
    for i in intermediate_caption:
        if i != 'endseq':
            final_caption.append(i)
        else:
            break
    final_caption = ' '.join(final_caption[1:])
    return final_caption
```

app.py

```
from flask import Flask, render_template, request, redirect, url_for
from werkzeug.utils import secure_filename
from tensorflow.keras.models import load_model
from tensorflow.keras.datasets import imdb
import numpy as np
import os

from caption_generator import encode, greedy_search, beam_search, model,
wordtoix, ixtoword, max_length

app = Flask(__name__)
app.config['UPLOAD_FOLDER'] = 'static/uploads/'

# Image Captioning Functions
def process_image(file_path):
    enc = encode(file_path)
    image = enc.reshape(1, 2048)
    caption_greedy = greedy_search(image)
    caption_beam_3 = beam_search(image)
    caption_beam_5 = beam_search(image, 5)
    return caption_greedy, caption_beam_3, caption_beam_5

@app.route('/', methods=['GET', 'POST'])
def index():
    selected_model = None
    if request.method == 'POST':
        selected_model = request.form['model']
    return render_template('index.html', selected_model=selected_model)

@app.route('/upload', methods=['POST'])
def upload_image():
    if request.method == 'POST':
        file = request.files['file']
        filename = secure_filename(file.filename)
        file_path = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(file_path)
        caption_greedy, caption_beam_3, caption_beam_5 =
process_image(file_path)
        return render_template('results.html',
selected_model='image_captioning', filename=filename,
caption_greedy=caption_greedy, caption_beam_3=caption_beam_3,
caption_beam_5=caption_beam_5)

@app.route('/display/<filename>')
```

```
def display_image(filename):
    return redirect(url_for('static', filename='uploads/' + filename),
code=301)

# Sentiment Analysis Functions
model = load_model("sentiment_analysis_model.h5")
word_index = imdb.get_word_index()

def vectorize_sequences(sequences, dimension=10000):
    results = np.zeros((len(sequences), dimension))
    for i, sequence in enumerate(sequences):
        results[i, sequence] = 1.
    return results

def sentiment_analysis(text):
    text_vector = vectorize_sequences([[word_index[word] for word in
text.lower().split() if word in word_index]])
    prediction = model.predict(text_vector)
    if prediction[0][0] > 0.5:
        return "Positive"
    else:
        return "Negative"

@app.route("/sentiment_analysis", methods=["GET", "POST"])
def sentiment_analysis_route():
    if request.method == "POST":
        text = request.form["text"]
        sentiment = sentiment_analysis(text)
        return render_template("results.html",
selected_model='sentiment_analysis', sentiment=sentiment)
    return render_template("index.html")

if __name__ == '__main__':
    app.run(debug=True,host='0.0.0.0',port=8080)
```

requirements.txt

```
absl-py==1.4.0
astunparse==1.6.3
blinker==1.6.2
cachetools==5.3.0
certifi==2022.12.7
charset-normalizer==3.1.0
click==8.1.3
Flask==2.3.1
flatbuffers==23.3.3
gast==0.4.0
google-auth==2.17.3
google-auth-oauthlib==1.0.0
google-pasta==0.2.0
grpcio==1.54.0
h5py==3.8.0
idna==3.4
itsdangerous==2.1.2
Jinja2==3.1.2
keras==2.12.0
libclang==16.0.0
Markdown==3.4.3
MarkupSafe==2.1.2
numpy==1.23.5
```

```
oauthlib==3.2.2
opencv-python-headless==4.7.0.72
opt-einsum==3.3.0
packaging==23.1
pandas==2.0.1
Pillow==9.5.0
protobuf==4.22.3
pyasn1==0.5.0
pyasn1-modules==0.3.0
python-dateutil==2.8.2
pytz==2023.3
requests==2.28.2
requests-oauthlib==1.3.1
rsa==4.9
scipy==1.10.1
six==1.16.0
tensorboard==2.12.2
tensorboard-data-server==0.7.0
tensorboard-plugin-wit==1.8.1
tensorflow==2.12.0
tensorflow-estimator==2.12.0
termcolor==2.3.0
typing_extensions==4.5.0
tzdata==2023.3
urllib3==1.26.15
Werkzeug==2.3.0
wrapt==1.14.1
```

inception_v3_weights_tf_dim_ordering_tf_kernels.h5

ixtoword.pkl

wordtoix.pkl

new-model-1.h5

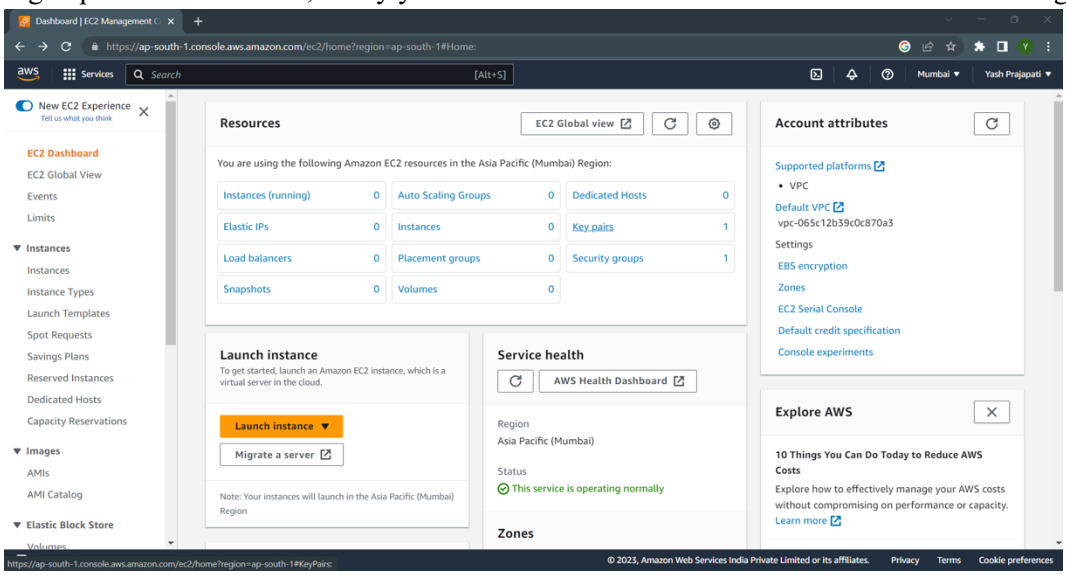
sentiment_analysis_model.h5

How to run it in localhost Environment:

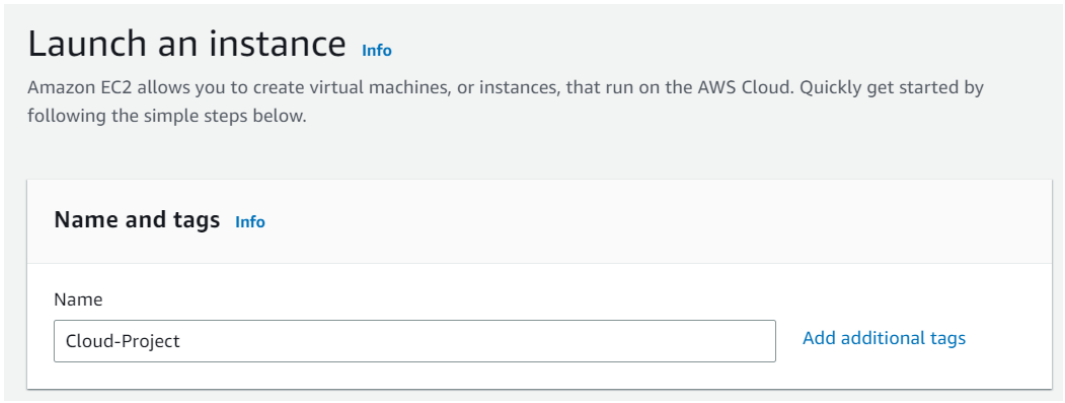
- *pip3 install virtualenv*
- *python3 -m venv venv*
- *source venv/bin/activate*
- *pip3 install -r requirements.txt*
- *python3 app.py*

AWS Connectivity

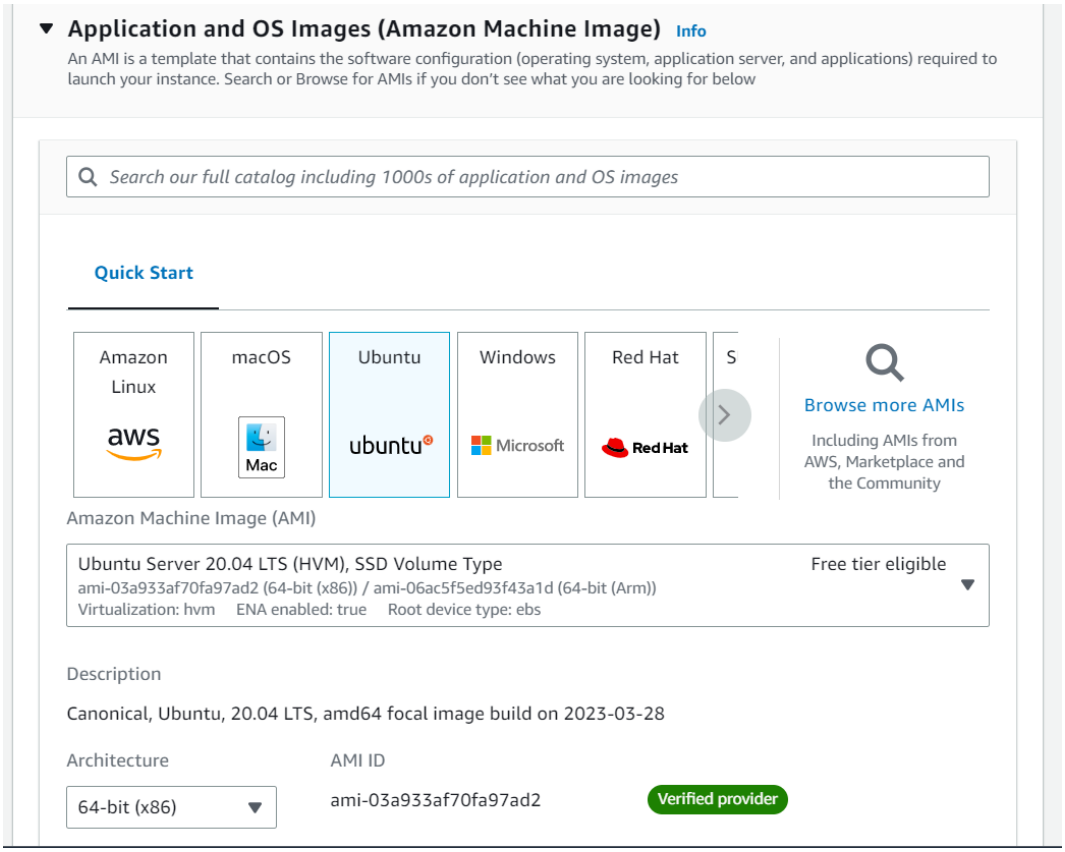
1. Sign up to AWS free Tier, verify your account and then select EC2 on Dashboard after login



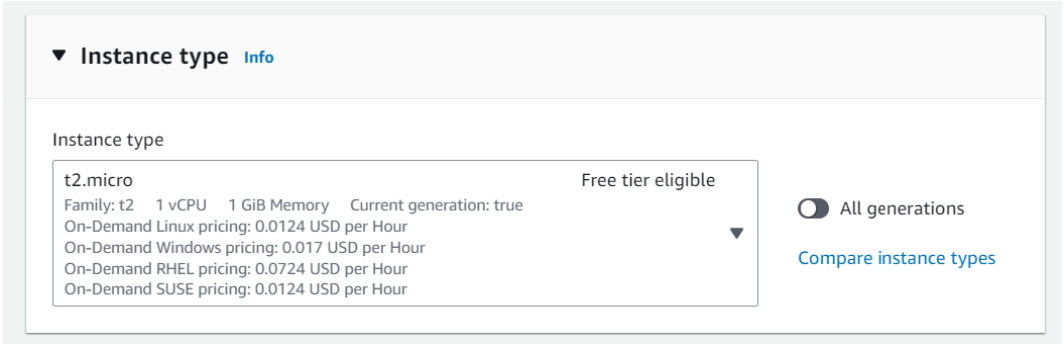
2. Select Launch Instance and Give the appropriate name to project



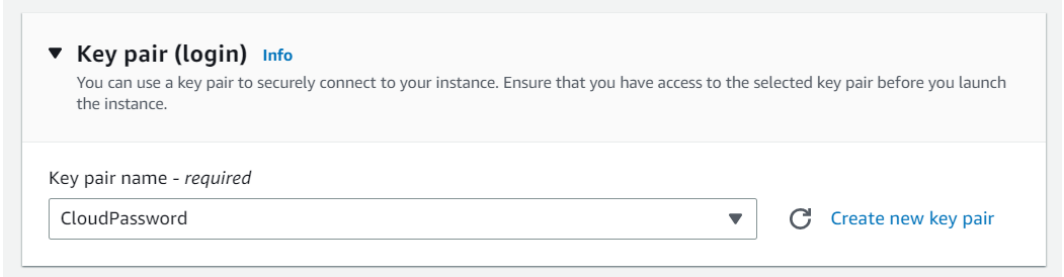
3. Choose Application and OS Images i.e., Ubuntu free tier 20.04 LTS

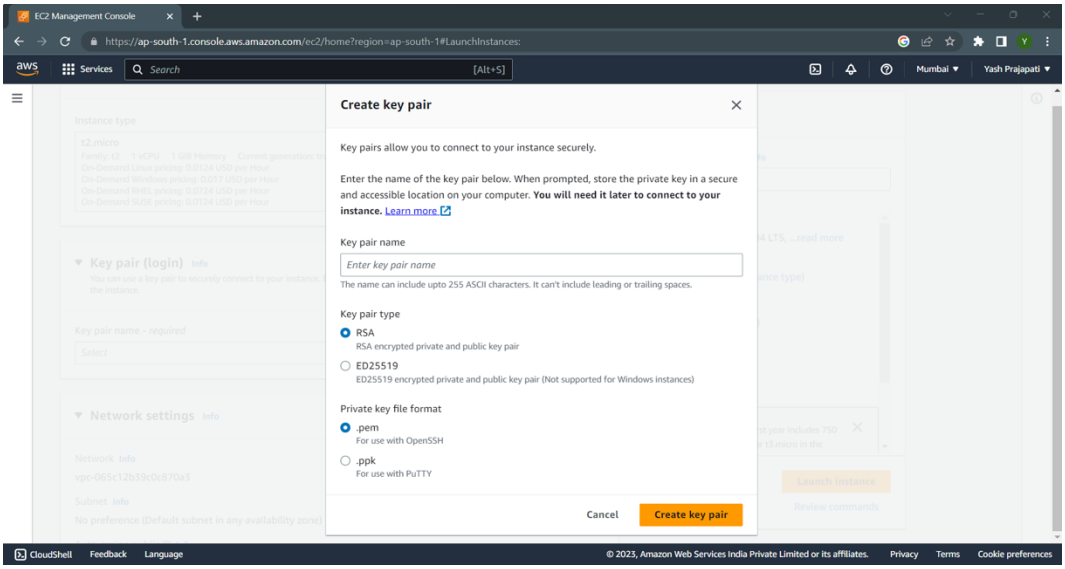


4. Choose Instance type : t2.micro

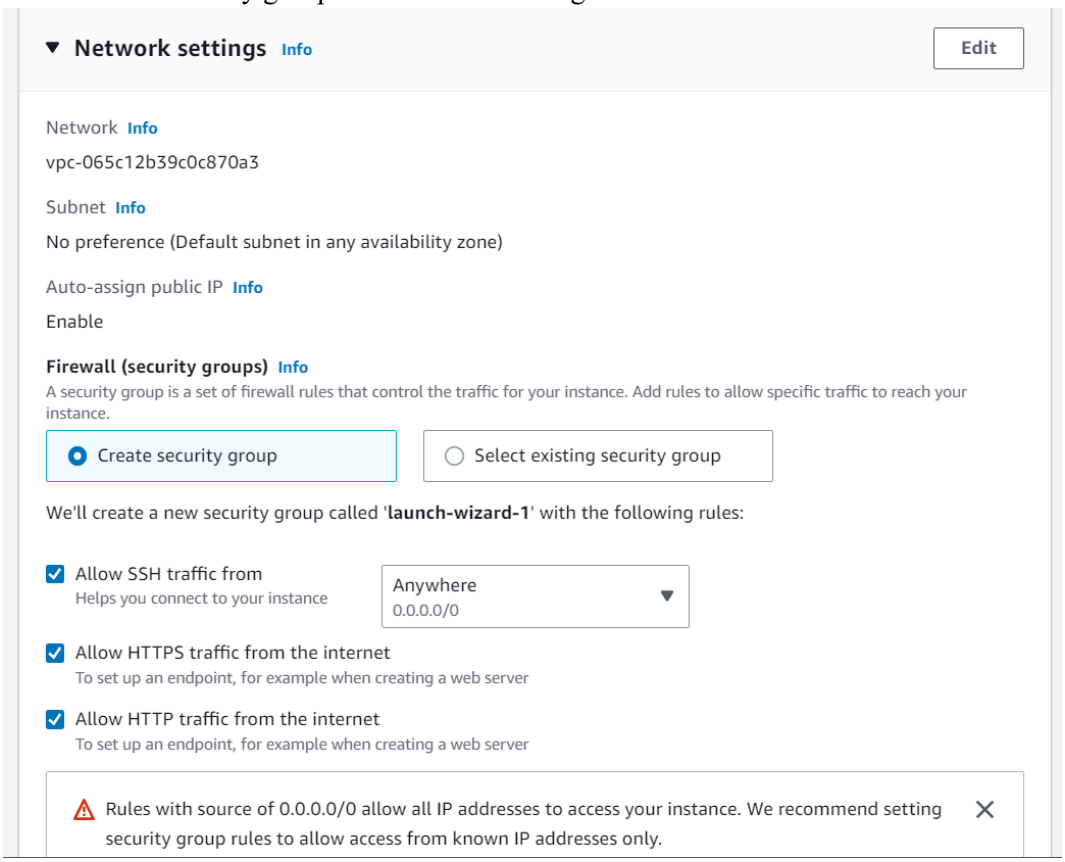


5. Create New Key-Pair and download the .pem file

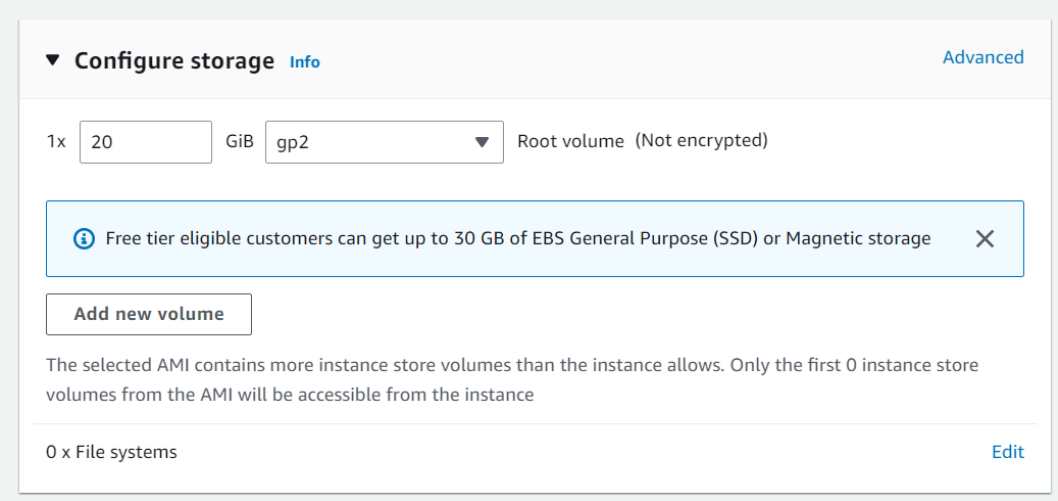




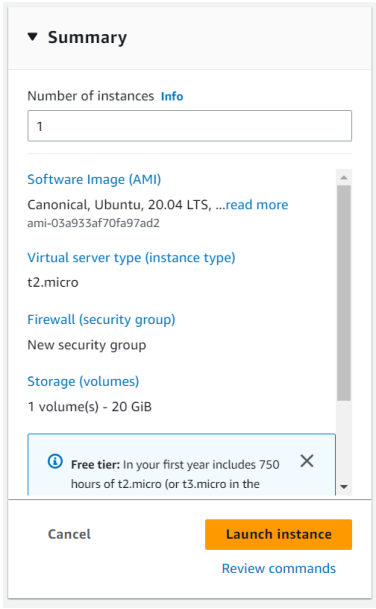
6. Create a new security group from network setting:

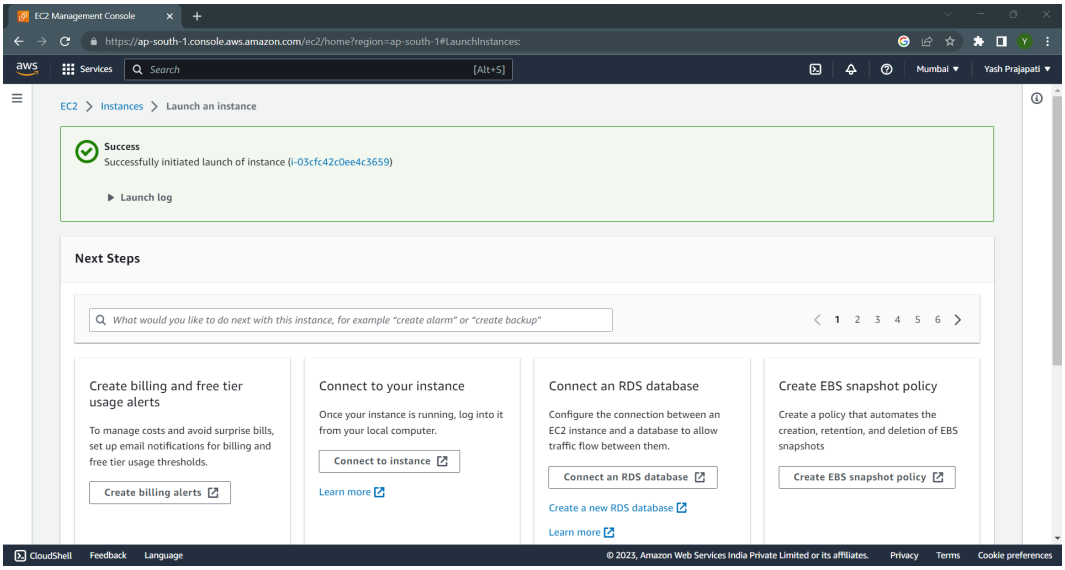


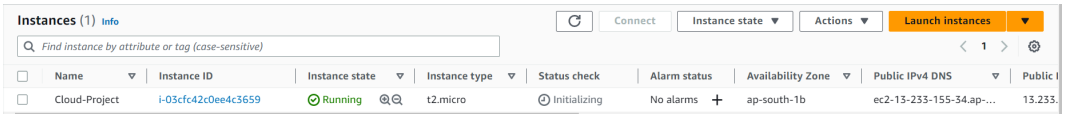
7. Configure Storage :

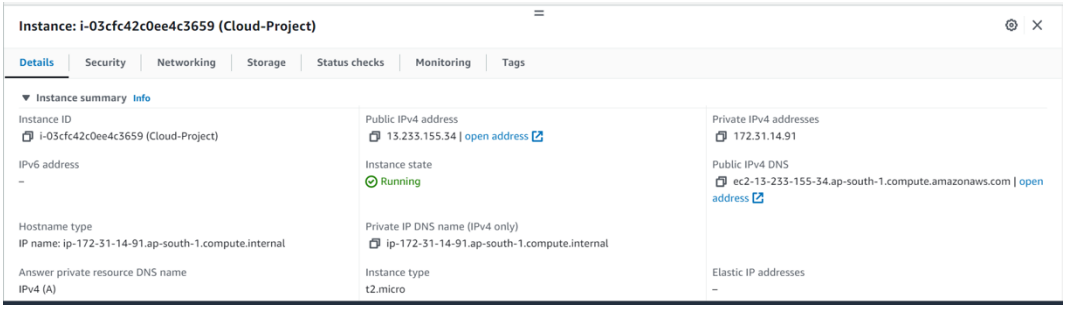


8. Click on Launch Instance:

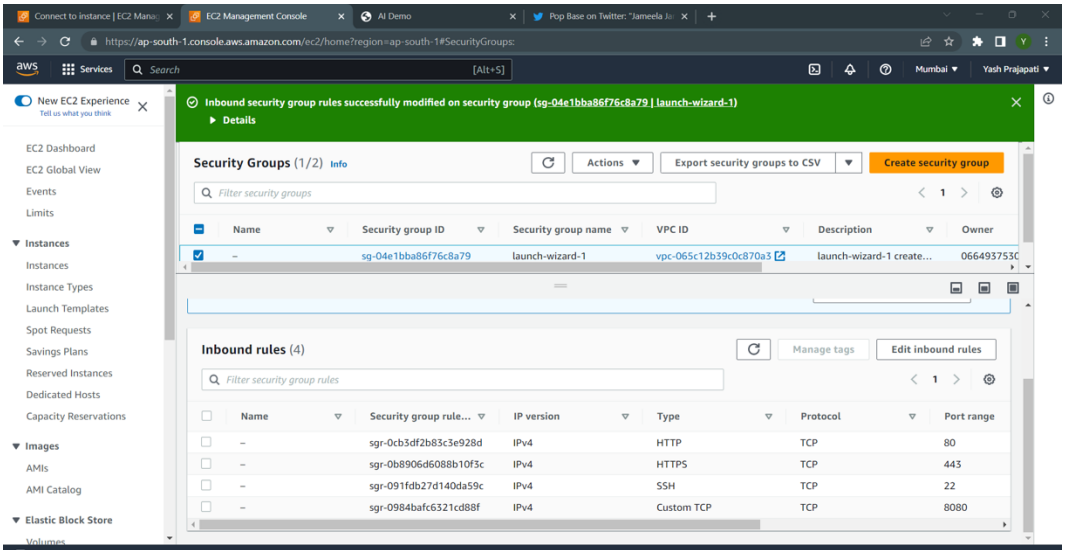


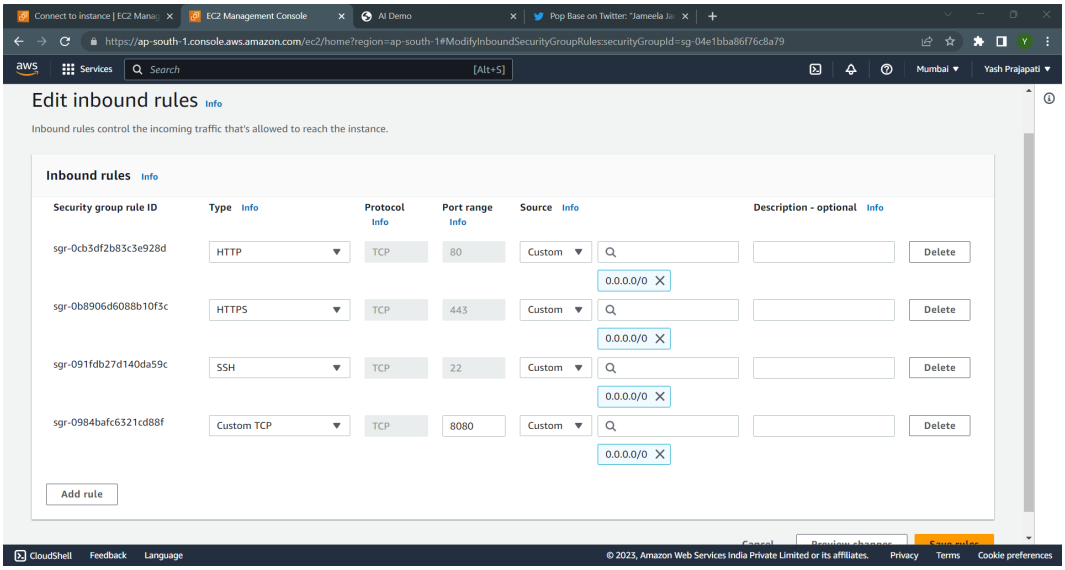






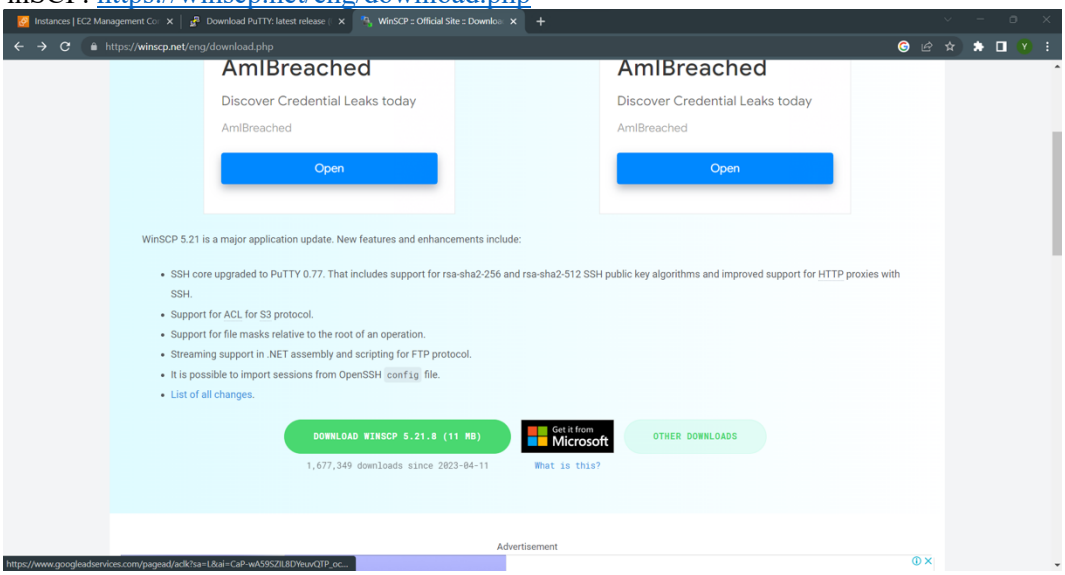
9. Edit Inbound Rules in Security Groups:



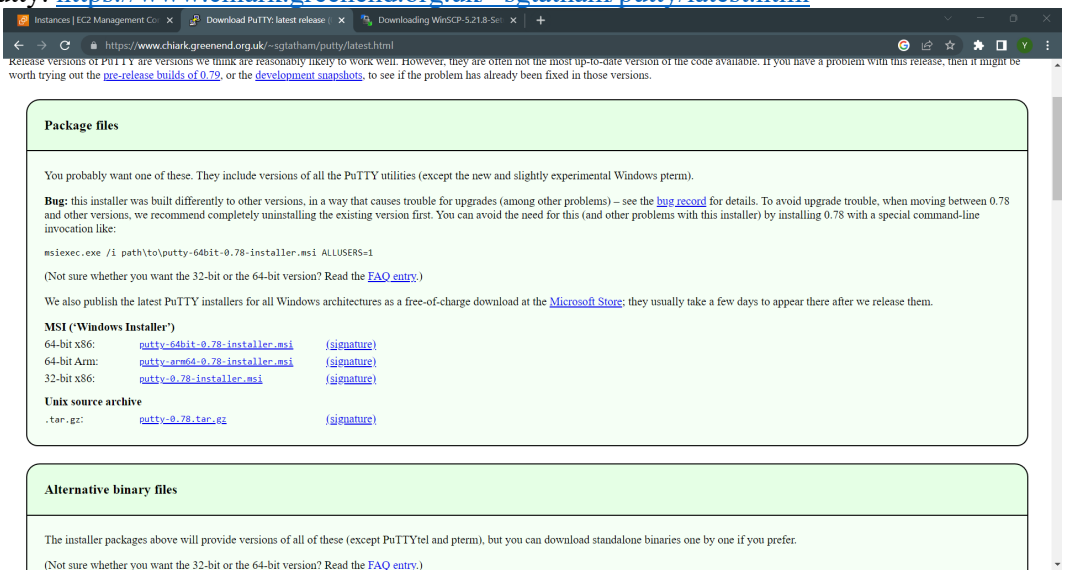


Putty and WinSCP

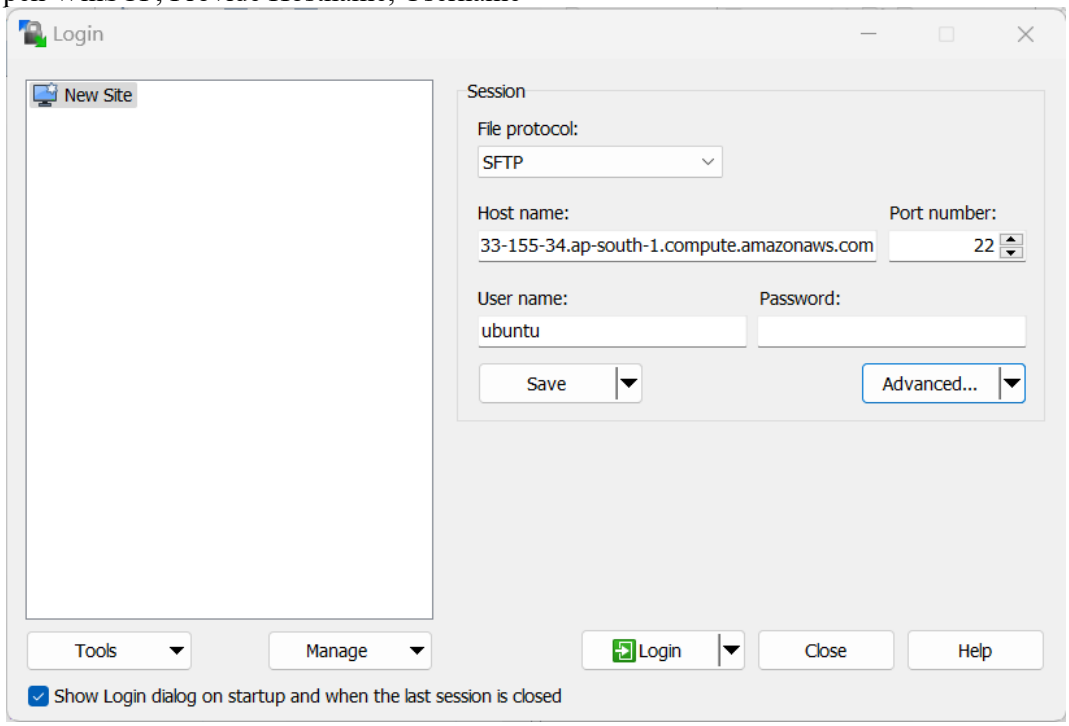
1. Download WinSCP and Putty from respective websites and install it on your PC
   WinSCP: https://winscp.net/eng/download.php



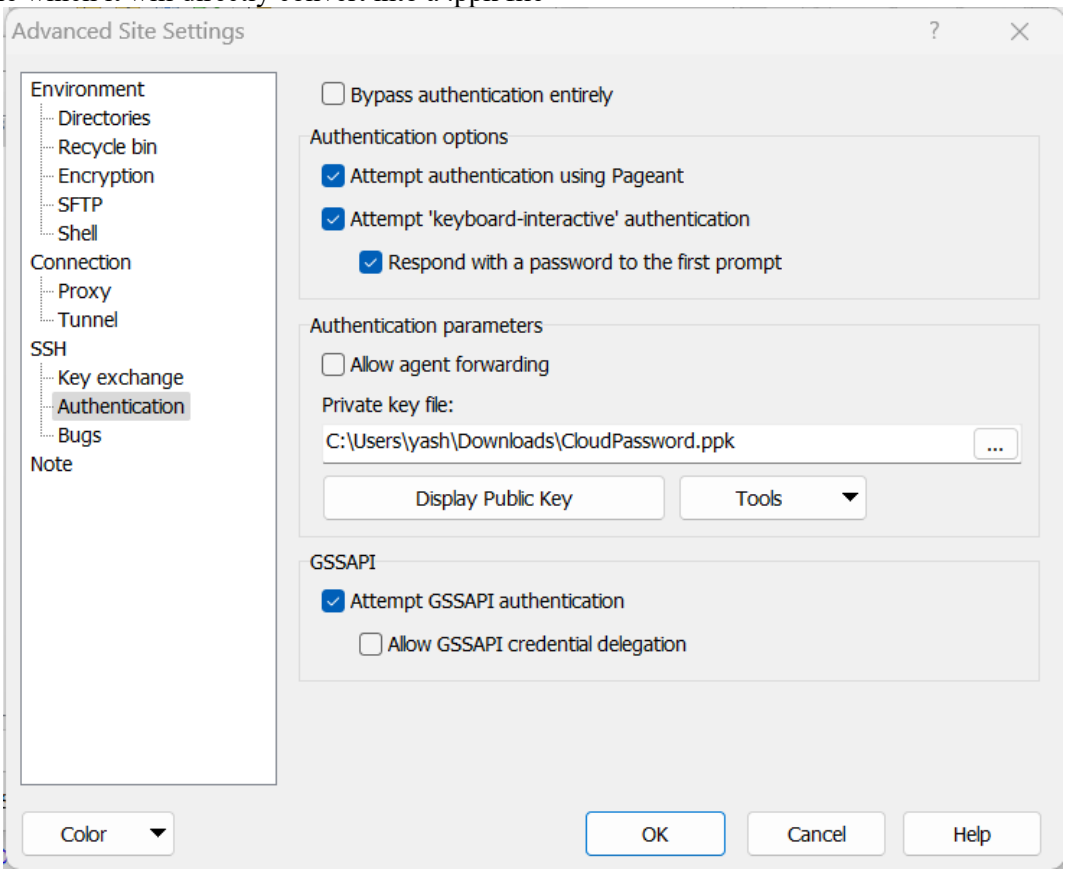
Putty: https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html



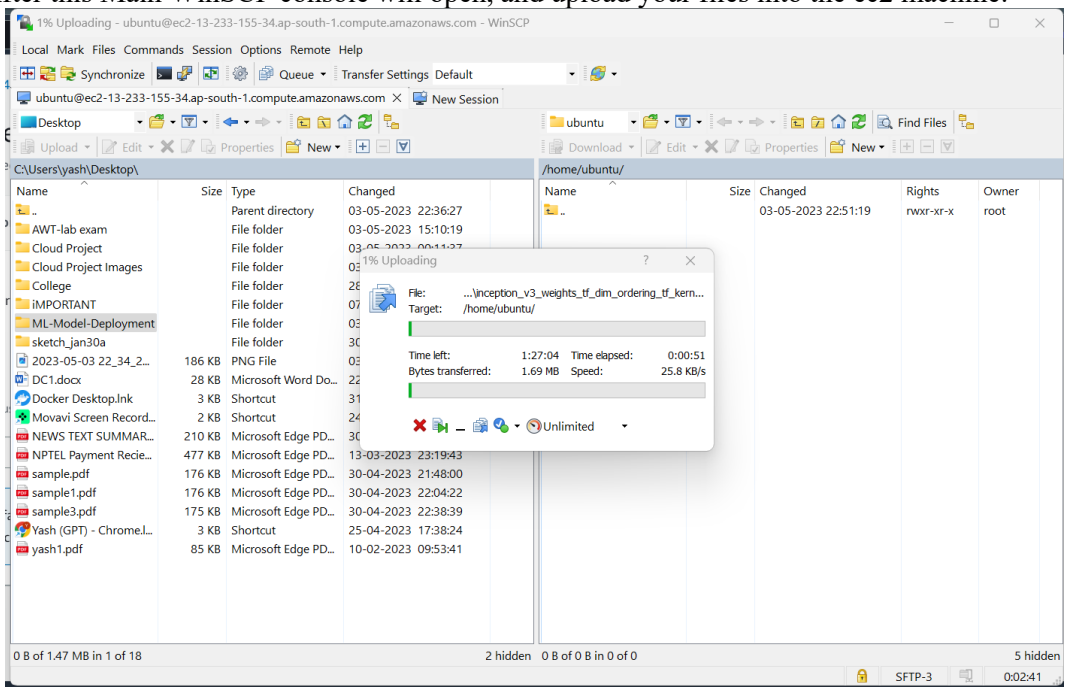
2. Open WinSCP, Provide Hostname, Username

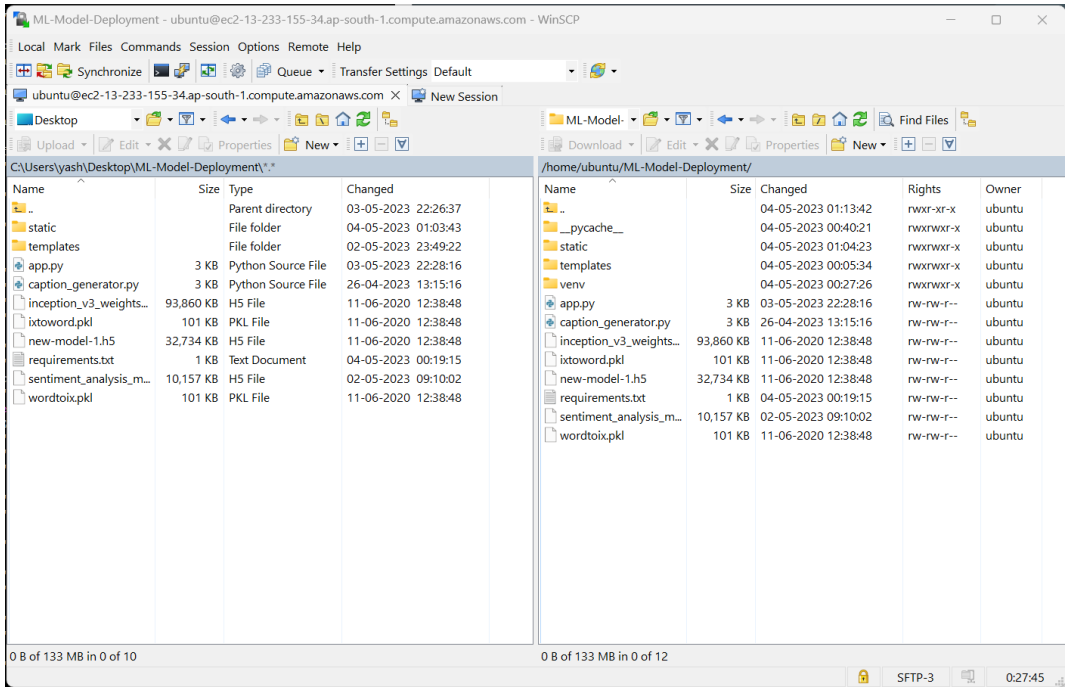


3. In the Password section, Click on Advanced settings, SSH, Authentication, and Provide .pem file which it will directly convert into a .ppk file



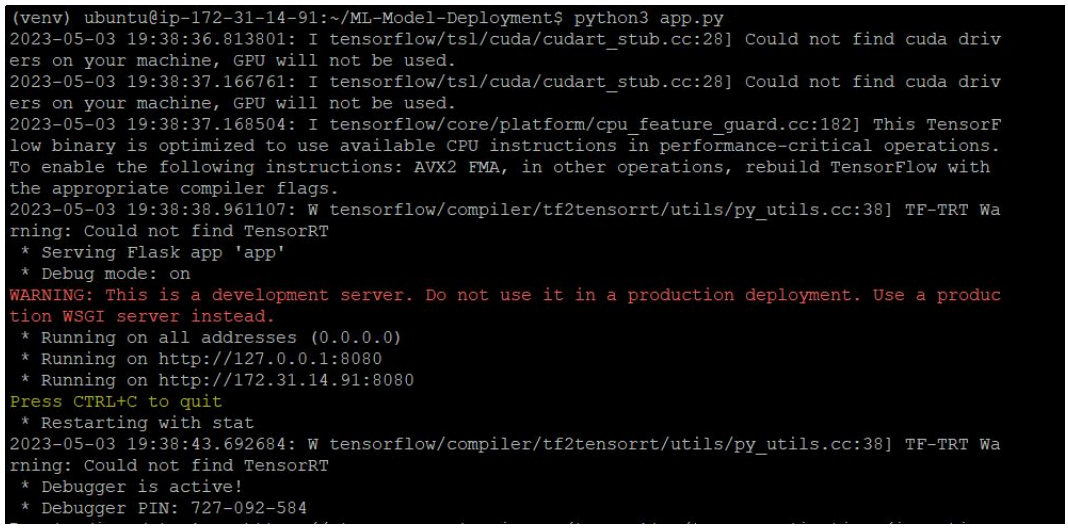
4. After this Main WinSCP console will open, and upload your files into the ec2 machine:





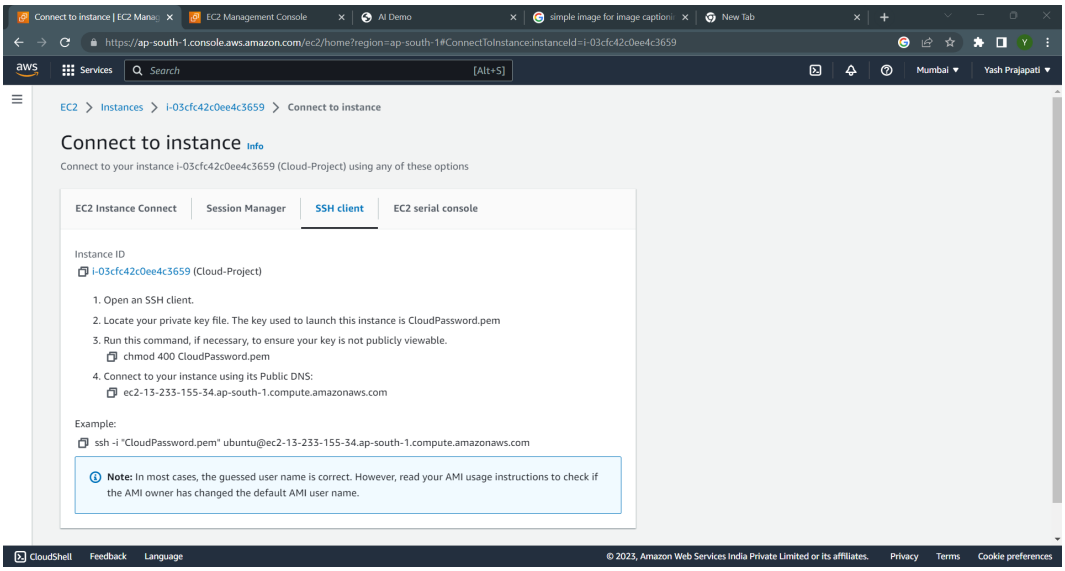
5. Open Session in Putty directly from WinSCP Menu bar, 2nd Row fifth icon and then Run Following Commands in it:
    - *python*
    - *sudo apt install python3*
    - *python3*
    - *sudo apt -get update && sudo apt -get install python3-pip*
    - *cd ML-Model-Deployment*
    - *sudo apt-update*
    - *sudo apt install python3.8-venv*
    - *python3 -m venv venv*
    - *source venv/bin/activate*
    - *pip3 install -r requirements.txt*
    - *sudo fallocate -l 2G /swapfile*
    - *sudo chmod 600 /swapfile*
    - *sudo mkswap /swapfile*
    - *sudo swapon /swapfile*
    - *pip3 install tensorflow*
    - *python3 app.py*

6. Your Server will be started Now

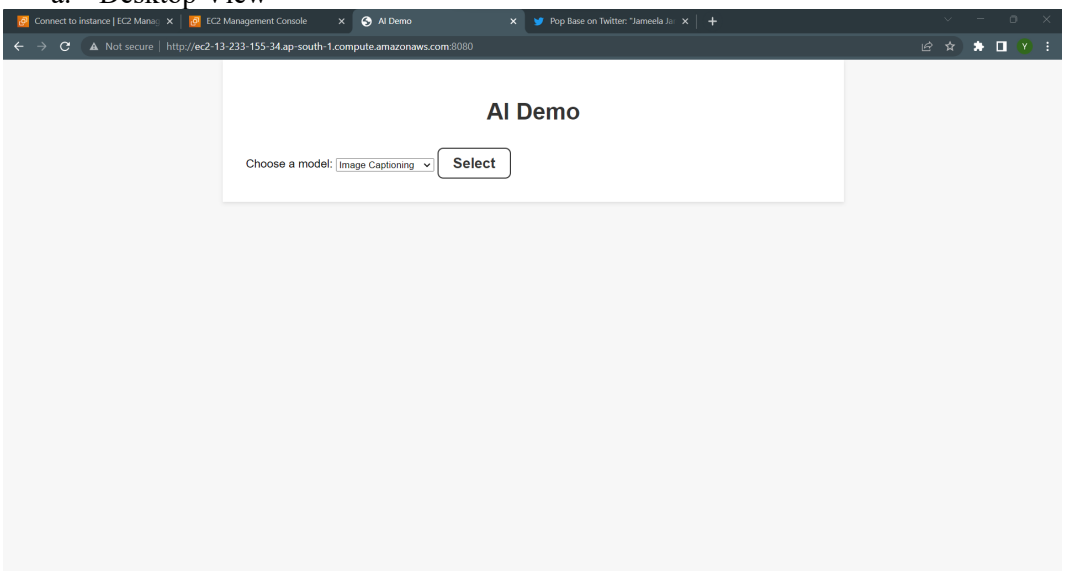


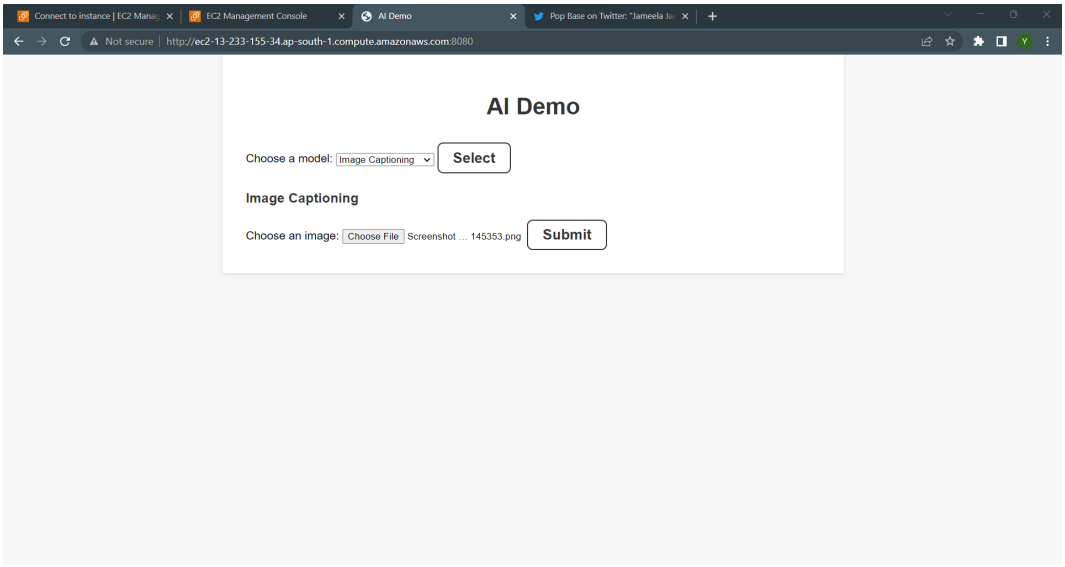
Testing on Different Devices

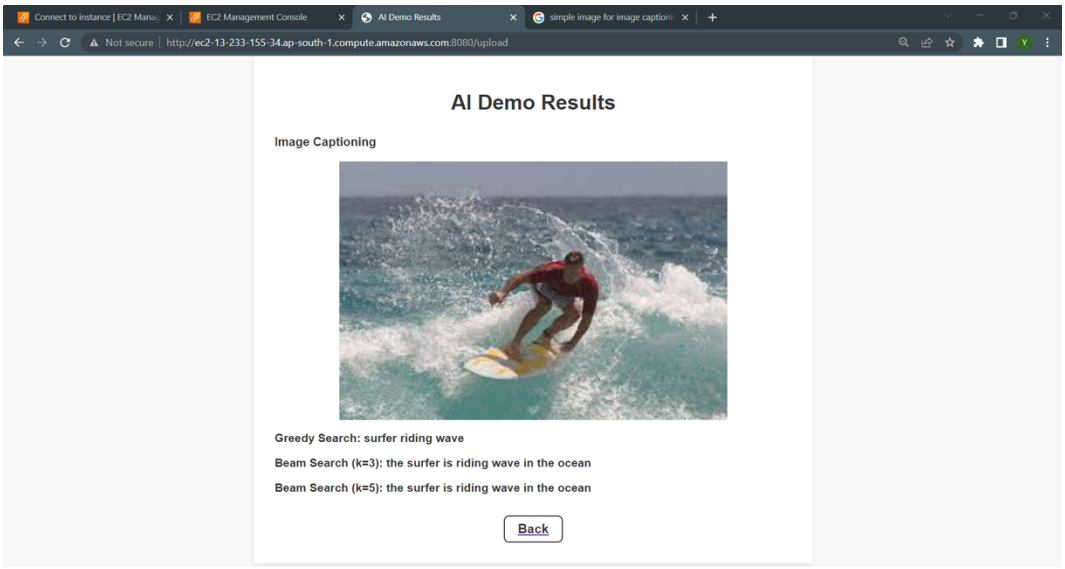
1. The server will be started on the following link : http://ec2-13-233-155-34.ap-south-1.compute.amazonaws.com:8080/

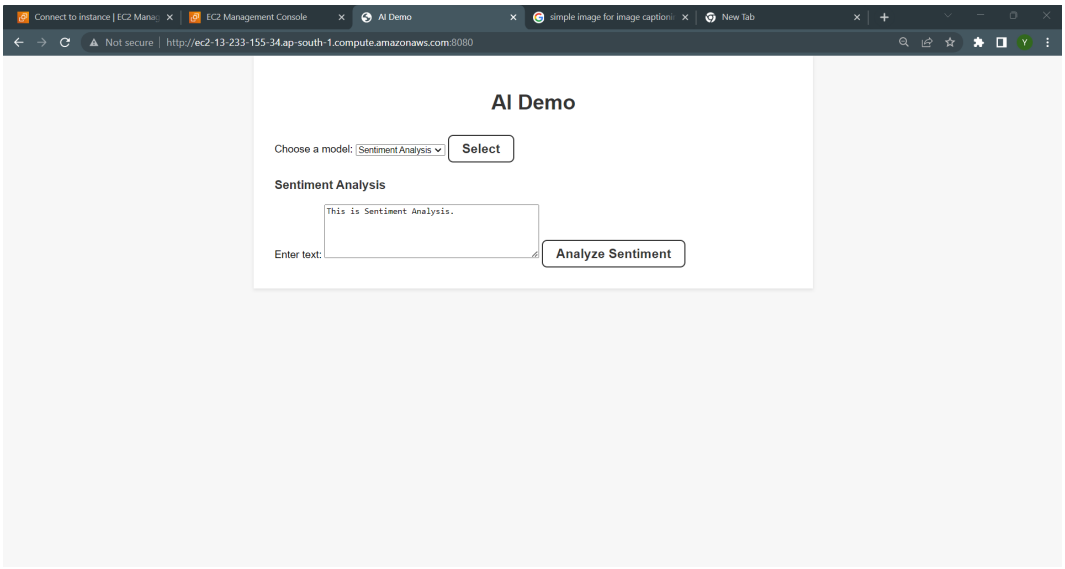


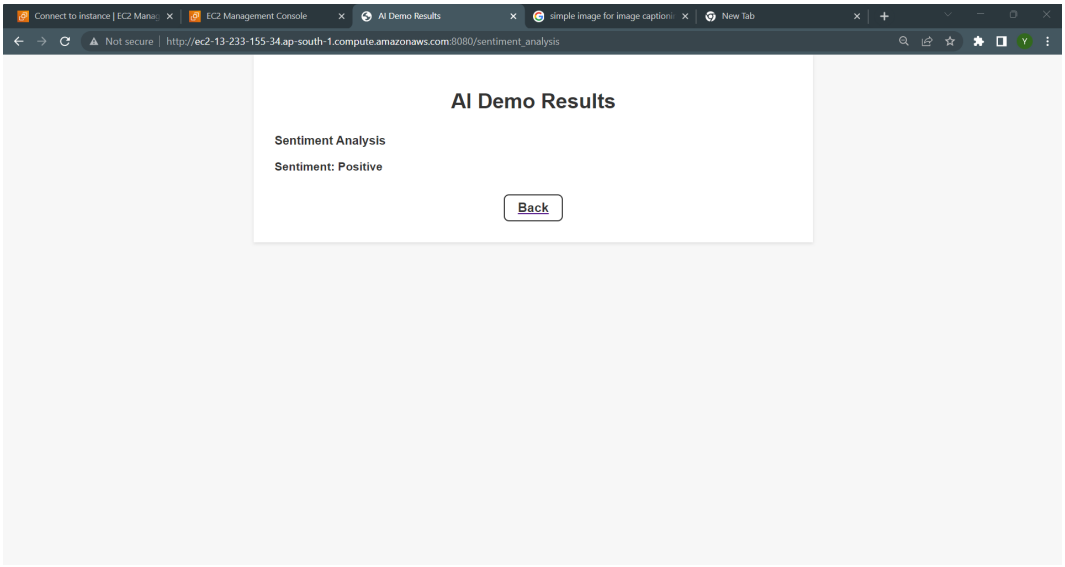
2. Open the APP on different devices and use it!
    a. Desktop View



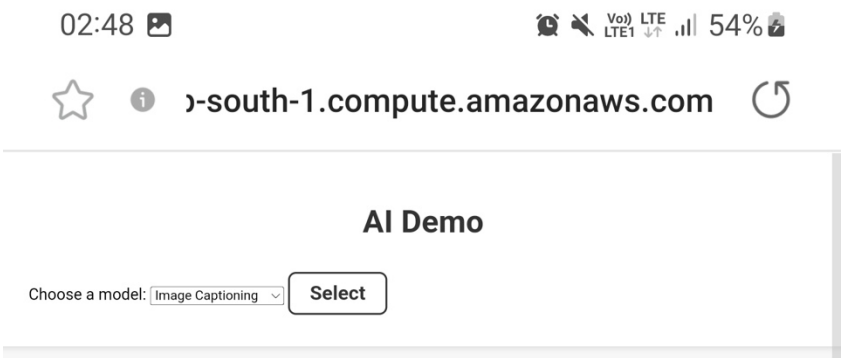

Connect to instance | EC2 Mana
EC2 Management Console
AI Demo
Pop Base on Twitter: "Jameela Ja
Not secure
http://ec2-13-233-155-34.ap-south-1.compute.amazonaws.com:8080
AI Demo
Choose a model: Image Captioning
Select
Image Captioning
Choose an image: Choose File Screenshot ... 145353.png
Submit

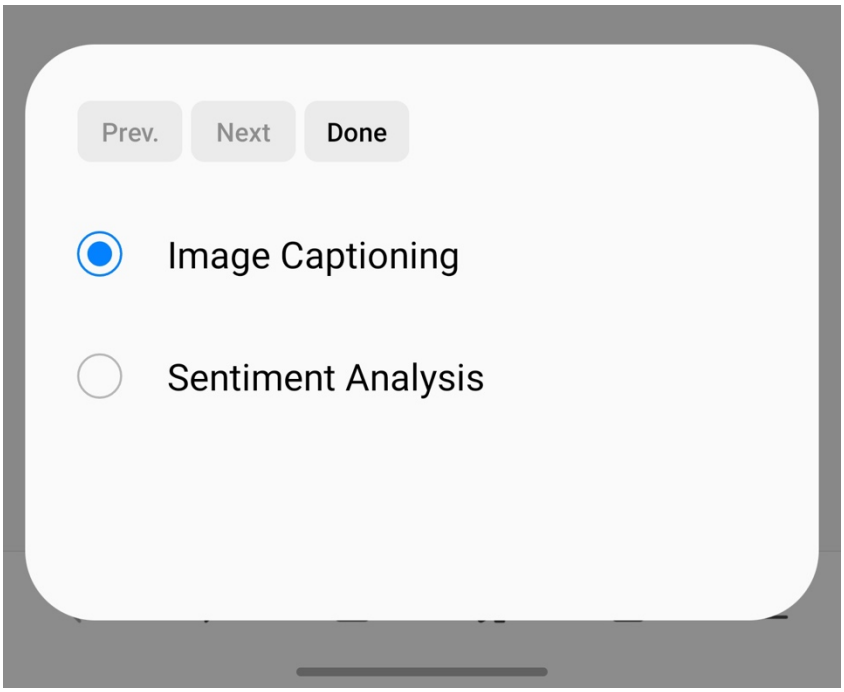

Connect to instance | EC2 Mana
EC2 Management Console
AI Demo Results
simple image for image captioni
Not secure
http://ec2-13-233-155-34.ap-south-1.compute.amazonaws.com:8080/upload
AI Demo Results
Image Captioning
Greedy Search: surfer riding wave
Beam Search (k=3): the surfer is riding wave in the ocean
Beam Search (k=5): the surfer is riding wave in the ocean
Back

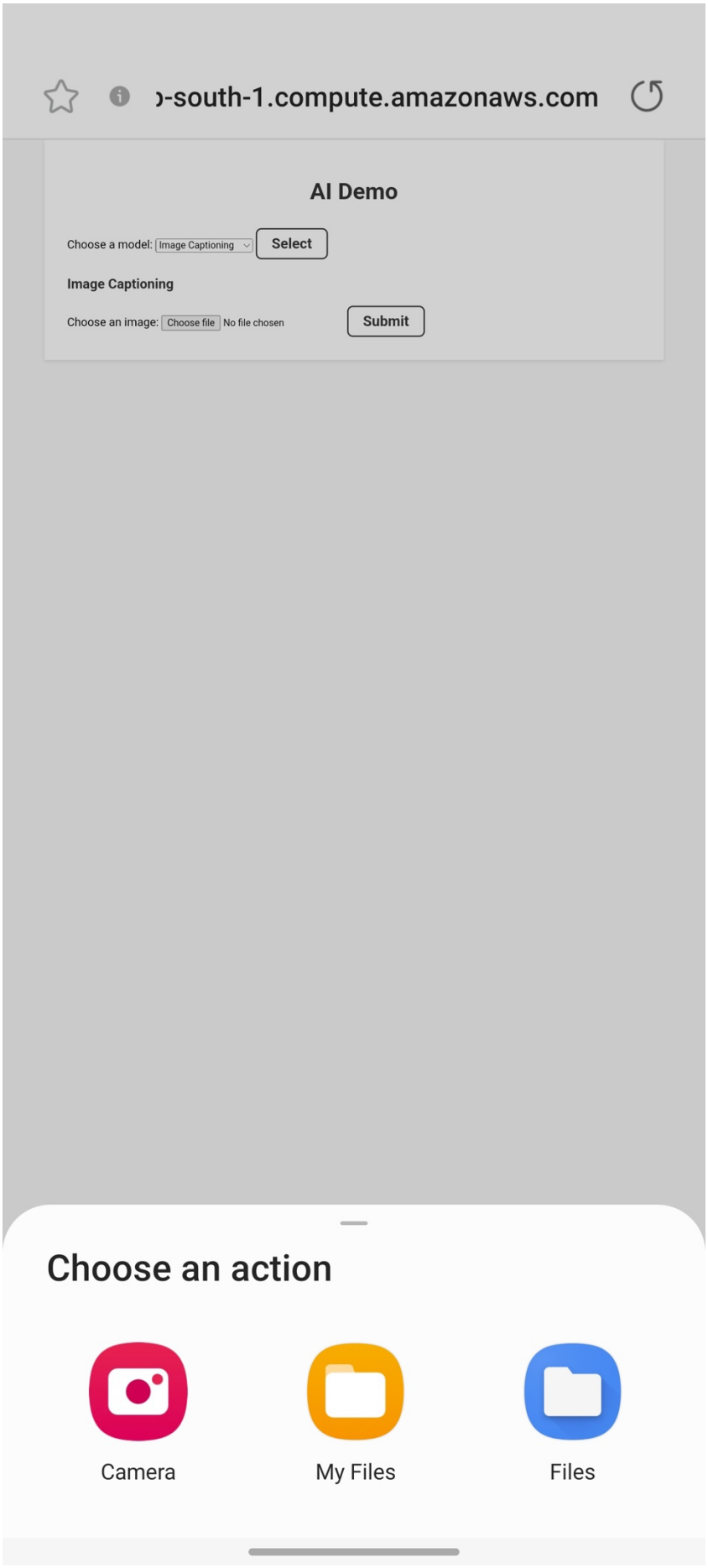

Connect to instance | EC2 Mana
EC2 Management Console
AI Demo
simple image for image captioni
New Tab
Not secure
http://ec2-13-233-155-34.ap-south-1.compute.amazonaws.com:8080
AI Demo
Choose a model: Sentiment Analysis
Select
Sentiment Analysis
Enter text: This is Sentiment Analysis.
Analyze Sentiment

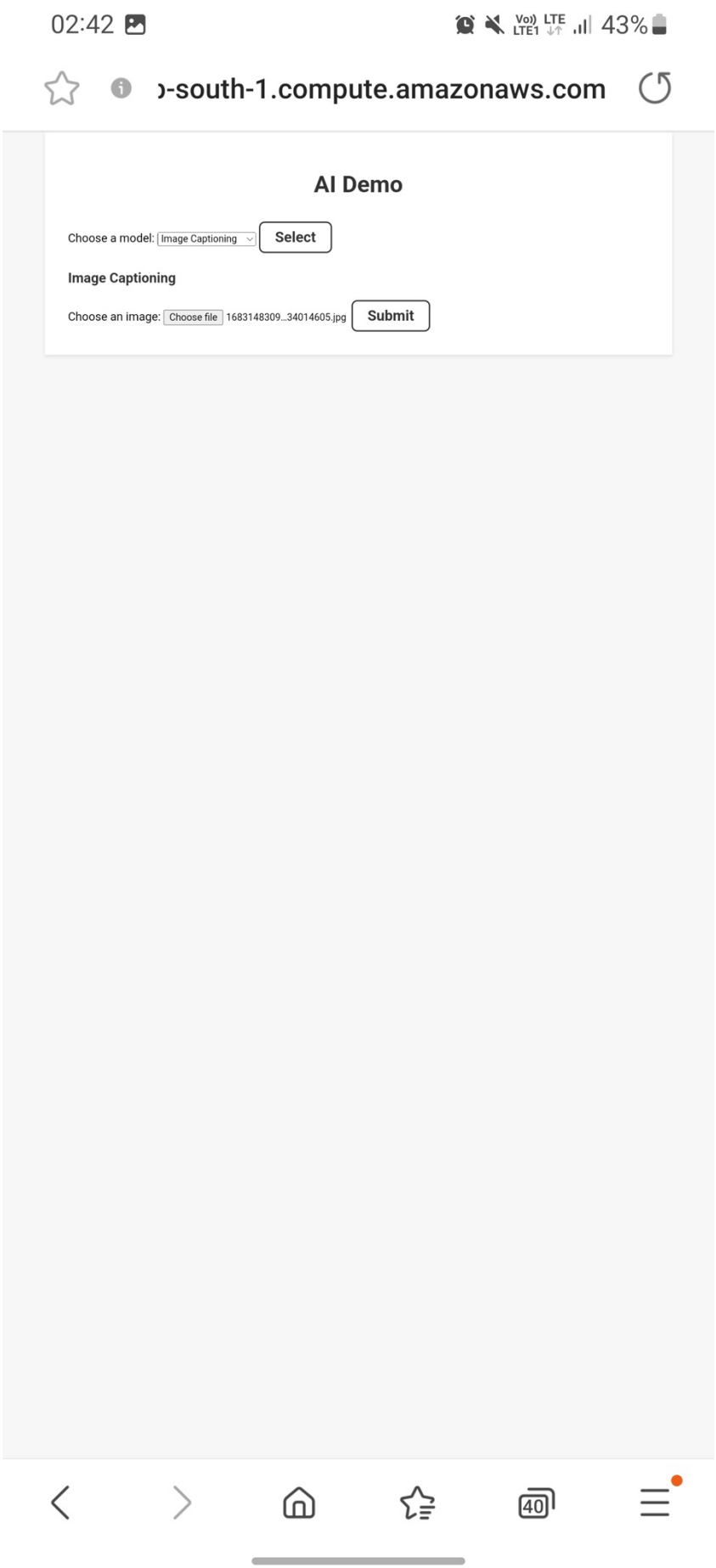


b. Mobile View



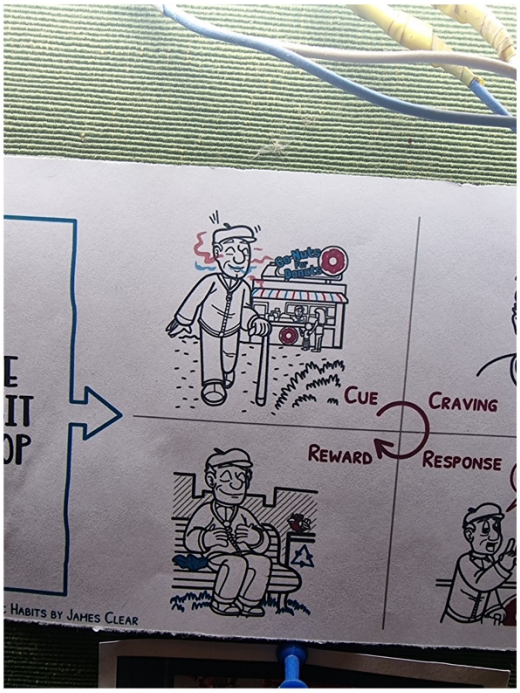



ɔ-south-1.compute.amazonaws.com
AI Demo
Choose a model: Image Captioning
Select
Image Captioning
Choose an image: Choose file No file chosen
Submit
Choose an action
Camera
My Files
Files


02:42
43%
-south-1.compute.amazonaws.com
AI Demo
Choose a model: Image Captioning
Select
Image Captioning
Choose an image: Choose file 1683148309...34014605.jpg
Submit
40

AI Demo Results

Image Captioning



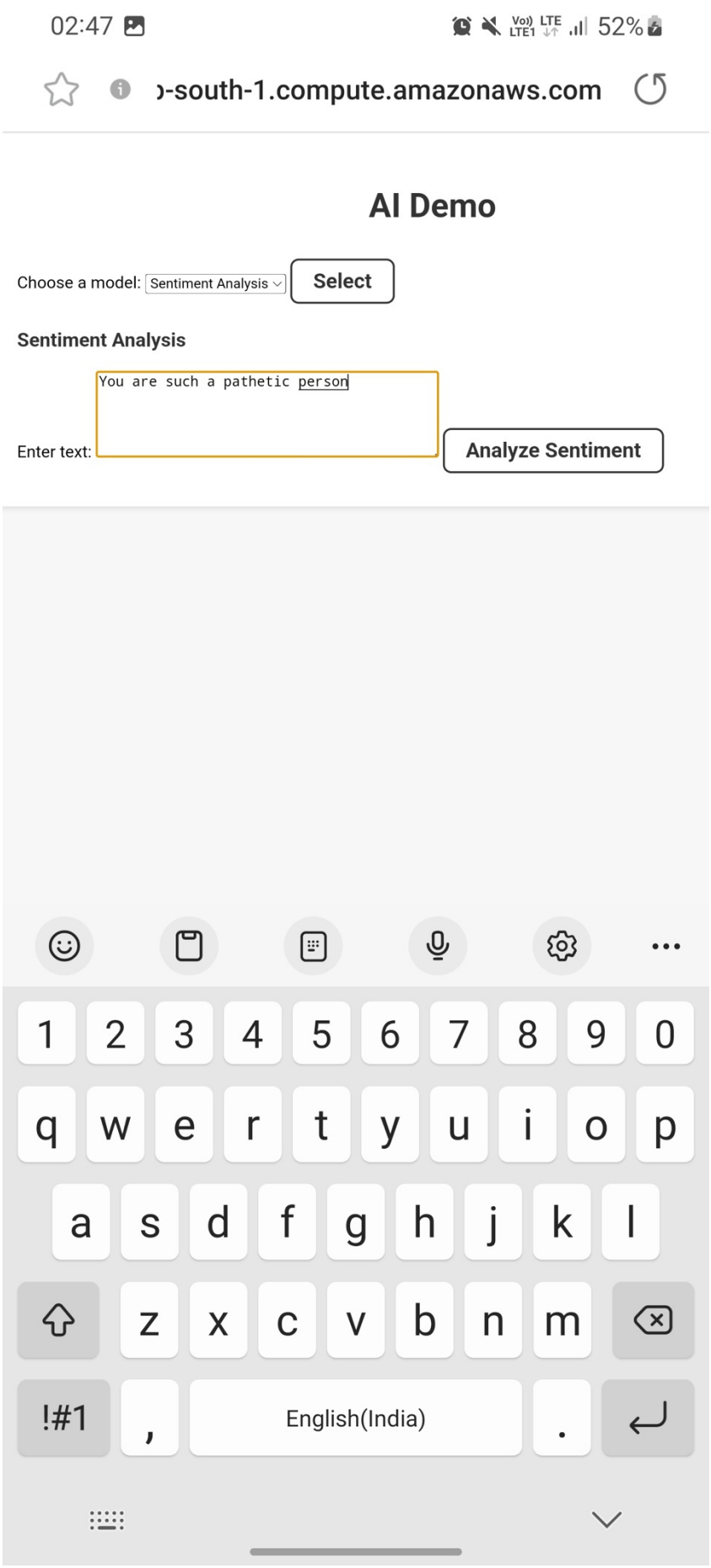
Greedy Search: man in hazard mask is sitting on the floor with his head on his chest

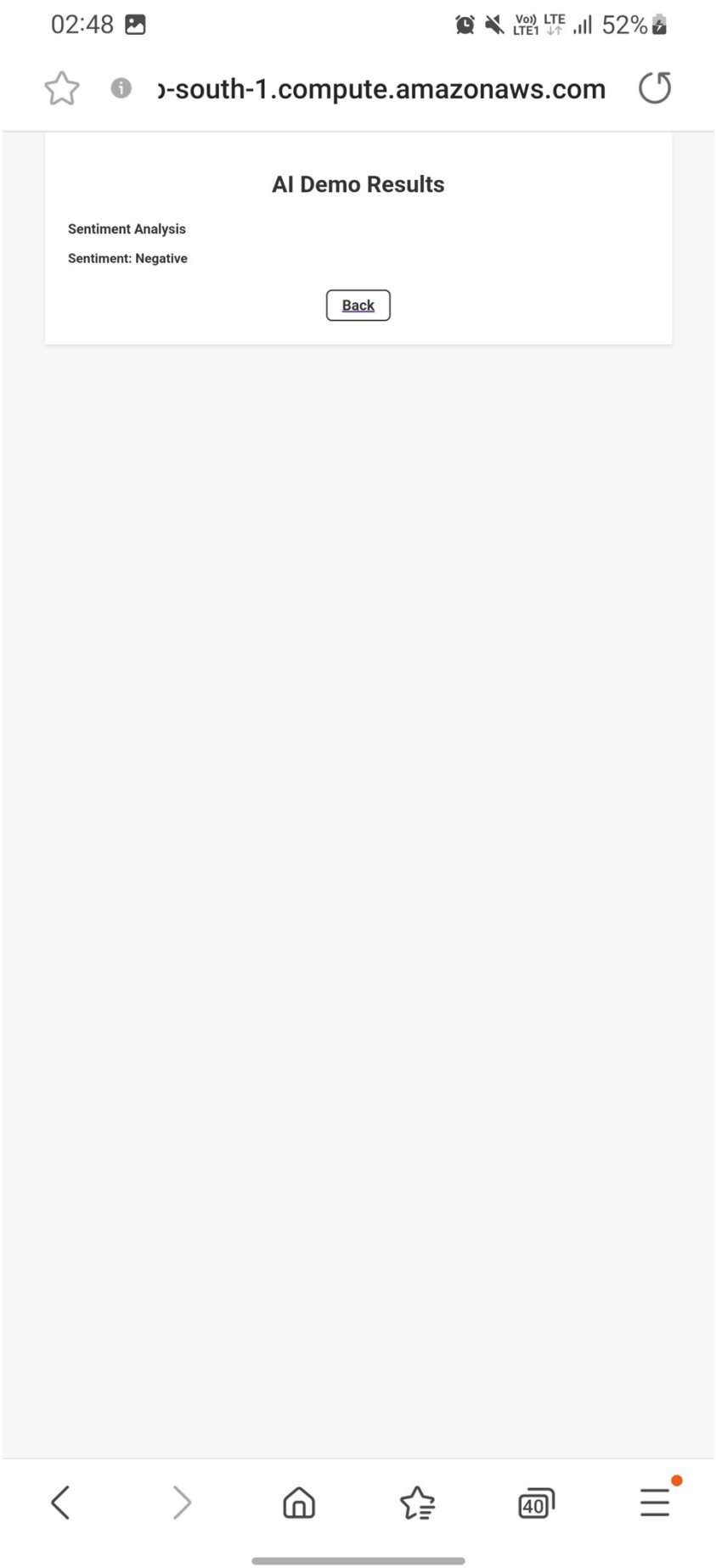
Beam Search (k=3): man in blue shirt is sleeping on the subway

Beam Search (k=5): man in blue shirt is sleeping on the subway

Back


02:46
49%
-south-1.compute.amazonaws.com
AI Demo
Choose a model: Sentiment Analysis
Select
Prev.
Next
Done
Image Captioning
Sentiment Analysis


02:47
52%
ɔ-south-1.compute.amazonaws.com
AI Demo
Choose a model: Sentiment Analysis
Select
Sentiment Analysis
You are such a pathetic person
Enter text:
Analyze Sentiment
1 2 3 4 5 6 7 8 9 0
q w e r t y u i o p
a s d f g h j k l
z x c v b n m
!#1
English(India)


02:48
52%
-south-1.compute.amazonaws.com
AI Demo Results
Sentiment Analysis
Sentiment: Negative
Back
40

Learning

In the realm of artificial intelligence, image captioning and sentiment analysis are two pivotal areas of research that have gained significant momentum in recent years. This project explores the development and deployment of an Image Captioning and Sentiment Analysis App on Amazon Web Services (AWS) Elastic Compute Cloud (EC2) in the context of a cloud computing lab project. Throughout the learning process, we not only improved our knowledge of machine learning and deep learning concepts but also gained valuable insights into deploying applications on a cloud platform.

The first stage of this project entailed the development of an Image Captioning model, employing convolutional neural networks (CNN) and recurrent neural networks (RNN) to generate meaningful captions for images. The model was trained using a vast dataset of images and corresponding captions to ensure its robustness and accuracy. Meanwhile, we also created a Sentiment Analysis model that leveraged natural language processing (NLP) techniques to analyze and classify the sentiment behind textual data.

The combination of Image Captioning and Sentiment Analysis models allowed us to create a comprehensive application that not only generated captions for images but also interpreted the emotions conveyed by the text. To make this application accessible to a wide audience, we chose to deploy it on AWS EC2, an industry-leading cloud platform offering scalable and reliable computing resources.

Deploying the application on AWS EC2 required us to familiarize ourselves with various AWS services and tools. This included learning how to configure EC2 instances, setting up security groups, and utilizing Amazon Simple Storage Service (S3) for data storage. Moreover, we delved into the practical aspects of configuring networking, load balancing, and auto-scaling to ensure optimal performance and availability of the application.

Throughout the project, we encountered numerous challenges that led to a deeper understanding of machine learning models, cloud computing concepts, and software development best practices. By overcoming these hurdles, we honed our problem-solving and analytical skills, preparing us for more complex projects in the future.

In conclusion, the Image Captioning and Sentiment Analysis App deployment on AWS EC2 in a cloud computing lab project provided a multi-faceted learning experience. This project not only expanded our knowledge of AI and cloud computing but also fostered invaluable skills in project management, collaboration, and technical expertise. As the demand for AI-driven applications continues to grow, the insights gleaned from this project will undoubtedly prove beneficial in future endeavors.

Conclusion

In conclusion, the Image Captioning and Sentiment Analysis App deployment project on AWS EC2 have successfully demonstrated the powerful synergy between artificial intelligence and cloud computing. By developing and integrating two advanced machine learning models, we created a versatile application capable of generating descriptive captions for images and analyzing the sentiment behind textual data. Deploying this application on the AWS EC2 platform allowed us to leverage the scalability, reliability, and flexibility offered by cloud computing, making the application accessible to a wide audience.

The project has provided invaluable learning experiences in several aspects, including enhancing our understanding of machine learning techniques, natural language processing, convolutional and recurrent neural networks, and the intricacies of deploying applications on cloud platforms. Furthermore, our skills in project management, problem-solving, and collaboration have been honed, equipping us to take on more complex projects in the future.

As artificial intelligence continues to make significant strides, the integration of AI-driven applications into various industries is becoming increasingly essential. The Image Captioning and Sentiment Analysis App project serve as a testament to the potential of AI in revolutionizing the way we process and understand visual and textual information. The insights and experiences gained through this project will undoubtedly prove to be an asset in our future endeavors, as we strive to develop innovative solutions that drive progress and improve our world.