Boom AI Technical Assessment Tests


/ Task 1: Q&A with PyTorch

Q&A with PyTorch

INSTRUCTIONS

Deploy a question-answering model using FastAPI and Docker by following this documentation.

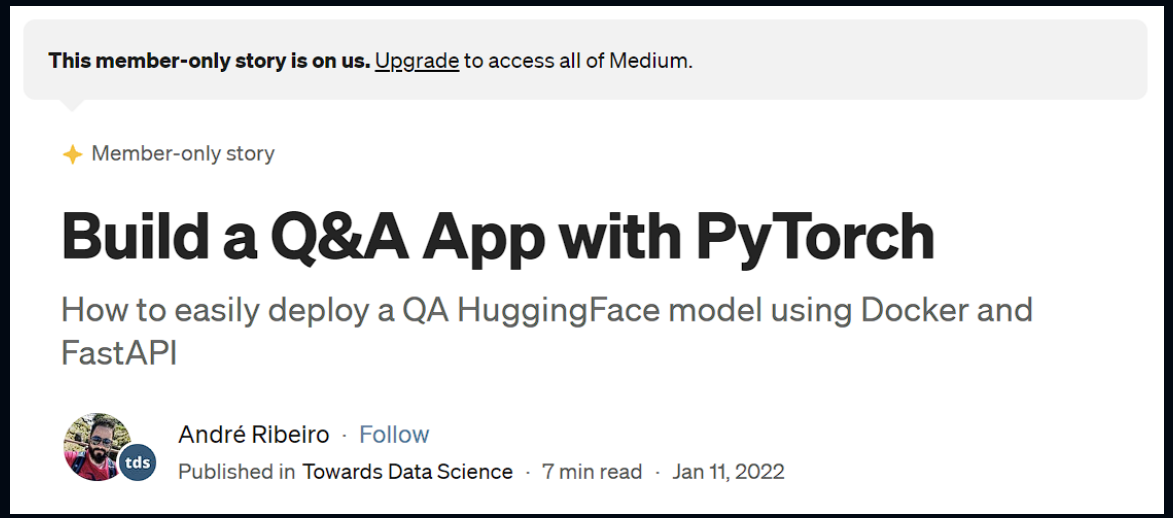
Below are the specifications:

a. Demonstrate the use of HuggingFace question-answering model
b. Use FastAPI to expose an endpoint
c. Use Docker to serve and deploy the endpoint
d. Bonus points: Host it on a free AWS EC2/GCP VM/DigitalOcean Droplet

Q&A with PyTorch

DOCUMENTATION

The documentation is a Medium member-only story. While Medium has a strong security on its member-only story, the easiest way to unlock it is to create a new Medium account.



Q&A with PyTorch

GETTING STARTED

I have not yet built a Question Answering (QA) model, but I am familiar with PyTorch and HuggingFace. The challenge for me is that I have not yet deployed a model using Docker and FastAPI. There's always a first.

The scripts are available in my public Github repository vbcalinao/boomai-mle.

Q&A with PyTorch

INSTALLATION

I have not yet built a Question Answering (QA) model, but I am familiar with PyTorch and HuggingFace. The challenge for me is that I have not yet deployed a model using Docker and FastAPI. There's always a first.

The scripts are available in my public Github repository vbcalinao/boomai-mle.

Q&A with PyTorch

CONTEXT BASED QA MODEL

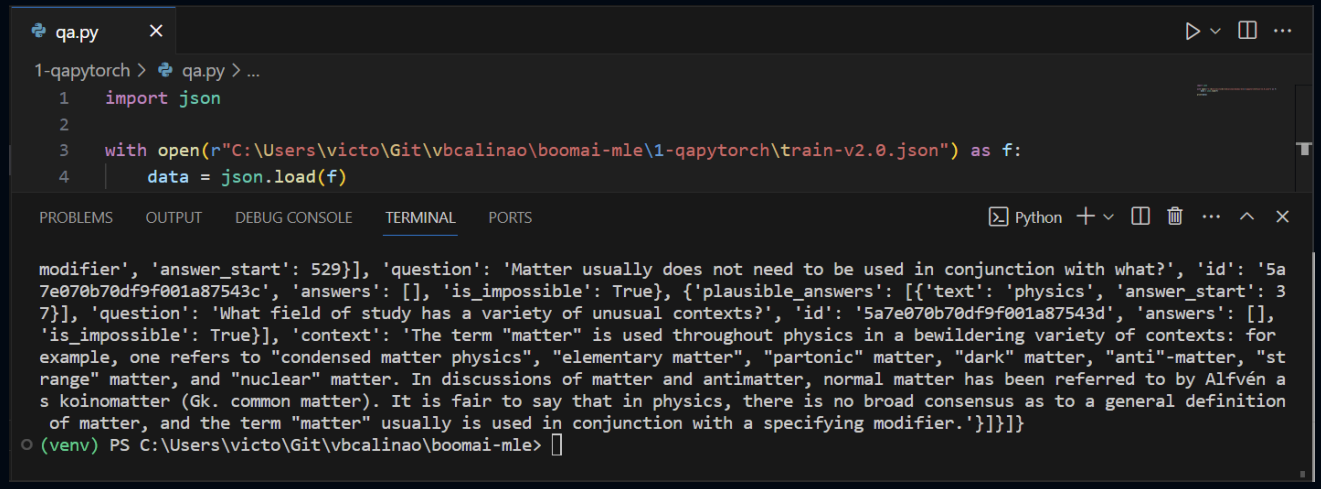
QA models are ubiquitous as they are present in virtual assistants like Siri and Google. "What's the weather today?" that's the QA model answering it for you. In the article, there are two common types of QA tasks

Knowledge-based QA model: encodes a large corpus of domain specific knowledge into the model and generates an answer based on the learned knowledge.
Context-based QA model: makes use of a given context and extracts the best paragraph / answer from that context.

Q&A with PyTorch

SQuAD 2.0

The dataset given offers context-based. It provides a passage of text along with questions that are to be answered based on that text. Upon examining, the dataset contains both answerable and unanswerable questions, requiring models to extract answers from the context where possible and to identify when questions cannot be answered using the provided passage.

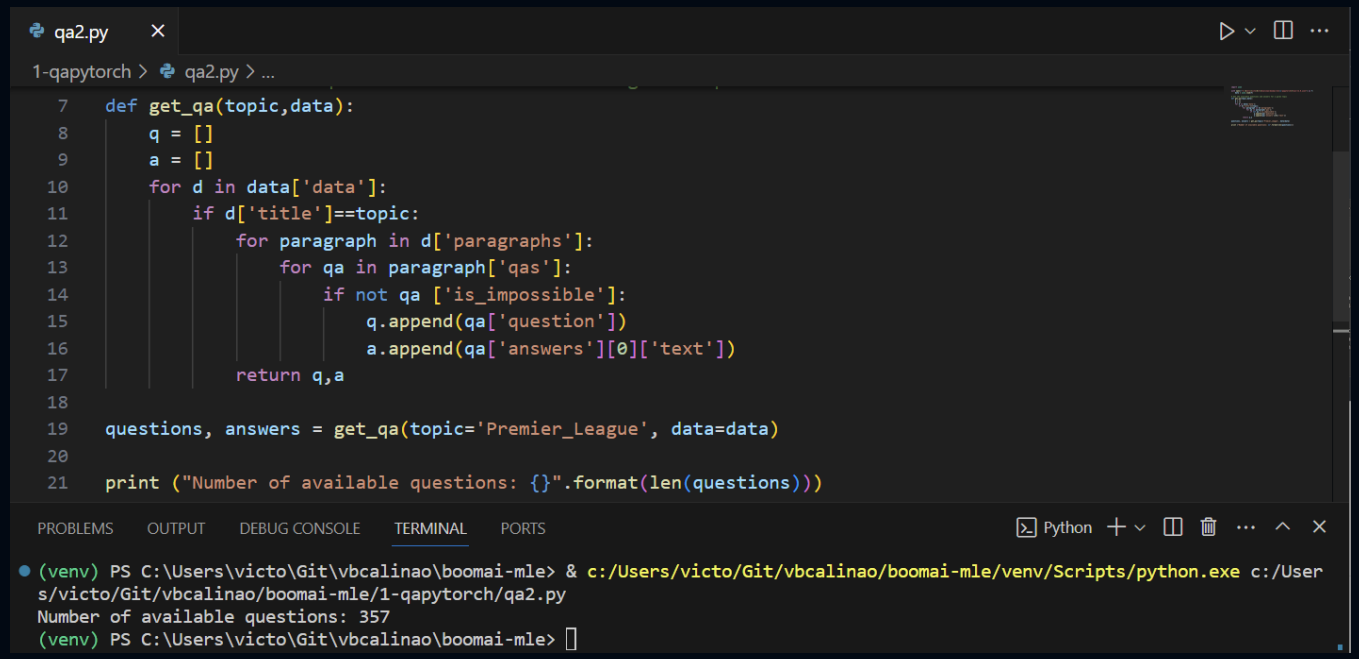


Q&A with PyTorch

TOPIC: PREMIER LEAGUE

The tutorial wants to focus on the 'questions' and 'answers' fields where the topic is 'Premier League'.

The code provided returns a set of 357 pairs of questions and answers.



Q&A with PyTorch

PRETRAINED MODEL

QA embedding model transforms QA into a format that a computer can understand. It converts text to numbers, then compare those numbers to find the best match, and use that match to provide the user with an answer.

Training a model from scratch is time consuming and expensive. This is where HuggingFace comes in. The shell script for downloading the pretrained model from HuggingFace is provided, however, the provided script is for Unix-like or MacOS (definitelty from MacOS) because it uses 'wget'. As a Windows user, use 'curl' instead.

Q&A with PyTorch

PRETRAINED EMBEDDING MODEL

$ downloadmodel.sh ×

1-qapytorch > $ downloadmodel.sh

```
#!/bin/bash

# defines the qa model
MODEL_DIR="https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2/resolve/main"
MODEL_NAME="paraphrase-MiniLM-L6-v2"

# downloads the qa model. To make this image more general one can use curl
# with the "-O" argument to download the necessary files defined
# in "require.txt".

mkdir ${MODEL_NAME} &&\
    curl -o ${MODEL_NAME}/vocab.txt ${MODEL_DIR}/vocab.txt &&\
    curl -o ${MODEL_NAME}/tokenizer_config.json ${MODEL_DIR}/tokenizer_config.json &&\
    curl -o ${MODEL_NAME}/tokenizer.json ${MODEL_DIR}/tokenizer.json &&\
    curl -o ${MODEL_NAME}/special_tokens_map.json ${MODEL_DIR}/special_tokens_map.json &&\
    curl -o ${MODEL_NAME}/sentence_bert_config.json ${MODEL_DIR}/sentence_bert_config.json &&\
    curl -o ${MODEL_NAME}/pytorch_model.bin ${MODEL_DIR}/pytorch_model.bin &&\
    curl -o ${MODEL_NAME}/modules.json ${MODEL_DIR}/modules.json &&\
    curl -o ${MODEL_NAME}/config_sentence_transformers.json ${MODEL_DIR}/config_sentence_transformers.json &&\
    curl -o ${MODEL_NAME}/config.json ${MODEL_DIR}/config.json
```

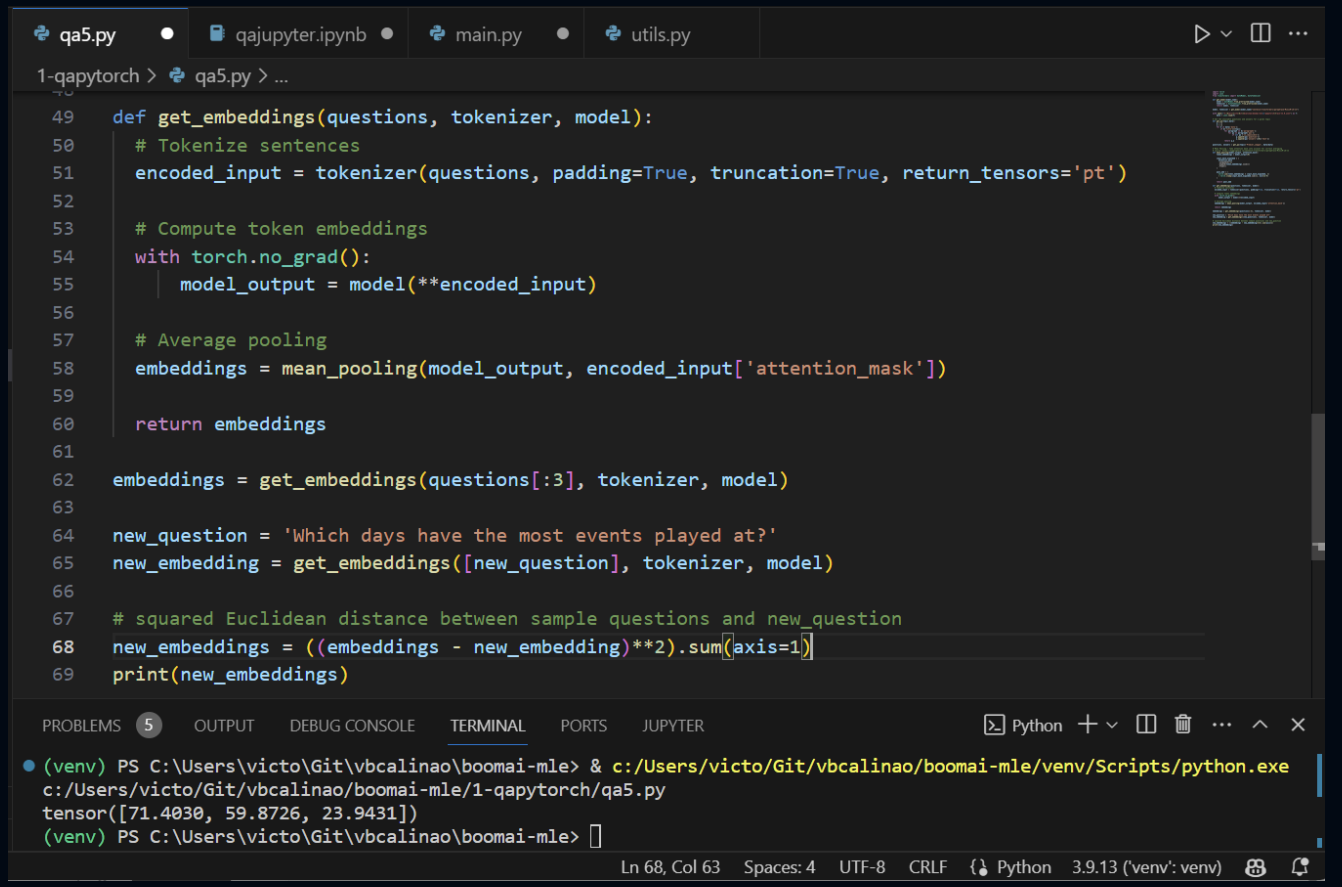
Q&A with PyTorch

TESTING MODEL LOCALLY

We test the similarity of the context to a new question.

'Which days have the most events played at?'

The code should output the distances, indicating that the last question in our sample is indeed the closest (smallest distance) to our new question.

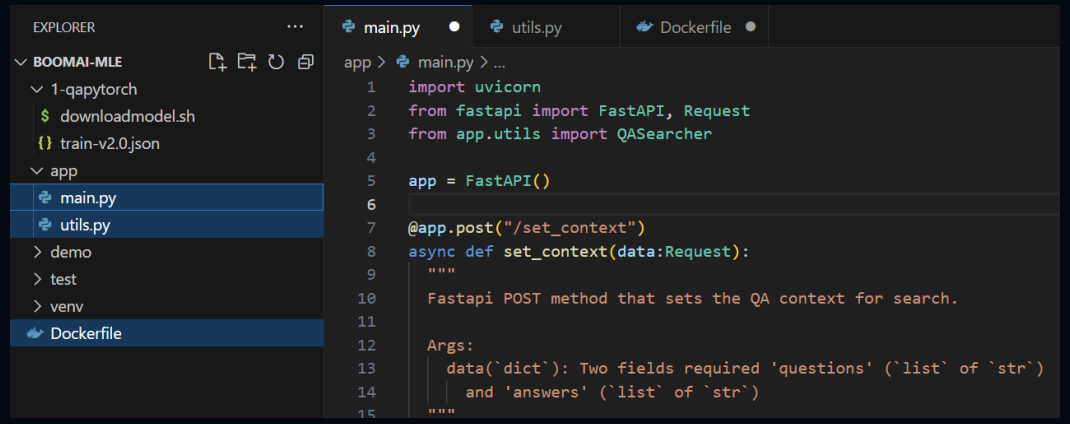


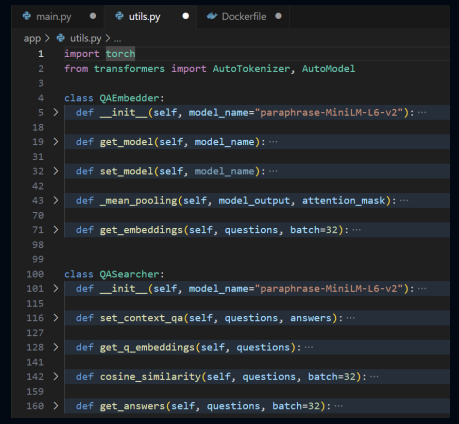
Q&A with PyTorch

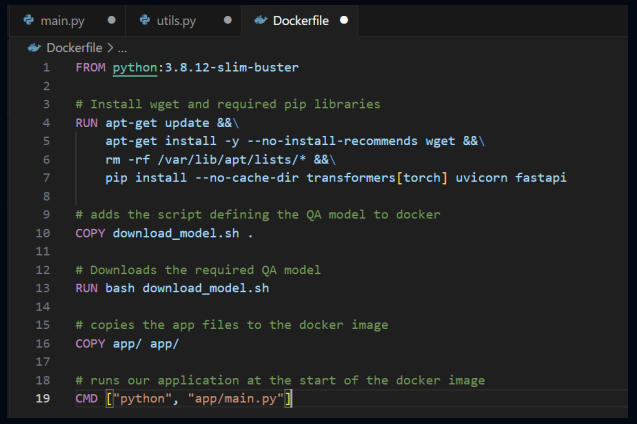
MODEL DEPLOYMENT USING DOCKER AND FASTAPI

Now that we are going to deploy the model, the tutorial proposes three steps:

1. Wrap the previous functions in one or more easy to use classes;
2. Define an app and call the required class methods through HTTP;
3. Wrap the whole app and dependencies in a container for easy scalability.



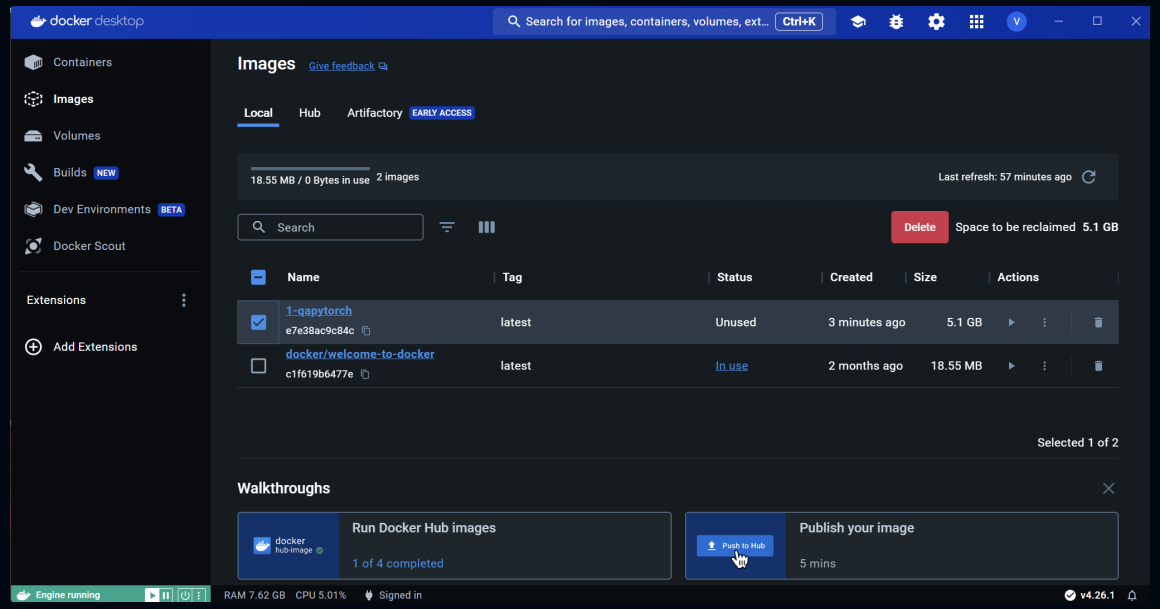


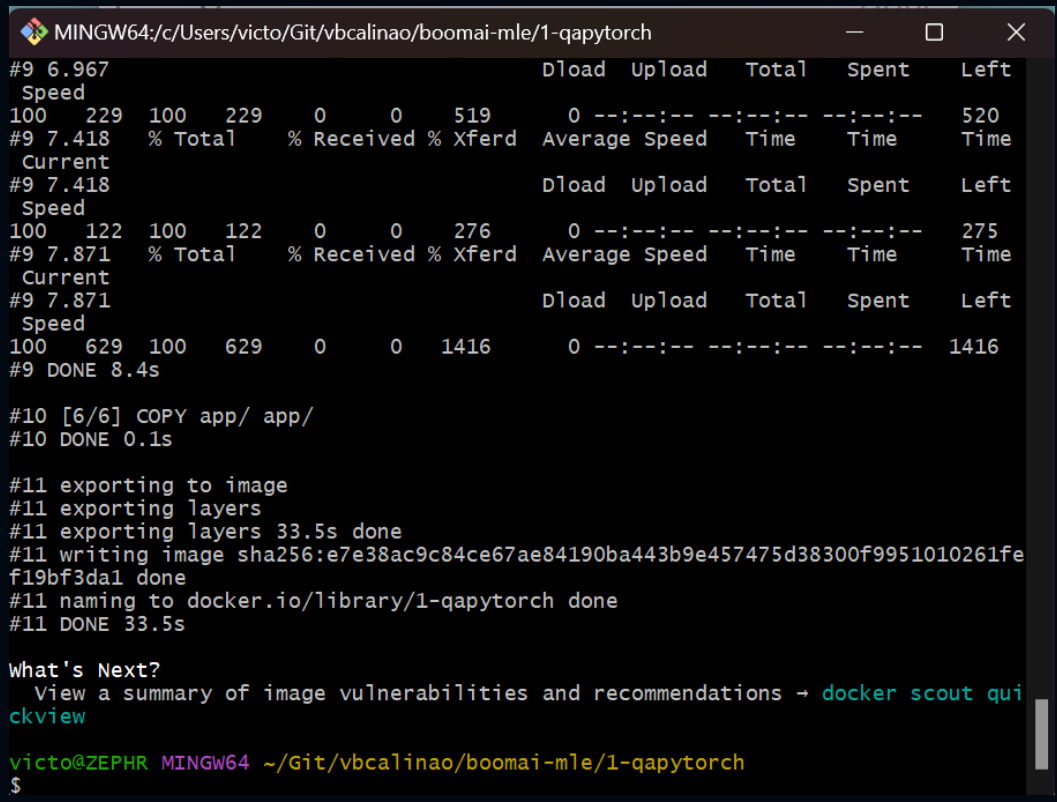


Q&A with PyTorch

DOCKER

1. Building docker on Windows needs 'curl'. I've edited the dockerfile to install curl.

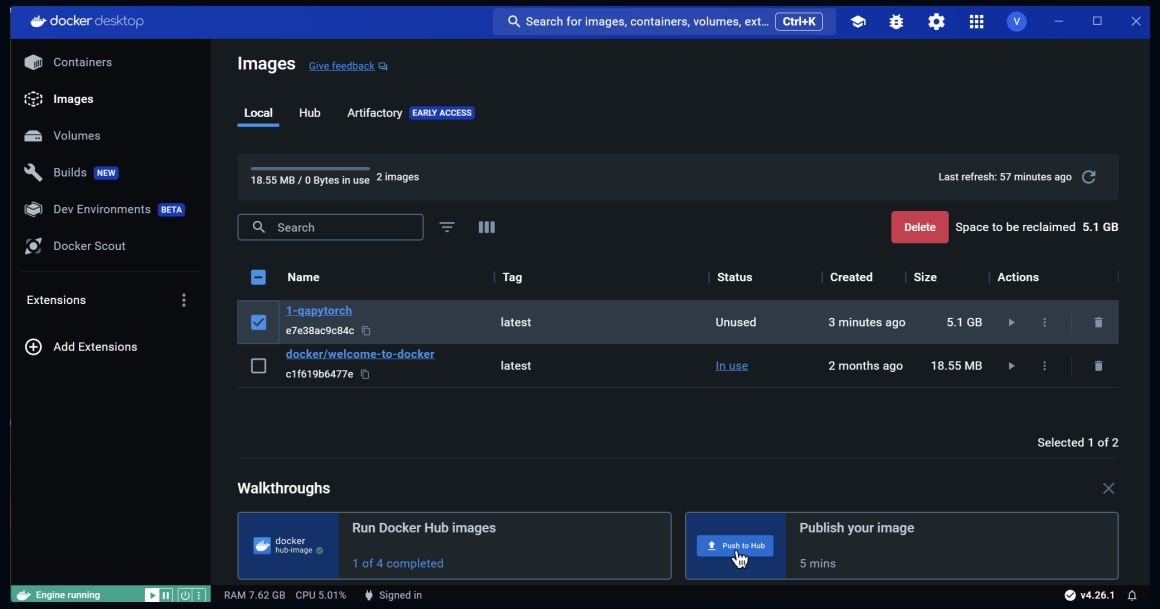


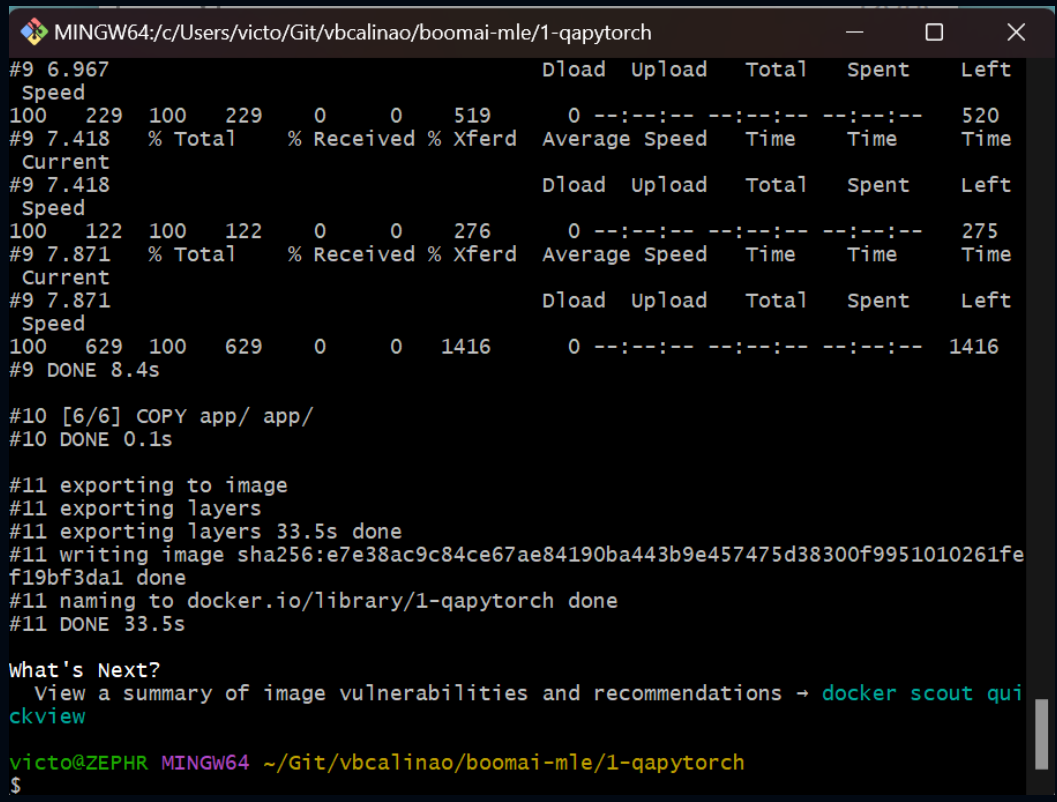


Q&A with PyTorch

DOCKER

1. Building docker on Windows needs 'curl'. I've edited the dockerfile to install curl.

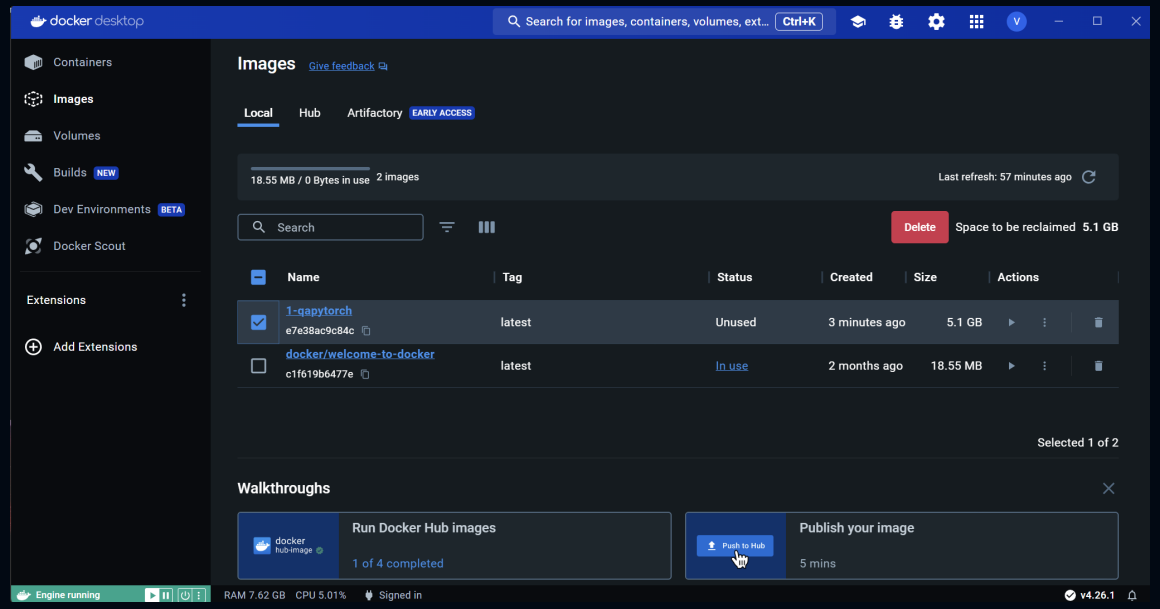


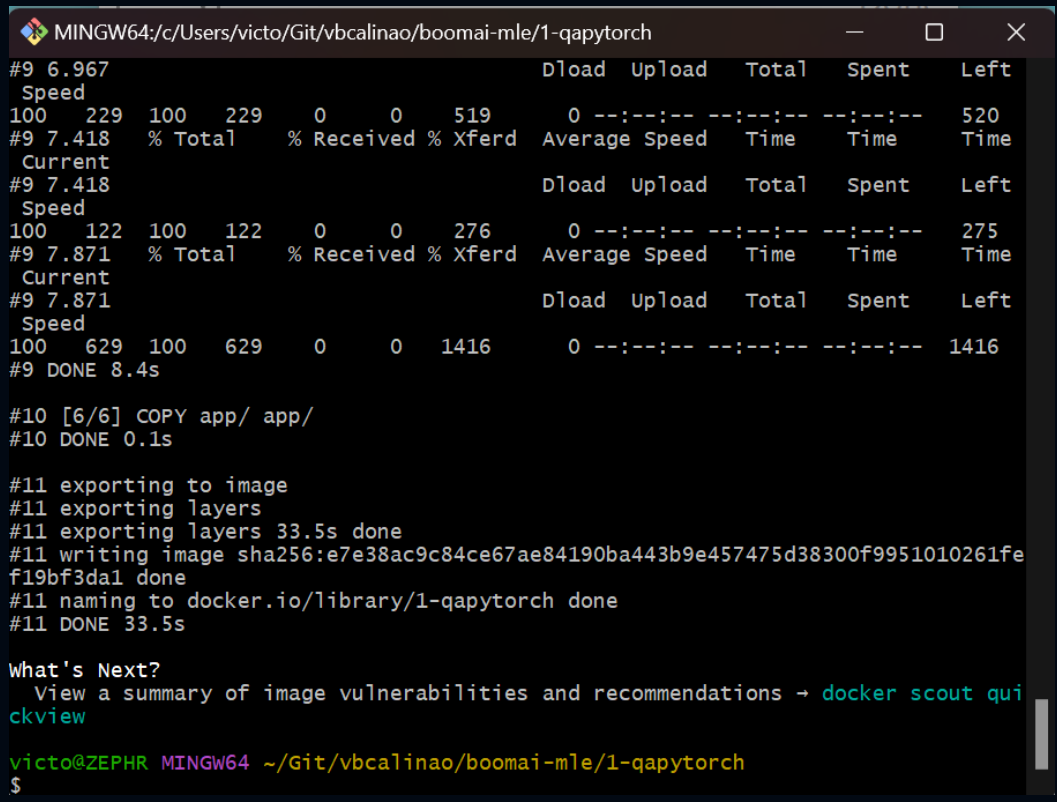


Q&A with PyTorch

DOCKER

1. Building docker on Windows needs 'curl'. I've edited the dockerfile to install curl.

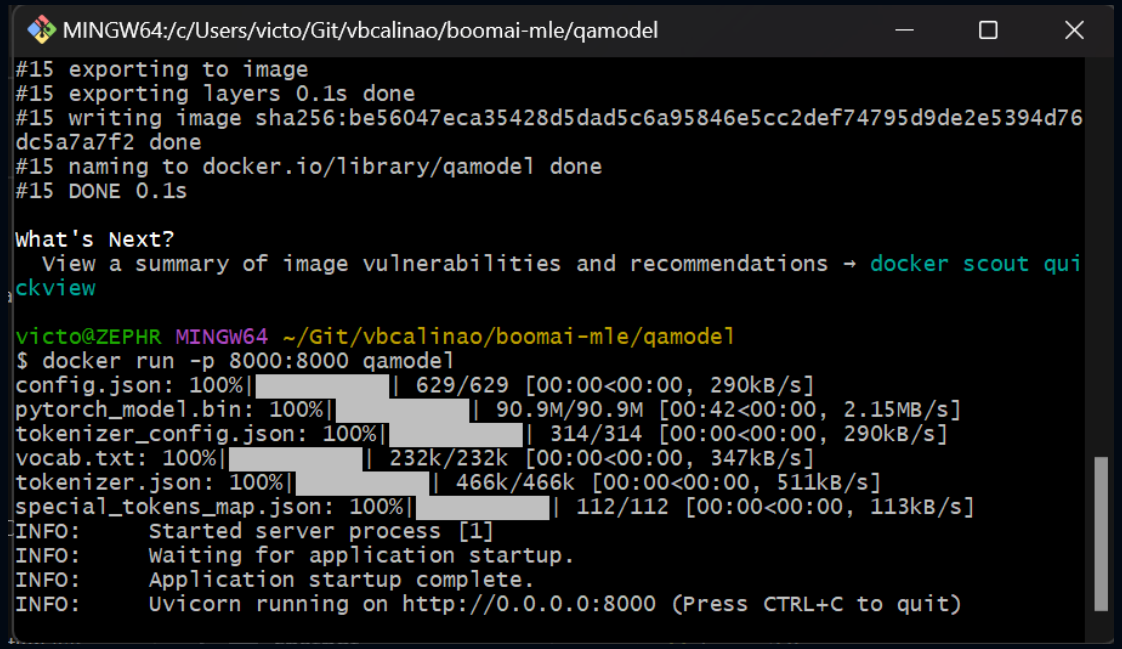




Q&A with PyTorch

DOCKER RUN

Nice! successfully started your Docker container and the application inside it is running as expected. The application is a FastAPI server running on Uvicorn, and it's listening for HTTP requests on port 8000.



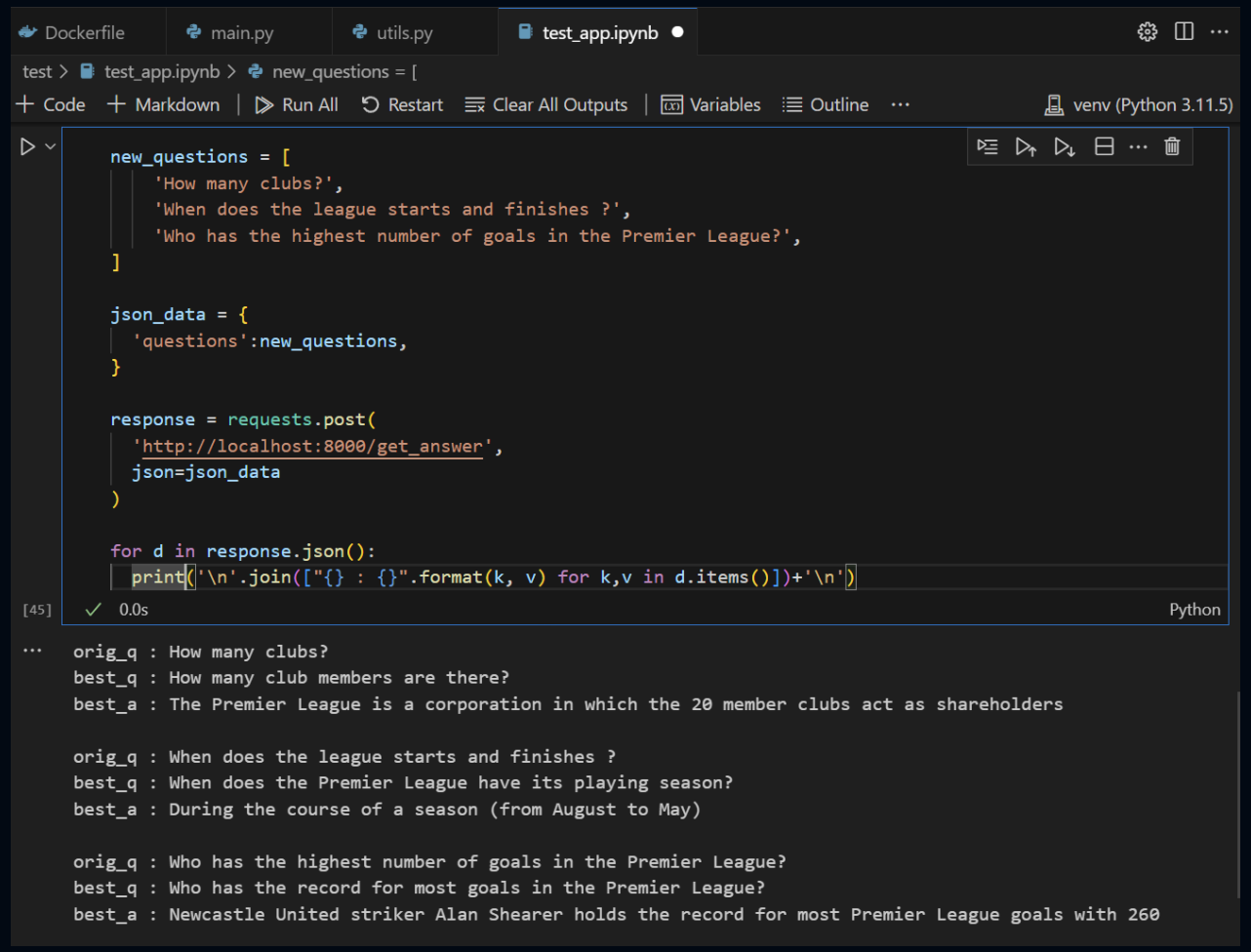
Q&A with PyTorch

TEST APP

This is a test app code that is used to interact with a local server that can answer the provided new questions.

The server is expected to return a response where each item is a dictionary containing the answers to the questions.

Voila! This is our own Q&A app.




/ Task 2: AAI's Speech-to-text

Speech-to-text

INSTRUCTIONS

Go through Assembly AI's documentation and script out, using Python, a real-time speech-to-text transcription service. The code itself is literally on the front page of the website. Below are the specifications:

a. Demonstrate, using your mic, that you can perform live transcription from speech and display the text either in a terminal or a simple web UI.
b. As part of the transcription service, identify the keywords that end in a vowel and append a "-v" to that word. (don't build a machine learning model. Just perform classic deterministic identification) i. Transcript "The quick brown fox" becomes "The-v quick brown fox"
c. As part of the transcription service, identify the keywords that end in a consonant and append a "-c" to that word. (don't build a machine learning model. Just perform classic deterministic identification) i. Transcript "The quick brown fox" becomes "The-v quick-c brown-c fox-c"
d. Bonus points: Host it on a free AWS EC2/GCP VM/DigitalOcean Droplet

Speech-to-text

DOCUMENTATION

Assembly AI's website is 'maganda.' Aside from that, I just get started and had my first transcription in no time.

START BUILDING WITH ASSEMBLYAI

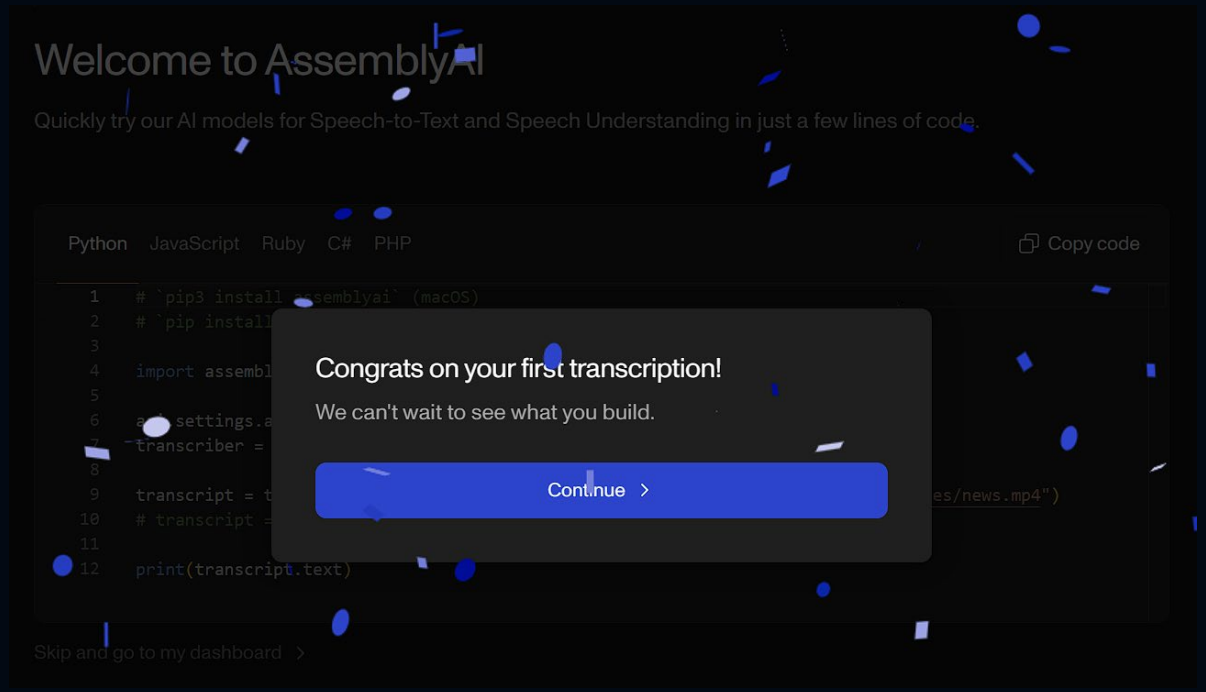
Get started in seconds

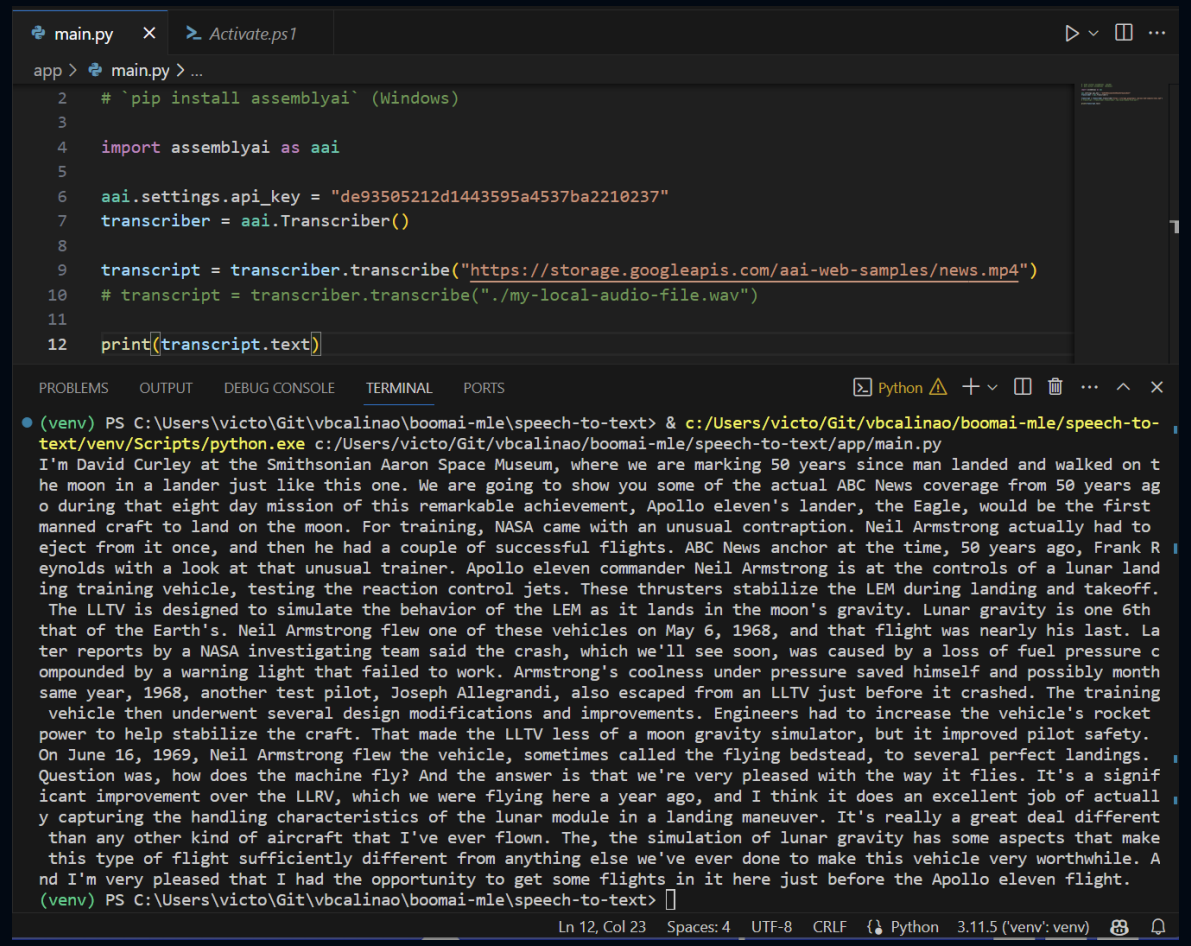
Use our API | Contact sales

```python
import assemblyai as aai

transcriber = aai.Transcriber()
transcript = transcriber.transcribe(URL, config)

print(transcript.text)
```

Speech-to-text

GETTING STARTED IN SECONDS (literally)



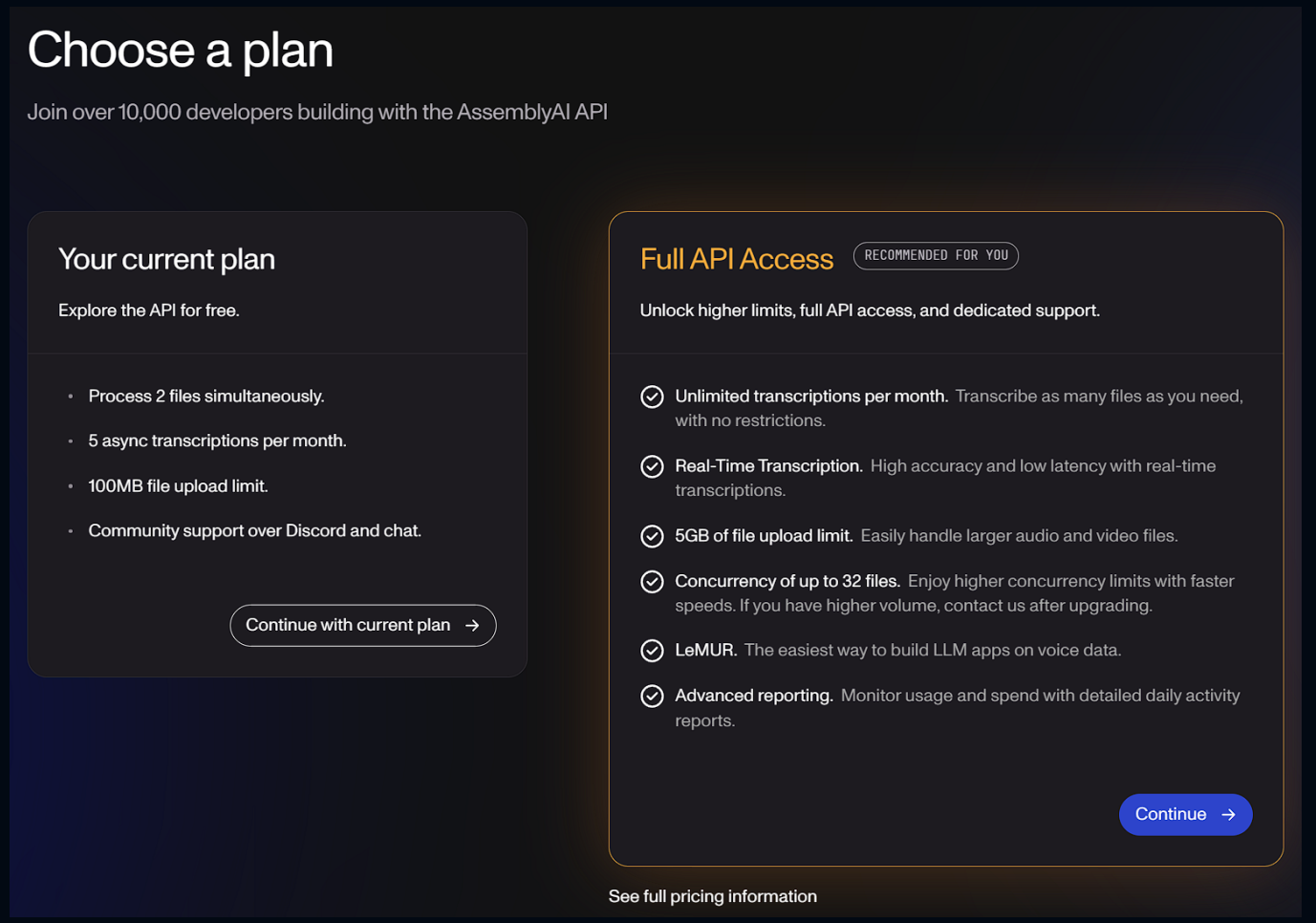


Speech-to-text

FREE TRIAL

As of this writing, real-time transcription comes with a cost, for a minimum of $8, we'll get the Full API Access and real-time transcriptions.

The blocker is that the current plan don't have real-time transcription. We need to upgrade and get the billing account

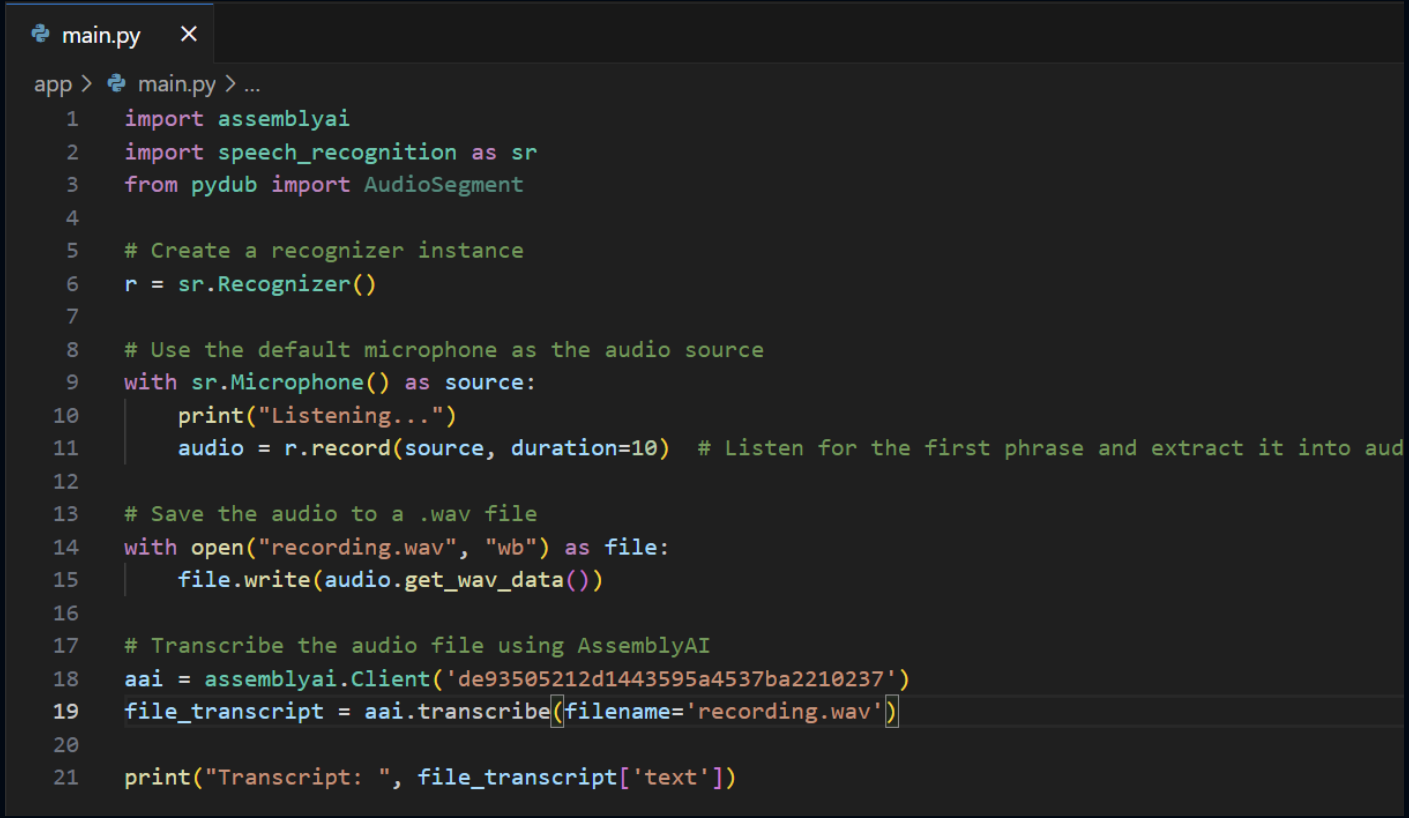


Speech-to-text

WORKAROUND

As a workaround, we'll use the local device microphone as the audio source, then it will listen for 5 seconds, will save the recording in a wav file and simultaneously transcript the file afterwards.

So, we can treat this as a "one-file 5 second delay live transcription"



```python
import assemblyai
import speech_recognition as sr
from pydub import AudioSegment

# Create a recognizer instance
r = sr.Recognizer()

# Use the default microphone as the audio source
with sr.Microphone() as source:
    print("Listening...")
    audio = r.record(source, duration=10)  # Listen for the first phrase and extract it into aud

# Save the audio to a .wav file
with open("recording.wav", "wb") as file:
    file.write(audio.get_wav_data())

# Transcribe the audio file using AssemblyAI
aai = assemblyai.Client('de93505212d1443595a4537ba2210237')
file_transcript = aai.transcribe(filename='recording.wav')

print("Transcript: ", file_transcript['text'])
```

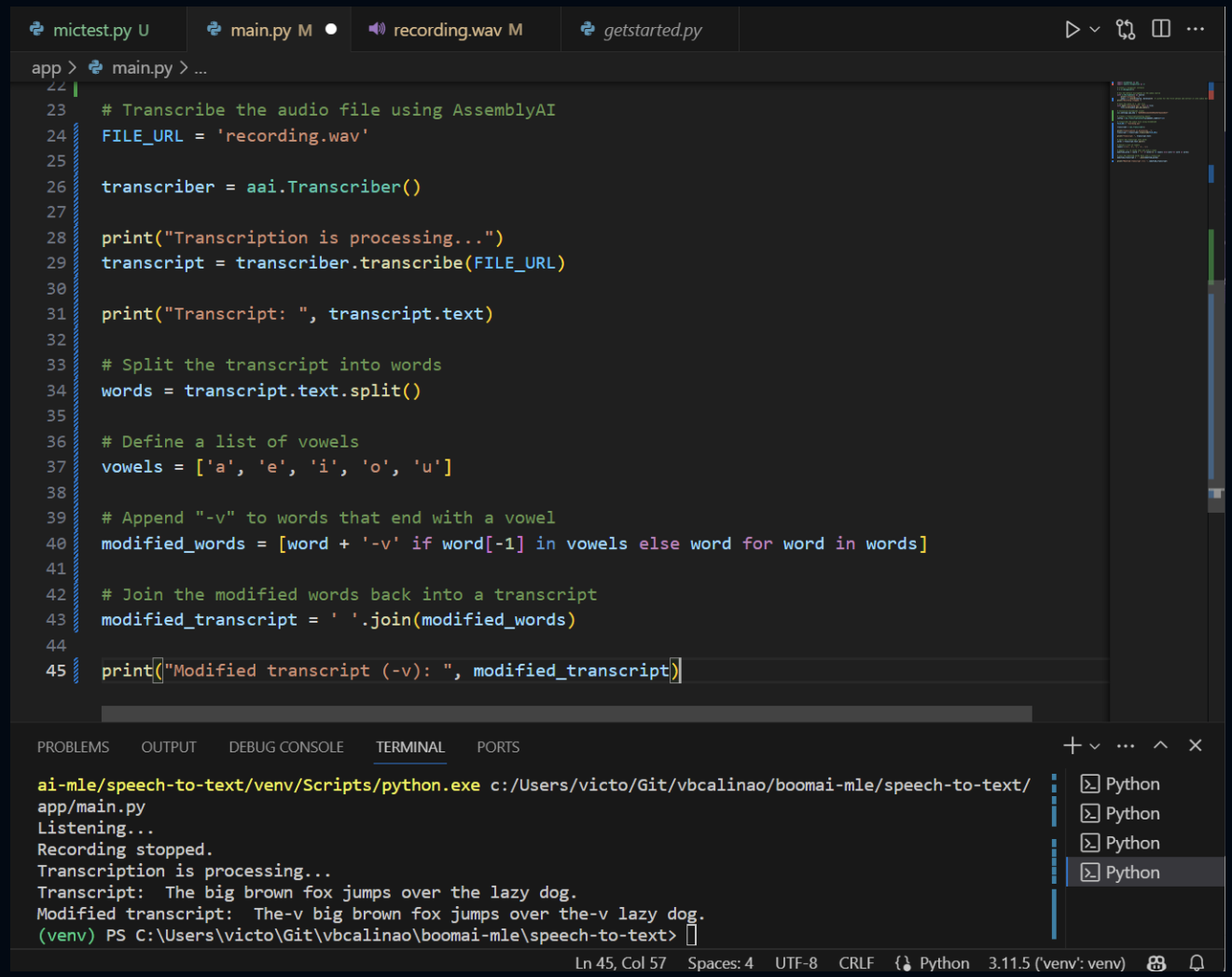
Speech-to-text

LIVE TRANSCRIPTION

Speech-to-text

-v VOWEL CHALLENGE

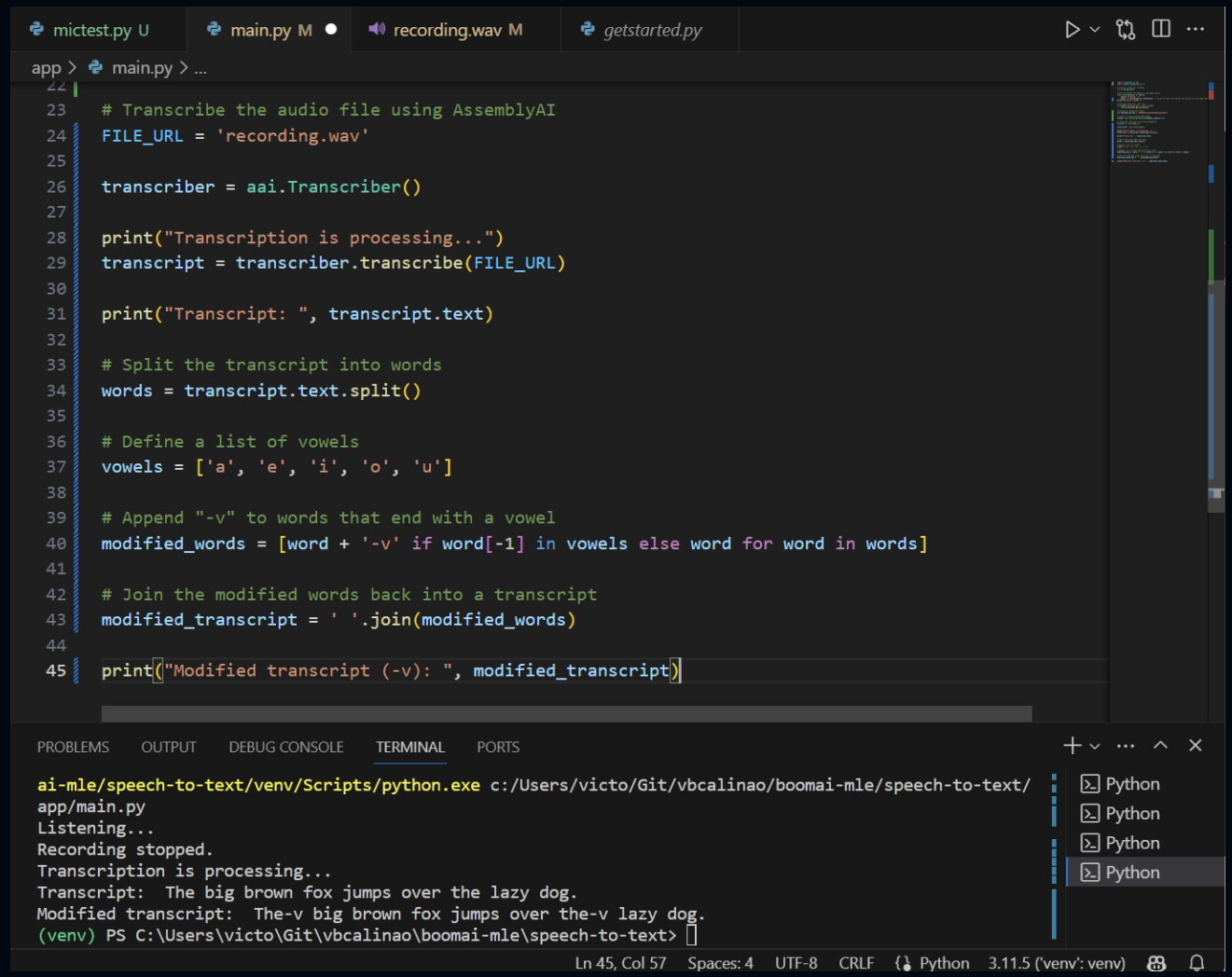
Performing classic deterministic identification for vowels, we can achieve this by splitting the transcript into words, checking the last letter of each word, and appending "-v" to words that end with a vowel.



Speech-to-text

-v VOWEL CHALLENGE

Performing classic deterministic identification for vowels, we can achieve this by splitting the transcript into words, checking the last letter of each word, and appending "-v" to words that end with a vowel.



Speech-to-text

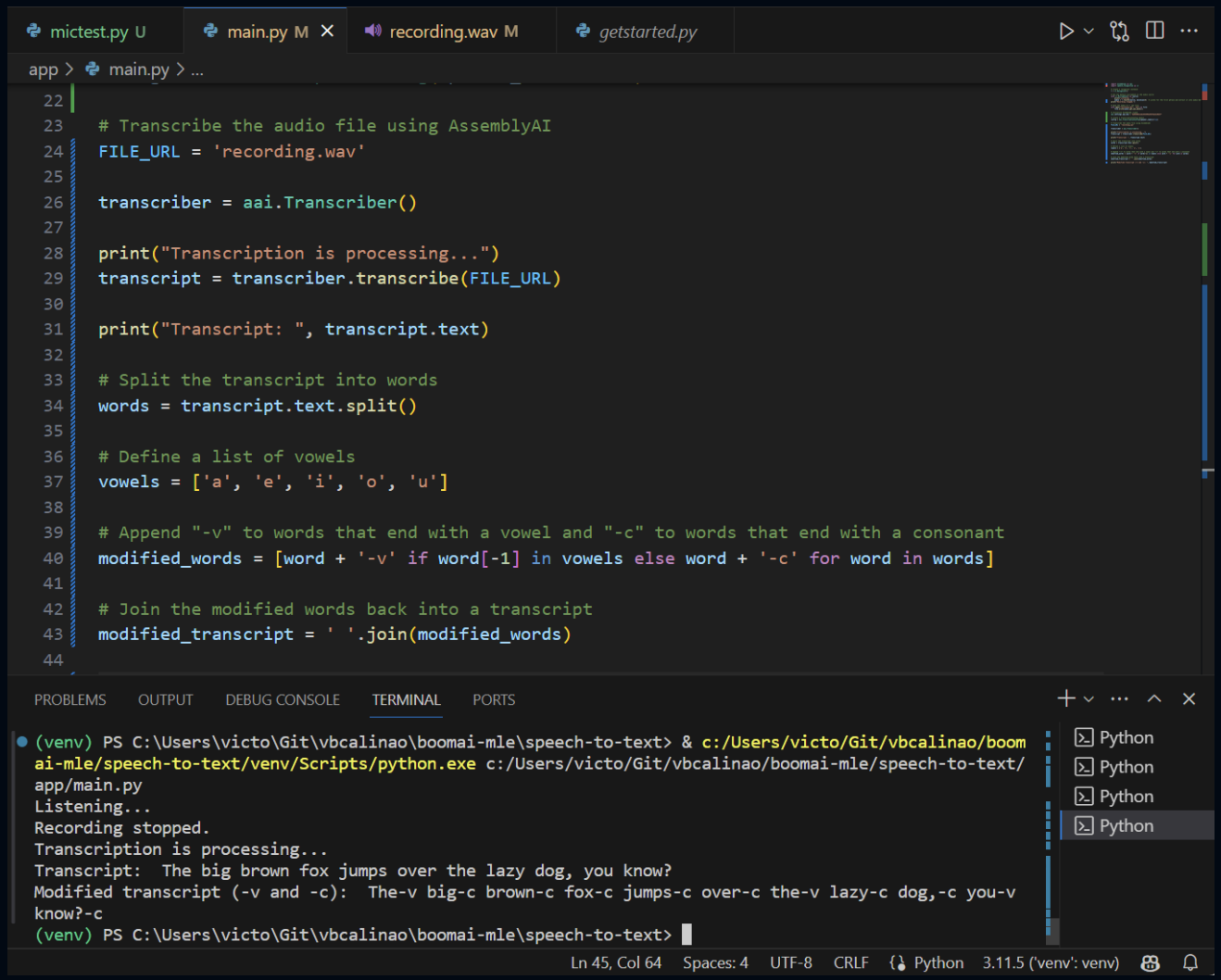
-c CONSONANT CHALLENGE

Performing classic deterministic identification for consonants, we can achieve this by using our current code and adding an else function that adds '-c'.

Now we get,

Transcript: The big brown fox jumps over the lazy dog, you know?

Modified transcript (-v and -c): The-v big-c brown-c fox-c jumps-c over-c the-v lazy-c dog,-c you-v know?-c



Speech-to-text

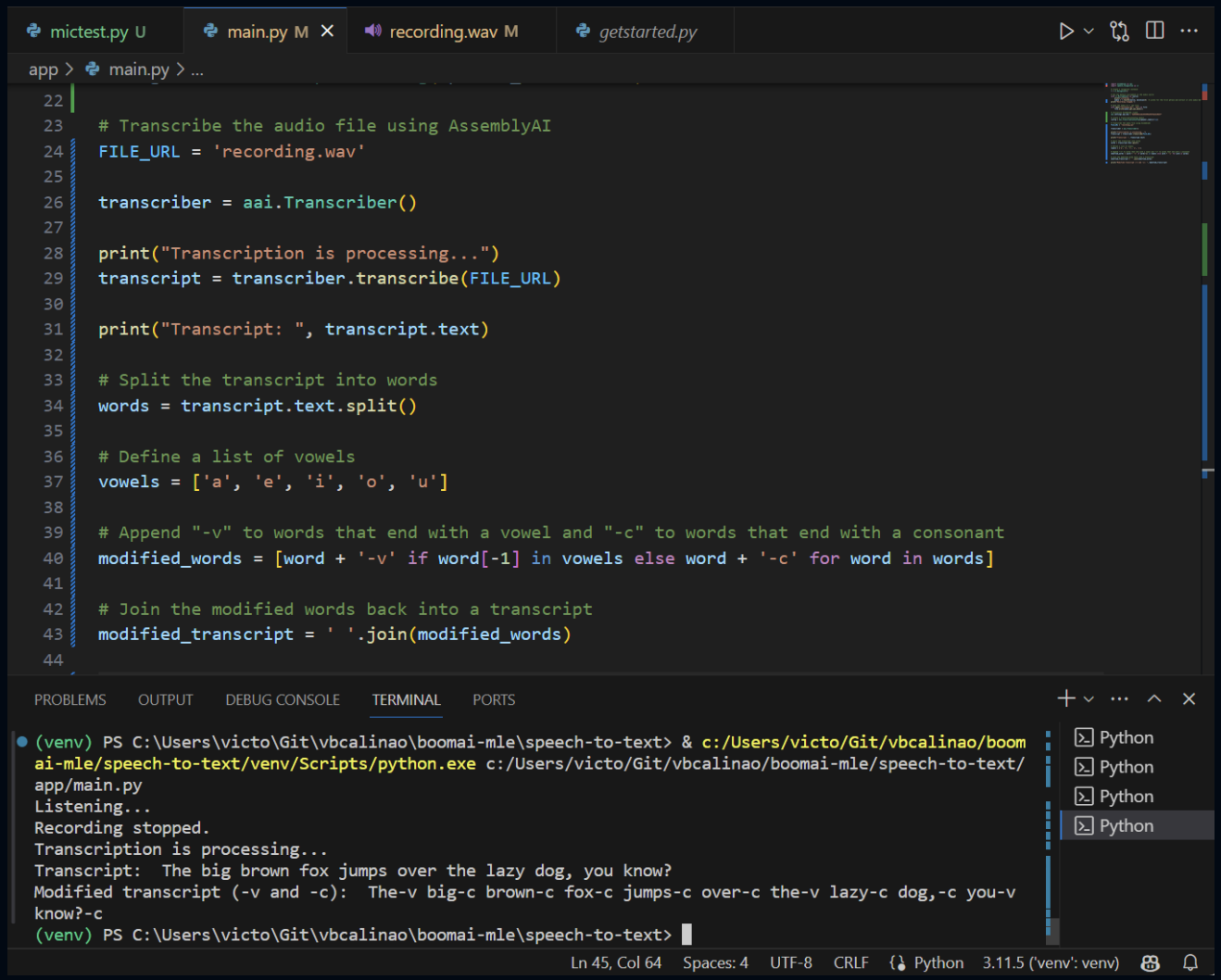
-c CONSONANT CHALLENGE

Performing classic deterministic identification for consonants, we can achieve this by using our current code and adding an else function that adds '-c'.

Now we get,

Transcript: The big brown fox jumps over the lazy dog, you know?

Modified transcript (-v and -c): The-v big-c brown-c fox-c jumps-c over-c the-v lazy-c dog,-c you-v know?-c




/ Task 3: BART Text Summarization

Text Summarization

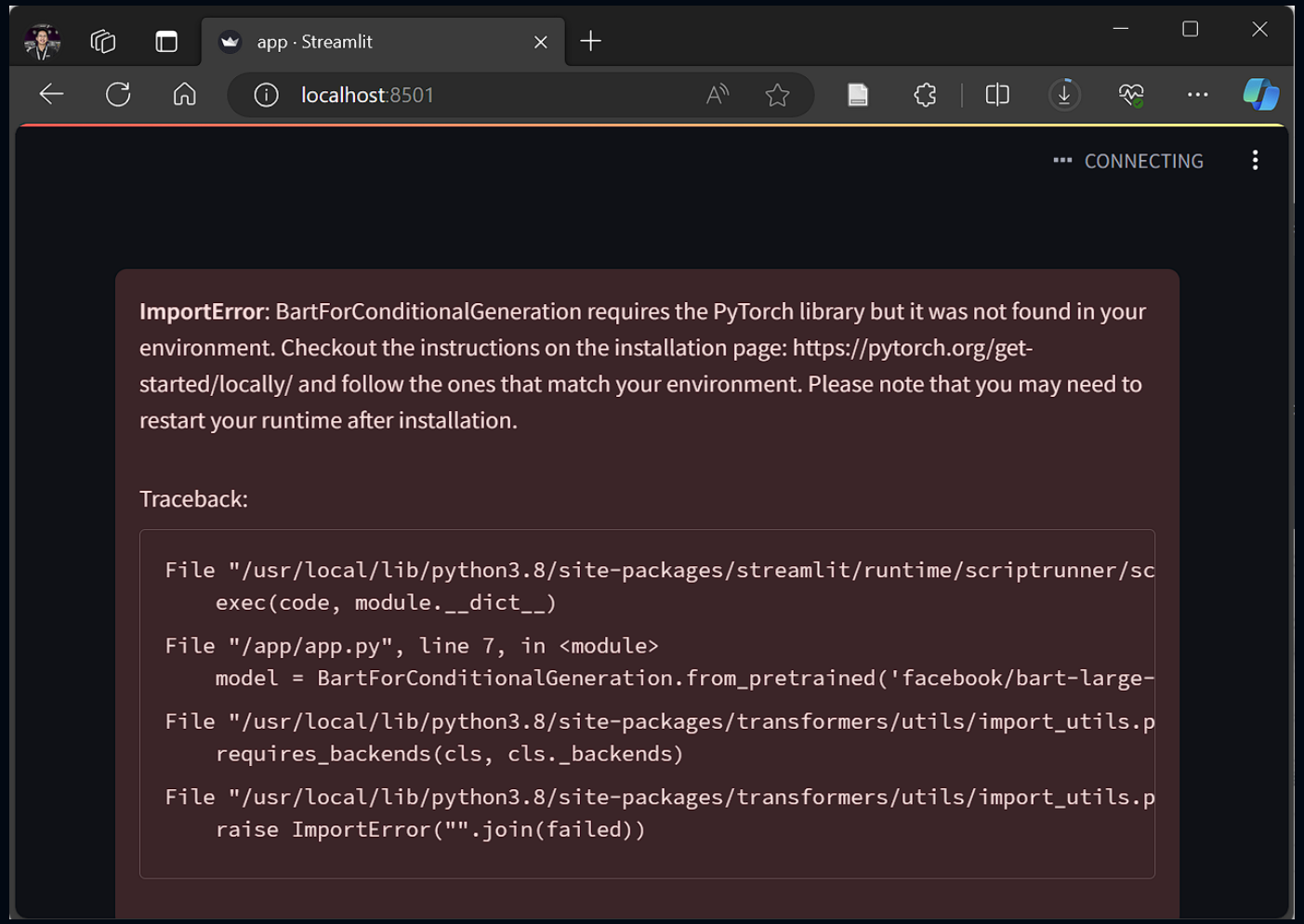
INSTRUCTIONS

Deploy a text summarization model using the BART model from Hugging Face. Below are the specifications:

a. Utilize the BART pre-trained model from the Hugging Face Transformers library (important: use a variant suitable for text summarization)
b. Develop a web interface using Streamlit that allows users to input text and then displays the summarized text
c. Containerize the Streamlit application using Docker
d. Note: you may use the text in Appendix A to demonstrate your summarization model
e. Bonus points: Host it on a free AWS EC2/GCP VM/DigitalOcean Droplet

Text Summarization

CHALLENGE




Ask anything.