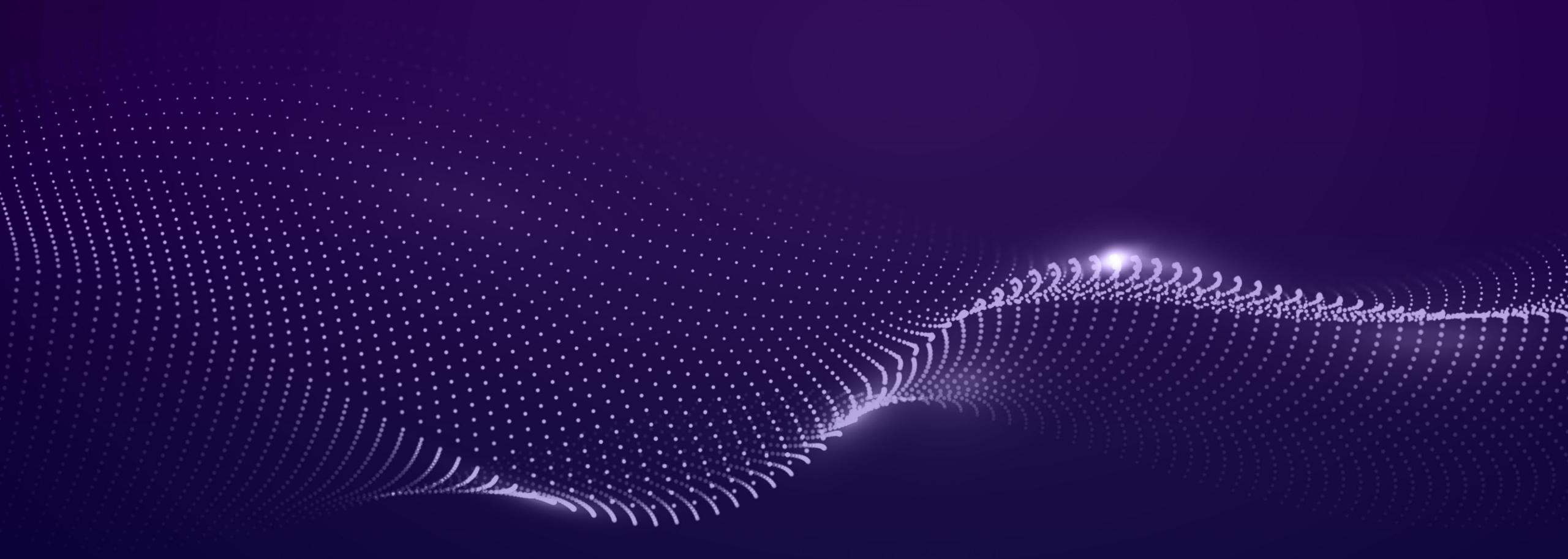


# Boom AI Technical Assessment Tests



# / 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.

This member-only story is on us. [Upgrade](#) to access all of Medium.

★ Member-only story

## Build a Q&A App with PyTorch

How to easily deploy a QA HuggingFace model using Docker and FastAPI

 André Ribeiro · [Follow](#)  
Published in Towards Data Science · 7 min read · Jan 11, 2022

# 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/booma-mle](https://github.com/vbcalinao/booma-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/booma-mle](https://github.com/vbcalinao/booma-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

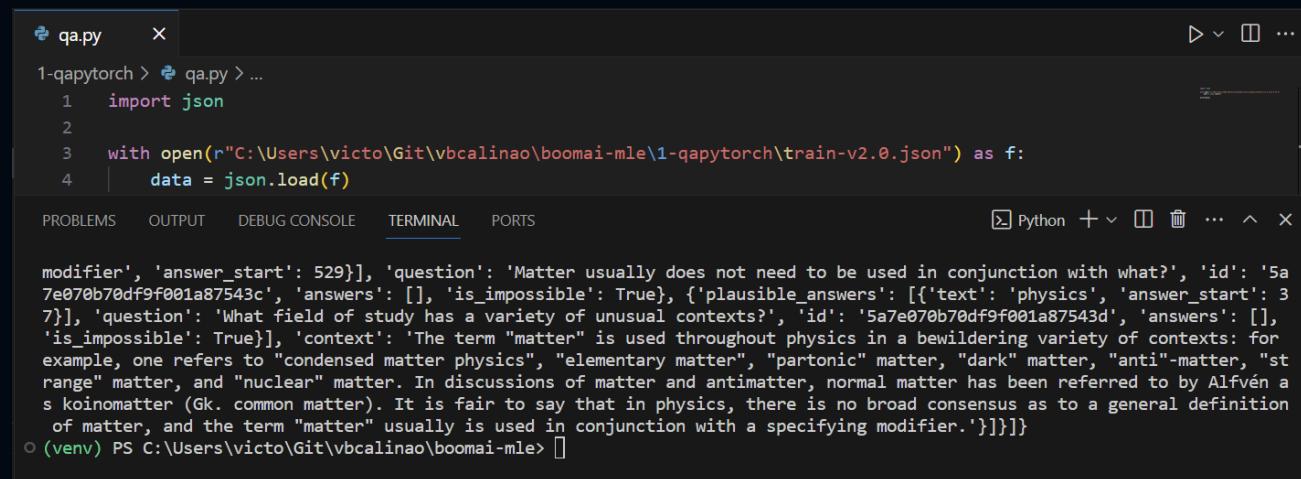
Knowledge-based QA model: encodes a large corpus of domain specific knowledge into the model and generates an answer based on the learned knowledge.

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.



A screenshot of a code editor window titled "qa.py". The code imports json and reads a file "train-v2.0.json" located at "C:\Users\victo\Git\vbcalinao\boomai-mle\1-qapytorch\train-v2.0.json". The editor interface includes tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS, and a bottom bar with Python-related icons.

```
qa.py
1-qapytorch > qa.py > ...
1 import json
2
3 with open(r"C:\Users\victo\Git\vbcalinao\boomai-mle\1-qapytorch\train-v2.0.json") as f:
4     data = json.load(f)

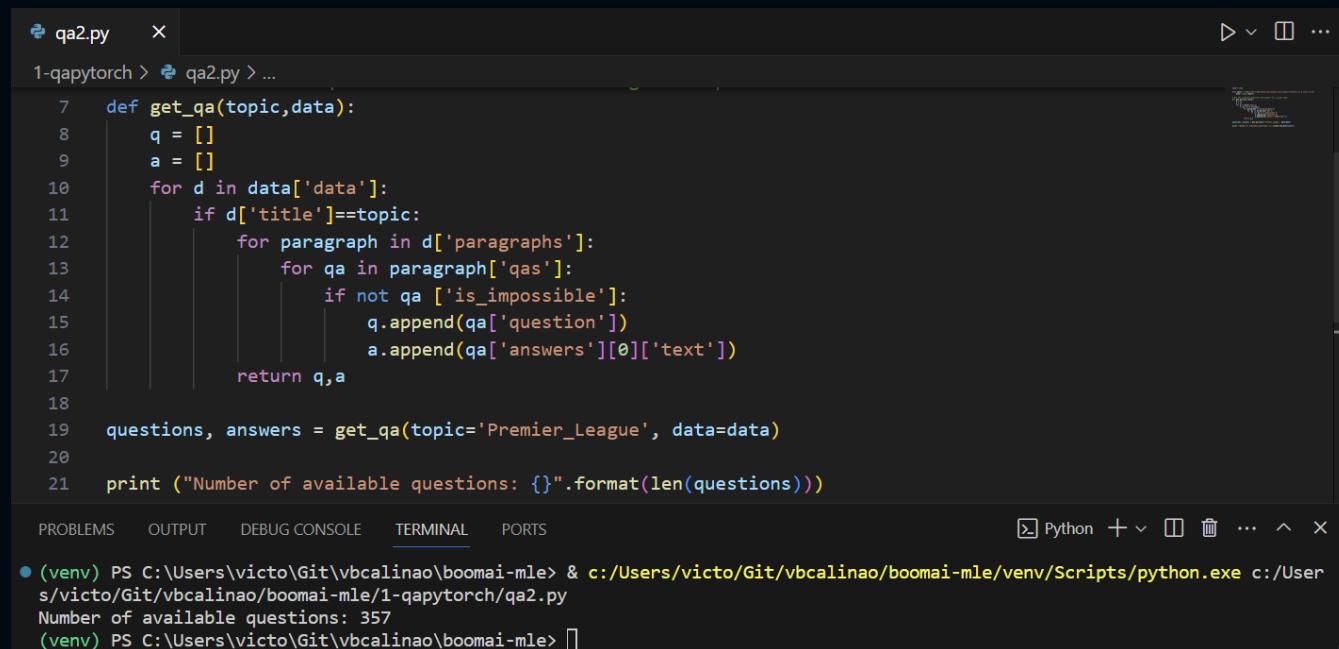
modifier', 'answer_start': 529}], 'question': 'Matter usually does not need to be used in conjunction with what?', 'id': '5a7e070b70df9f001a87543c', 'answers': [], 'is_impossible': True}, {'plausible_answers': [{'text': 'physics', 'answer_start': 37}], 'question': 'What field of study has a variety of unusual contexts?', 'id': '5a7e070b70df9f001a87543d', 'answers': [], 'is_impossible': True}], 'context': 'The term "matter" is used throughout physics in a bewildering variety of contexts: for example, one refers to "condensed matter physics", "elementary matter", "partonic" matter, "dark" matter, "anti"-matter, "strange" matter, and "nuclear" matter. In discussions of matter and antimatter, normal matter has been referred to by Alfvén as koinomatter (Gk. common matter). It is fair to say that in physics, there is no broad consensus as to a general definition of matter, and the term "matter" usually is used in conjunction with a specifying modifier.'}]]}
```

(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle>

# 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.



```
qa2.py
1-qapytorch > qa2.py > ...
7 def get_qa(topic,data):
8     q = []
9     a = []
10    for d in data['data']:
11        if d['title']==topic:
12            for paragraph in d['paragraphs']:
13                for qa in paragraph['qas']:
14                    if not qa ['is_impossible']:
15                        q.append(qa['question'])
16                        a.append(qa['answers'][0]['text'])
17    return q,a
18
19 questions, answers = get_qa(topic='Premier_League', data=data)
20
21 print ("Number of available questions: {}".format(len(questions)))
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + ⌂ ⌄ ⌅ ⌆ ⌇ ⌈ ⌉ ×
● (venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle> & c:/Users/victo/Git/vbcalinao/boomai-mle/venv/Scripts/python.exe c:/Users/victo/Git/vbcalinao/boomai-mle/1-qapytorch/qa2.py
Number of available questions: 357
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle> 
```

# 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 (definitely from MacOS) because it uses 'wget'. As a Windows user, use 'curl' instead.

# Q&A with PyTorch

## PRETRAINED EMBEDDING MODEL

```
$ downloadmodel.sh X
1-qapytorch > $ downloadmodel.sh
1  #!/bin/bash
2
3  # defines the qa model
4  MODEL_DIR="https://huggingface.co/sentence-transformers/paraphrase-MiniLM-L6-v2/resolve/main"
5  MODEL_NAME="paraphrase-MiniLM-L6-v2"
6
7  # downloads the qa model. To make this image more general one can use curl
8  # with the "-O" argument to download the necessary files defined
9  # in "require.txt".
10
11 mkdir ${MODEL_NAME} && \
12     curl -o ${MODEL_NAME}/vocab.txt ${MODEL_DIR}/vocab.txt && \
13     curl -o ${MODEL_NAME}/tokenizer_config.json ${MODEL_DIR}/tokenizer_config.json && \
14     curl -o ${MODEL_NAME}/tokenizer.json ${MODEL_DIR}/tokenizer.json && \
15     curl -o ${MODEL_NAME}/special_tokens_map.json ${MODEL_DIR}/special_tokens_map.json && \
16     curl -o ${MODEL_NAME}/sentence_bert_config.json ${MODEL_DIR}/sentence_bert_config.json && \
17     curl -o ${MODEL_NAME}/pytorch_model.bin ${MODEL_DIR}/pytorch_model.bin && \
18     curl -o ${MODEL_NAME}/modules.json ${MODEL_DIR}/modules.json && \
19     curl -o ${MODEL_NAME}/config_sentence_transformers.json ${MODEL_DIR}/config_sentence_transformers.json && \
20     curl -o ${MODEL_NAME}/config.json ${MODEL_DIR}/config.json
```

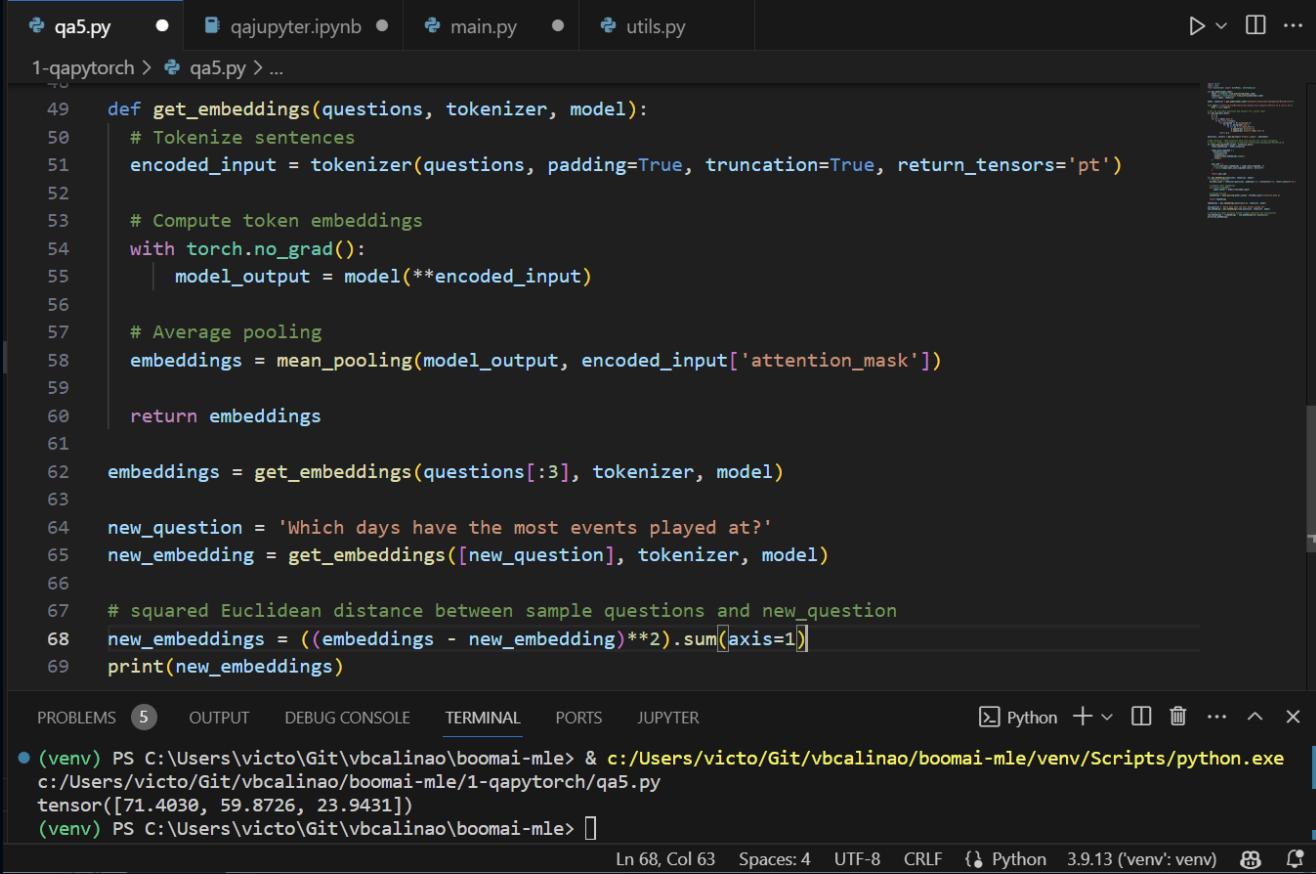
# Q&A with PyTorch

## 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.



A screenshot of a code editor showing a Python script named `qa5.py`. The code defines a function `get_embeddings` that tokenizes sentences, computes their embeddings using a PyTorch model, and performs average pooling. It then calculates the Euclidean distance between the sample embeddings and a new question's embedding. The terminal tab shows the command run and the resulting tensor output.

```
49 def get_embeddings(questions, tokenizer, model):
50     # Tokenize sentences
51     encoded_input = tokenizer(questions, padding=True, truncation=True, return_tensors='pt')
52
53     # Compute token embeddings
54     with torch.no_grad():
55         model_output = model(**encoded_input)
56
57     # Average pooling
58     embeddings = mean_pooling(model_output, encoded_input['attention_mask'])
59
60     return embeddings
61
62 embeddings = get_embeddings(questions[:3], tokenizer, model)
63
64 new_question = 'Which days have the most events played at?'
65 new_embedding = get_embeddings([new_question], tokenizer, model)
66
67 # squared Euclidean distance between sample questions and new_question
68 new_embeddings = ((embeddings - new_embedding)**2).sum(axis=1)
69 print(new_embeddings)
```

PROBLEMS 5 OUTPUT DEBUG CONSOLE TERMINAL PORTS JUPYTER

(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle> & c:/Users/victo/Git/vbcalinao/boomai-mle/venv/Scripts/python.exe  
c:/Users/victo/Git/vbcalinao/boomai-mle/1-qapytorch/qa5.py  
tensor([71.4030, 59.8726, 23.9431])  
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle> []

Ln 68, Col 63 Spaces: 4 UTF-8 CRLF { Python 3.9.13 ('venv': venv) ⌂

# 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.

```
EXPLORER ...  
BOOMAI-MLE  
1-qapytorch  
$ downloadmodel.sh  
{ train-v2.0.json  
app  
main.py  
utils.py  
> demo  
> test  
> venv  
Dockerfile  
main.py • utils.py • Dockerfile •  
app > main.py > ...  
1 import uvicorn  
2 from fastapi import FastAPI, Request  
3 from app.utils import QAEmbedder  
4  
5 app = FastAPI()  
6  
7 @app.post("/set_context")  
8 async def set_context(data:Request):  
9     """  
10        Fastapi POST method that sets the QA context for search.  
11    """  
12  
13    Args:  
14        data(`dict`): Two fields required 'questions' (`list` of `str`)  
15            and 'answers' (`list` of `str`)  
16  
17
```

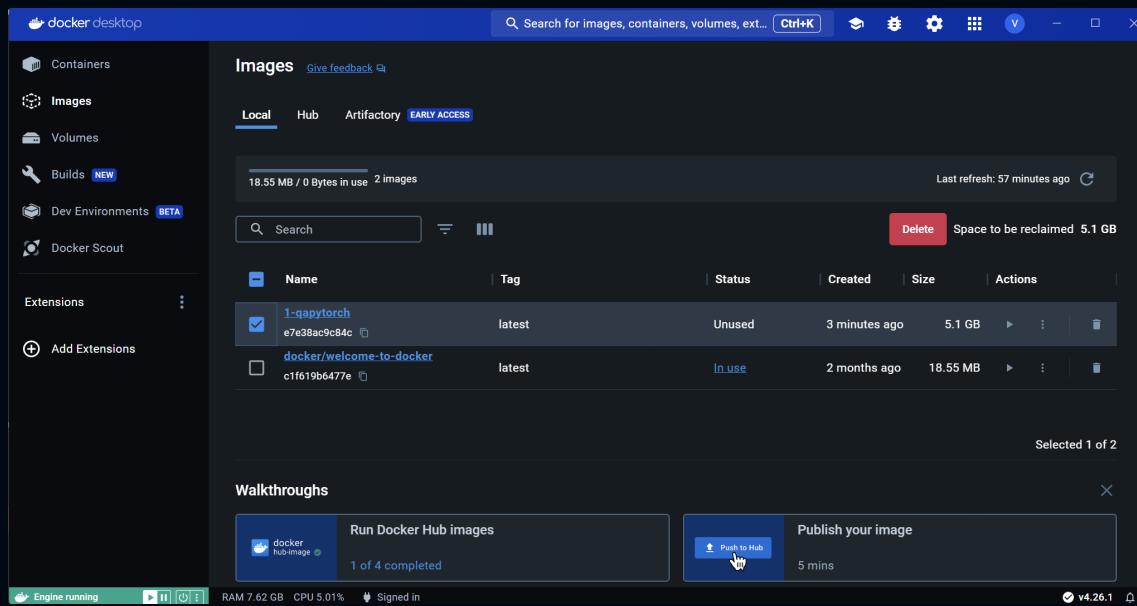
```
main.py • utils.py • Dockerfile •  
app > main.py > ...  
1 import torch  
2 from transformers import AutoTokenizer, AutoModel  
3  
4 class QAEmbedder:  
5     > def __init__(self, model_name="paraphrase-MiniLM-L6-v2"): ...  
18  
19     > def get_model(self, model_name): ...  
31  
32     > def set_model(self, model_name): ...  
42  
43     > def _mean_pooling(self, model_output, attention_mask): ...  
70  
71     > def get_embeddings(self, questions, batch=32): ...  
98  
99  
100    class QASearcher:  
101        > def __init__(self, model_name="paraphrase-MiniLM-L6-v2"): ...  
115  
116        > def set_context_qa(self, questions, answers): ...  
127  
128        > def get_q_embeddings(self, questions): ...  
141  
142        > def cosine_similarity(self, questions, batch=32): ...  
159  
160        > def get_answers(self, questions, batch=32): ...
```

```
main.py • utils.py • Dockerfile •  
Dockerfile > ...  
1 FROM python:3.8.12-slim-buster  
2  
3 # Install wget and required pip libraries  
4 RUN apt-get update &&  
5 apt-get install -y --no-install-recommends wget &&  
6 rm -rf /var/lib/apt/lists/* &&  
7 pip install --no-cache-dir transformers[torch] uvicorn fastapi  
8  
9 # adds the script defining the QA model to docker  
10 COPY download_model.sh .  
11  
12 # Downloads the required QA model  
13 RUN bash download_model.sh  
14  
15 # copies the app files to the docker image  
16 COPY app/ app/  
17  
18 # runs our application at the start of the docker image  
19 CMD ["python", "app/main.py"]
```

# Q&A with PyTorch

## DOCKER

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



```
#9 6.967
Speed
100 229 100 229 0 0 519 0 --:--:-- --:--:-- --:--:-- 520
#9 7.418 % Total % Received % Xferd Average Speed Time Time
Current
#9 7.418
Speed
100 122 100 122 0 0 276 0 --:--:-- --:--:-- --:--:-- 275
#9 7.871 % Total % Received % Xferd Average Speed Time Time
Current
#9 7.871
Speed
100 629 100 629 0 0 1416 0 --:--:-- --:--:-- --:--:-- 1416
#9 DONE 8.4s

#10 [6/6] COPY app/ app/
#10 DONE 0.1s

#11 exporting to image
#11 exporting layers
#11 exporting layers 33.5s done
#11 writing image sha256:e7e38ac9c84ce67ae84190ba443b9e457475d38300f9951010261fe
f19bf3da1 done
#11 naming to docker.io/library/1-qapytorch done
#11 DONE 33.5s

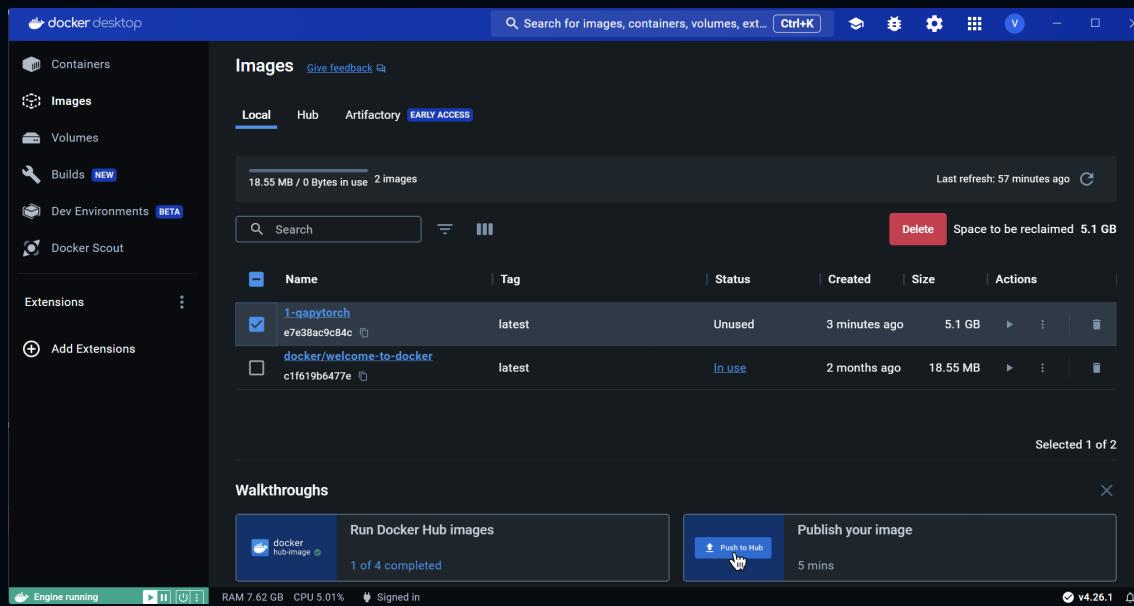
What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview

victo@ZEPHR MINGW64 ~/Git/vbcalinao/boomai-mle/1-qapytorch
$
```

# Q&A with PyTorch

## DOCKER

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



```
#9 6.967
Speed
100 229 100 229 0 0 519 0 --:--:-- --:--:-- --:--:-- 520
#9 7.418 % Total % Received % Xferd Average Speed Time Time
Current
#9 7.418
Speed
100 122 100 122 0 0 276 0 --:--:-- --:--:-- --:--:-- 275
#9 7.871 % Total % Received % Xferd Average Speed Time Time
Current
#9 7.871
Speed
100 629 100 629 0 0 1416 0 --:--:-- --:--:-- --:--:-- 1416
#9 DONE 8.4s

#10 [6/6] COPY app/ app/
#10 DONE 0.1s

#11 exporting to image
#11 exporting layers
#11 exporting layers 33.5s done
#11 writing image sha256:e7e38ac9c84ce67ae84190ba443b9e457475d38300f9951010261fe
f19bf3da1 done
#11 naming to docker.io/library/1-qapytorch done
#11 DONE 33.5s

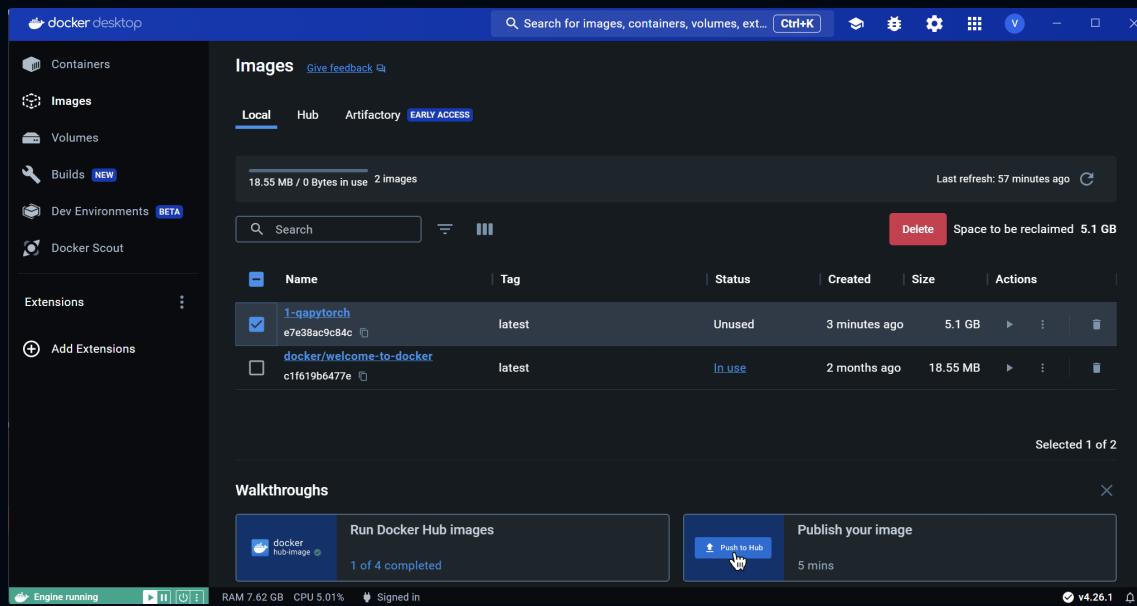
What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview

victo@ZEPHR MINGW64 ~/Git/vbcalinao/boomai-mle/1-qapytorch
$
```

# Q&A with PyTorch

## DOCKER

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



```
#9 6.967
Speed
100 229 100 229 0 0 519 0 --:--:-- --:--:-- --:--:-- 520
#9 7.418 % Total % Received % Xferd Average Speed Time Time
Current
#9 7.418
Speed
100 122 100 122 0 0 276 0 --:--:-- --:--:-- --:--:-- 275
#9 7.871 % Total % Received % Xferd Average Speed Time Time
Current
#9 7.871
Speed
100 629 100 629 0 0 1416 0 --:--:-- --:--:-- --:--:-- 1416
#9 DONE 8.4s

#10 [6/6] COPY app/ app/
#10 DONE 0.1s

#11 exporting to image
#11 exporting layers
#11 exporting layers 33.5s done
#11 writing image sha256:e7e38ac9c84ce67ae84190ba443b9e457475d38300f9951010261fe
f19bf3da1 done
#11 naming to docker.io/library/1-qapytorch done
#11 DONE 33.5s

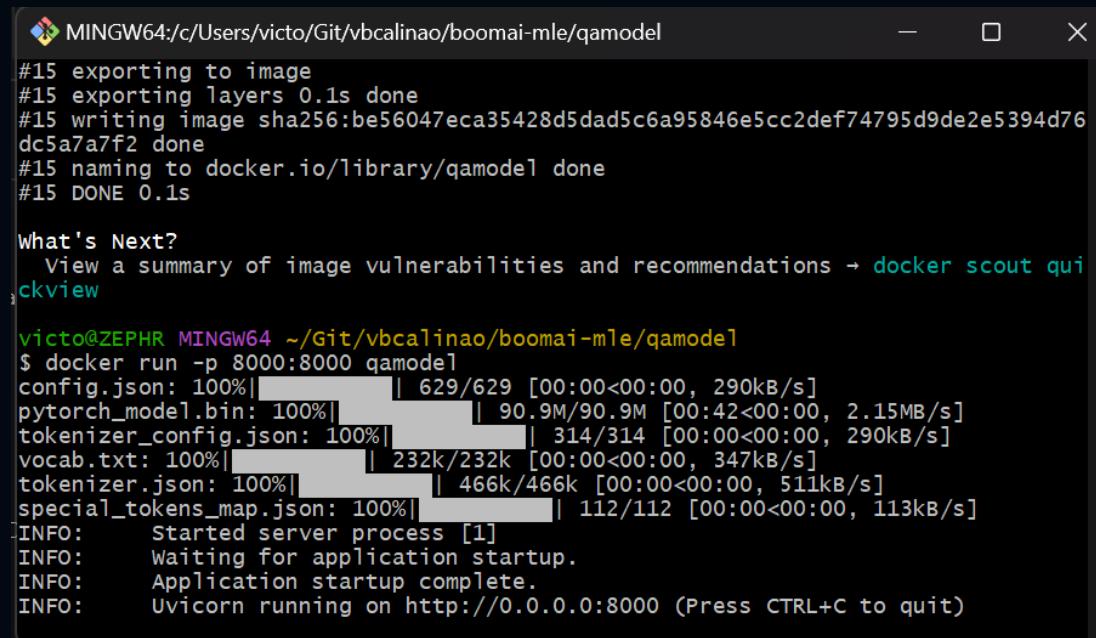
What's Next?
View a summary of image vulnerabilities and recommendations → docker scout quickview

victo@ZEPHR MINGW64 ~/Git/vbcalinao/boomai-mle/1-qapytorch
$
```

# 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.



```
MINGW64:/c/Users/victo/Git/vbcalinao/boomai-mle/qamodel
#15 exporting to image
#15 exporting layers 0.1s done
#15 writing image sha256:be56047eca35428d5dad5c6a95846e5cc2def74795d9de2e5394d76
dc5a7a7f2 done
#15 naming to docker.io/library/qamodel done
#15 DONE 0.1s

What's Next?
  View a summary of image vulnerabilities and recommendations → docker scout quickview

victo@ZEPHR MINGW64 ~/Git/vbcalinao/boomai-mle/qamodel
$ docker run -p 8000:8000 qamodel
config.json: 100%|██████████| 629/629 [00:00<00:00, 290kB/s]
pytorch_model.bin: 100%|██████████| 90.9M/90.9M [00:42<00:00, 2.15MB/s]
tokenizer_config.json: 100%|██████████| 314/314 [00:00<00:00, 290kB/s]
vocab.txt: 100%|██████████| 232k/232k [00:00<00:00, 347kB/s]
tokenizer.json: 100%|██████████| 466k/466k [00:00<00:00, 511kB/s]
special_tokens_map.json: 100%|██████████| 112/112 [00:00<00:00, 113kB/s]
INFO:     Started server process [1]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
```

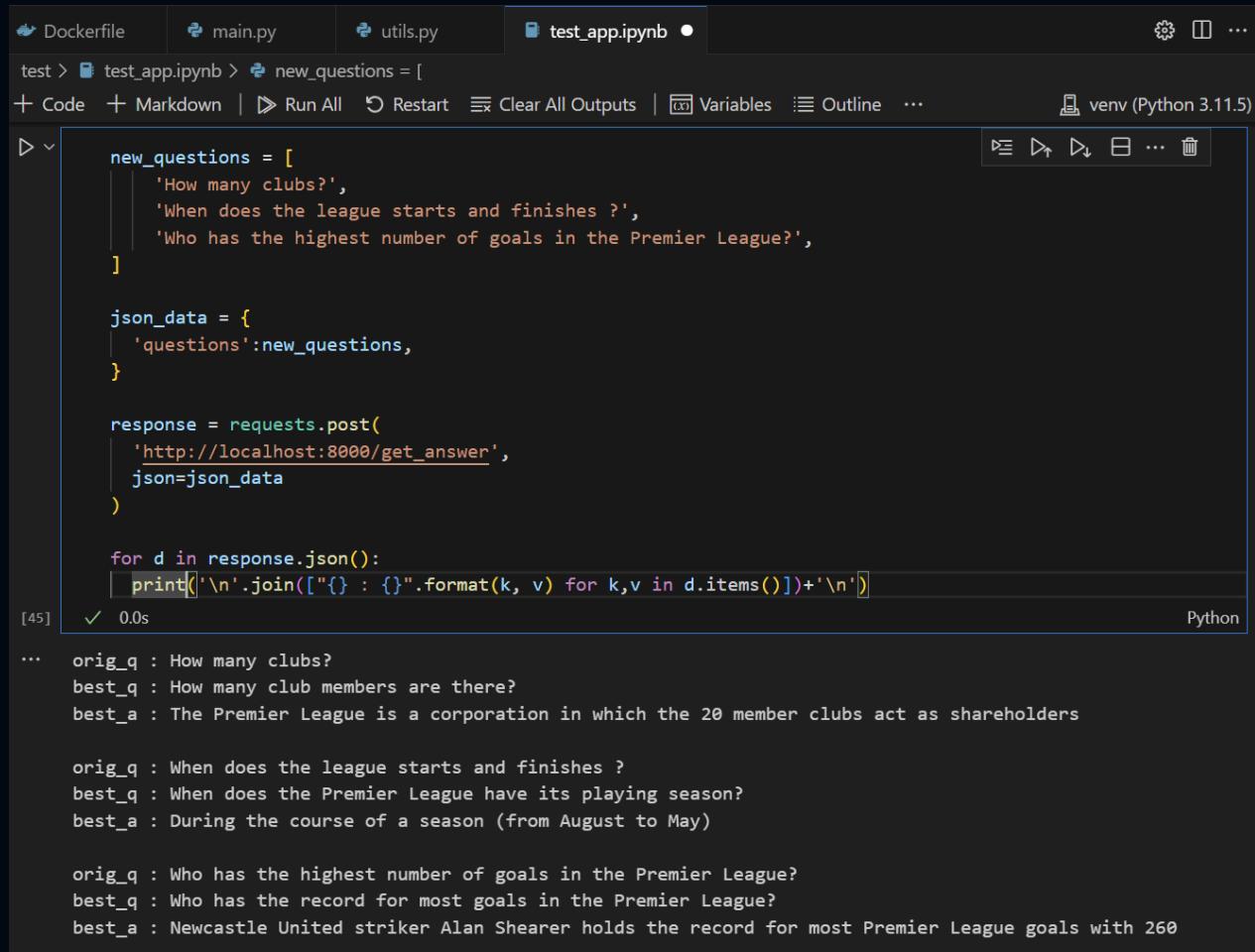
# Q&A with PyTorch

## 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.



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** Dockerfile, main.py, utils.py, test\_app.ipynb (selected), venv (Python 3.11.5).
- Toolbar:** Code, Markdown, Run All, Restart, Clear All Outputs, Variables, Outline, ...
- Code Cell:** Contains Python code for sending a POST request to a local server to get answers for new questions. The code uses the requests library.
- Output Cell:** Shows the results of the execution, displaying the questions and their corresponding answers returned by the server.
- Cell Number:** [45]
- Execution Time:** 0.0s
- Language:** Python

```
new_questions = [
    'How many clubs?',
    'When does the league starts and finishes ?',
    'Who has the highest number of goals in the Premier League?',
]

json_data = {
    'questions':new_questions,
}

response = requests.post(
    'http://localhost:8000/get_answer',
    json=json_data
)

for d in response.json():
    print("\n".join(["{} : {}".format(k, v) for k,v in d.items()])+'\n')

... orig_q : How many clubs?
best_q : How many club members are there?
best_a : The Premier League is a corporation in which the 20 member clubs act as shareholders

orig_q : When does the league starts and finishes ?
best_q : When does the Premier League have its playing season?
best_a : During the course of a season (from August to May)

orig_q : Who has the highest number of goals in the Premier League?
best_q : Who has the record for most goals in the Premier League?
best_a : Newcastle United striker Alan Shearer holds the record for most Premier League goals with 260
```

The background of the slide features a dark purple gradient with a subtle texture. It consists of several sets of thin, light-colored lines that curve and overlap, creating a sense of depth and motion. Scattered throughout these lines are numerous small, glowing particles of varying sizes, which appear to be moving along the paths of the lines.

/ 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](#)

```
1 import assemblyai as aai
2
3 transcriber = aai.Transcriber()
4 transcript = transcriber.transcribe(URL, config)
5
6 print(transcript.text)
```

# Speech-to-text

GETTING STARTED IN SECONDS (literally)

Welcome to AssemblyAI

Quickly try our AI models for Speech-to-Text and Speech Understanding in just a few lines of code.

Python JavaScript Ruby C# PHP

```
1 # `pip3 install assemblyai` (macOS)
2 # `pip install
3
4 import assembl
5
6 a.settings.a
7 transcriber =
8
9 transcript = t
10 # transcript =
11
12 print(transcript.text)
```

Congrats on your first transcription!

We can't wait to see what you build.

Continue >

Skip and go to my dashboard >

```
main.py Activate.ps1
```

```
app > main.py > ...
2 # `pip install assemblyai` (Windows)
3
4 import assemblyai as aai
5
6 aai.settings.api_key = "de93505212d1443595a4537ba2210237"
7 transcriber = aai.Transcriber()
8
9 transcript = transcriber.transcribe("https://storage.googleapis.com/aai-web-samples/news.mp4")
10 # transcript = transcriber.transcribe("./my-local-audio-file.wav")
11
12 print(transcript.text)
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle\speech-to-text> & c:/Users/victo/Git/vbcalinao\boomai-mle\speech-to-text\venv\Scripts\python.exe c:/Users/victo/Git/vbcalinao\boomai-mle\speech-to-text/app/main.py
I'm David Curley at the Smithsonian Aaron Space Museum, where we are marking 50 years since man landed and walked on t
he moon in a lander just like this one. We are going to show you some of the actual ABC News coverage from 50 years ag
o during that eight day mission of this remarkable achievement, Apollo eleven's lander, the Eagle, would be the first
manned craft to land on the moon. For training, NASA came with an unusual contraption. Neil Armstrong actually had to
eject from it once, and then he had a couple of successful flights. ABC News anchor at the time, 50 years ago, Frank R
eynolds with a look at that unusual trainer. Apollo eleven commander Neil Armstrong is at the controls of a lunar land
ing training vehicle, testing the reaction control jets. These thrusters stabilize the LEM during landing and takeoff.
The LLTV is designed to simulate the behavior of the LEM as it lands in the moon's gravity. Lunar gravity is one 6th
that of the Earth's. Neil Armstrong flew one of these vehicles on May 6, 1968, and that flight was nearly his last. La
ter reports by a NASA investigating team said the crash, which we'll see soon, was caused by a loss of fuel pressure c
ompounded by a warning light that failed to work. Armstrong's coolness under pressure saved himself and possibly month
same year, 1968, another test pilot, Joseph Allegrandi, also escaped from an LLTV just before it crashed. The training
vehicle then underwent several design modifications and improvements. Engineers had to increase the vehicle's rocket
power to help stabilize the craft. That made the LLTV less of a moon gravity simulator, but it improved pilot safety.
On June 16, 1969, Neil Armstrong flew the vehicle, sometimes called the flying bedstead, to several perfect landings.
Question was, how does the machine fly? And the answer is that we're very pleased with the way it flies. It's a signifi
cant improvement over the LLRV, which we were flying here a year ago, and I think it does an excellent job of actuall
y capturing the handling characteristics of the lunar module in a landing maneuver. It's really a great deal different
than any other kind of aircraft that I've ever flown. The, the simulation of lunar gravity has some aspects that make
this type of flight sufficiently different from anything else we've ever done to make this vehicle very worthwhile. A
nd I'm very pleased that I had the opportunity to get some flights in it here just before the Apollo eleven flight.
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle\speech-to-text> []
```

Ln 12, Col 23 Spaces: 4 UTF-8 CRLF (Python 3.11.5 ('venv': venv) 8 8

# 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

## Choose a plan

Join over 10,000 developers building with the AssemblyAI API

### Your current plan

Explore the API for free.

- Process 2 files simultaneously.
- 5 async transcriptions per month.
- 100MB file upload limit.
- Community support over Discord and chat.

[Continue with current plan →](#)

### Full API Access RECOMMENDED FOR YOU

Unlock higher limits, full API access, and dedicated support.

- ✓ **Unlimited transcriptions per month.** Transcribe as many files as you need, with no restrictions.
- ✓ **Real-Time Transcription.** High accuracy and low latency with real-time transcriptions.
- ✓ **5GB of file upload limit.** Easily handle larger audio and video files.
- ✓ **Concurrency of up to 32 files.** Enjoy higher concurrency limits with faster speeds. If you have higher volume, contact us after upgrading.
- ✓ **LeMUR.** The easiest way to build LLM apps on voice data.
- ✓ **Advanced reporting.** Monitor usage and spend with detailed daily activity reports.

[Continue →](#)

[See full pricing information](#)

# Speech-to-text

## WORKAROUND

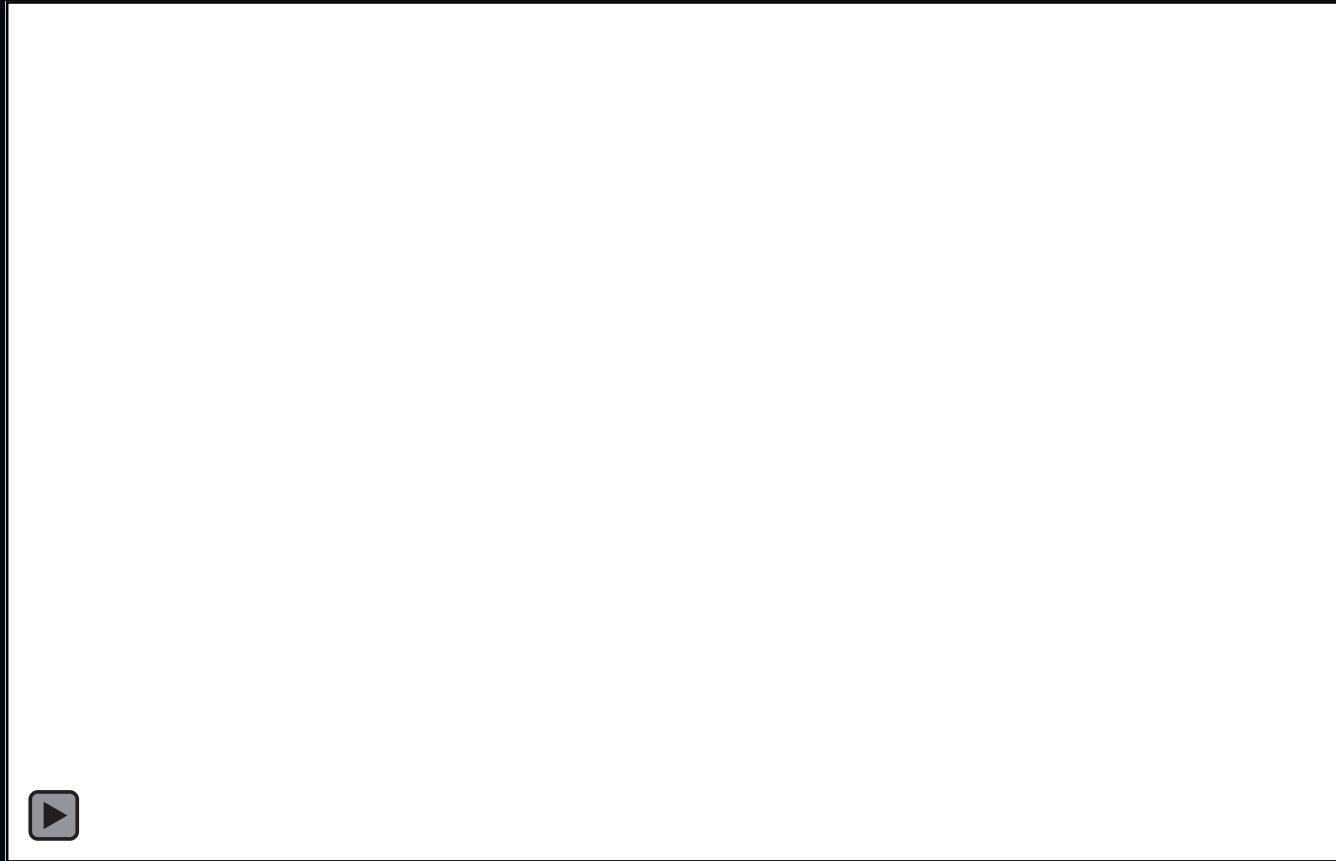
As a workaround, we'll use the local device microphone as the audio source, then it will listen for 5 seconds, will save the recording in a wav file and simultaneously transcript the file afterwards.

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

```
main.py  X
app > main.py > ...
1 import assemblyai
2 import speech_recognition as sr
3 from pydub import AudioSegment
4
5 # Create a recognizer instance
6 r = sr.Recognizer()
7
8 # Use the default microphone as the audio source
9 with sr.Microphone() as source:
10     print("Listening...")
11     audio = r.record(source, duration=10) # Listen for the first phrase and extract it into audio
12
13 # Save the audio to a .wav file
14 with open("recording.wav", "wb") as file:
15     file.write(audio.get_wav_data())
16
17 # Transcribe the audio file using AssemblyAI
18 aai = assemblyai.Client('de93505212d1443595a4537ba2210237')
19 file_transcript = aai.transcribe(filename='recording.wav')
20
21 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.

The screenshot shows a code editor interface with several tabs at the top: 'mictest.py U', 'main.py M', 'recording.wav M', and 'getstarted.py'. The main area displays the following Python code:

```
app > mictest.py U main.py M recording.wav M getstarted.py
22 |
23 # Transcribe the audio file using AssemblyAI
24 FILE_URL = 'recording.wav'
25
26 transcriber = aai.Transcriber()
27
28 print("Transcription is processing...")
29 transcript = transcriber.transcribe(FILE_URL)
30
31 print("Transcript: ", transcript.text)
32
33 # Split the transcript into words
34 words = transcript.text.split()
35
36 # Define a list of vowels
37 vowels = ['a', 'e', 'i', 'o', 'u']
38
39 # Append "-v" to words that end with a vowel
40 modified_words = [word + '-v' if word[-1] in vowels else word for word in words]
41
42 # Join the modified words back into a transcript
43 modified_transcript = ' '.join(modified_words)
44
45 print("Modified transcript (-v): ", modified_transcript)
```

Below the code editor is a terminal window showing the execution of the script:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ai-mle/speech-to-text/venv/Scripts/python.exe c:/Users/victo/Git/vbcalinao/boomai-mle/speech-to-text/app/main.py
Listening...
Recording stopped.
Transcription is processing...
Transcript: The big brown fox jumps over the lazy dog.
Modified transcript: The-v big brown fox jumps over the-v lazy dog.
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle\speech-to-text>
```

The terminal also shows a sidebar with multiple Python environments listed.

# 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.

The screenshot shows a code editor interface with several tabs at the top: 'mictest.py U', 'main.py M', 'recording.wav M', and 'getstarted.py'. The main editor area contains the following Python code:

```
app > mictest.py U
22 |
23 # Transcribe the audio file using AssemblyAI
24 FILE_URL = 'recording.wav'
25
26 transcriber = aai.Transcriber()
27
28 print("Transcription is processing...")
29 transcript = transcriber.transcribe(FILE_URL)
30
31 print("Transcript: ", transcript.text)
32
33 # Split the transcript into words
34 words = transcript.text.split()
35
36 # Define a list of vowels
37 vowels = ['a', 'e', 'i', 'o', 'u']
38
39 # Append "-v" to words that end with a vowel
40 modified_words = [word + '-v' if word[-1] in vowels else word for word in words]
41
42 # Join the modified words back into a transcript
43 modified_transcript = ' '.join(modified_words)
44
45 print("Modified transcript (-v): ", modified_transcript)
```

Below the code editor is a terminal window showing the execution of the script:

```
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
ai-mle/speech-to-text/venv/Scripts/python.exe c:/Users/victo/Git/vbcalinao/boomai-mle/speech-to-text/app/main.py
Listening...
Recording stopped.
Transcription is processing...
Transcript: The big brown fox jumps over the lazy dog.
Modified transcript: The-v big brown fox jumps over the-v lazy dog.
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle\speech-to-text>
```

The terminal also displays the current environment variables and the Python version being used.

# 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

The screenshot shows a code editor interface with several tabs at the top: 'mictest.py U', 'main.py M X', 'recording.wav M', 'getstarted.py', and others. The main area displays Python code for transcribing an audio file using AssemblyAI. The code includes logic to split the transcript into words, define vowels, and append '-v' or '-c' to words based on their ending. The code editor has a dark theme with syntax highlighting for Python. Below the code editor is a terminal window showing the command to run the script, the process of listening and recording, and the resulting modified transcript.

```
mictest.py U main.py M X recording.wav M getstarted.py
app > main.py > ...
22
23 # Transcribe the audio file using AssemblyAI
24 FILE_URL = 'recording.wav'
25
26 transcriber = aai.Transcriber()
27
28 print("Transcription is processing...")
29 transcript = transcriber.transcribe(FILE_URL)
30
31 print("Transcript: ", transcript.text)
32
33 # Split the transcript into words
34 words = transcript.text.split()
35
36 # Define a list of vowels
37 vowels = ['a', 'e', 'i', 'o', 'u']
38
39 # Append "-v" to words that end with a vowel and "-c" to words that end with a consonant
40 modified_words = [word + '-v' if word[-1] in vowels else word + '-c' for word in words]
41
42 # Join the modified words back into a transcript
43 modified_transcript = ' '.join(modified_words)
44

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ v ... ^ x
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle\speech-to-text> & c:/Users/victo/Git/vbcalinao/boomai-mle/speech-to-text/venv/Scripts/python.exe c:/Users/victo/Git/vbcalinao/boomai-mle/speech-to-text/app/main.py
Listening...
Recording stopped.
Transcription is processing...
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
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle\speech-to-text>
```

Python  
Python  
Python  
Python

# 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

The screenshot shows a code editor interface with several tabs at the top: 'mictest.py U', 'main.py M X', 'recording.wav M', 'getstarted.py', and others. The main area displays Python code for transcribing an audio file using AssemblyAI. The code includes logic to split the transcript into words, define vowels, and append '-v' or '-c' to words based on their ending. The code editor has a dark theme with syntax highlighting for Python. Below the code editor is a terminal window showing the command to run the script, the process of listening and recording, and the resulting modified transcript.

```
mictest.py U main.py M X recording.wav M getstarted.py
app > main.py > ...
22
23 # Transcribe the audio file using AssemblyAI
24 FILE_URL = 'recording.wav'
25
26 transcriber = aai.Transcriber()
27
28 print("Transcription is processing...")
29 transcript = transcriber.transcribe(FILE_URL)
30
31 print("Transcript: ", transcript.text)
32
33 # Split the transcript into words
34 words = transcript.text.split()
35
36 # Define a list of vowels
37 vowels = ['a', 'e', 'i', 'o', 'u']
38
39 # Append "-v" to words that end with a vowel and "-c" to words that end with a consonant
40 modified_words = [word + '-v' if word[-1] in vowels else word + '-c' for word in words]
41
42 # Join the modified words back into a transcript
43 modified_transcript = ' '.join(modified_words)
44

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
+ v ... ^ x
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle\speech-to-text> & c:/Users/victo/Git/vbcalinao/boomai-mle/speech-to-text/venv/Scripts/python.exe c:/Users/victo/Git/vbcalinao/boomai-mle/speech-to-text/app/main.py
Listening...
Recording stopped.
Transcription is processing...
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
(venv) PS C:\Users\victo\Git\vbcalinao\boomai-mle\speech-to-text>
```

Python  
Python  
Python  
Python

# / Task 3: BART Text Summarization

# Text Summarization

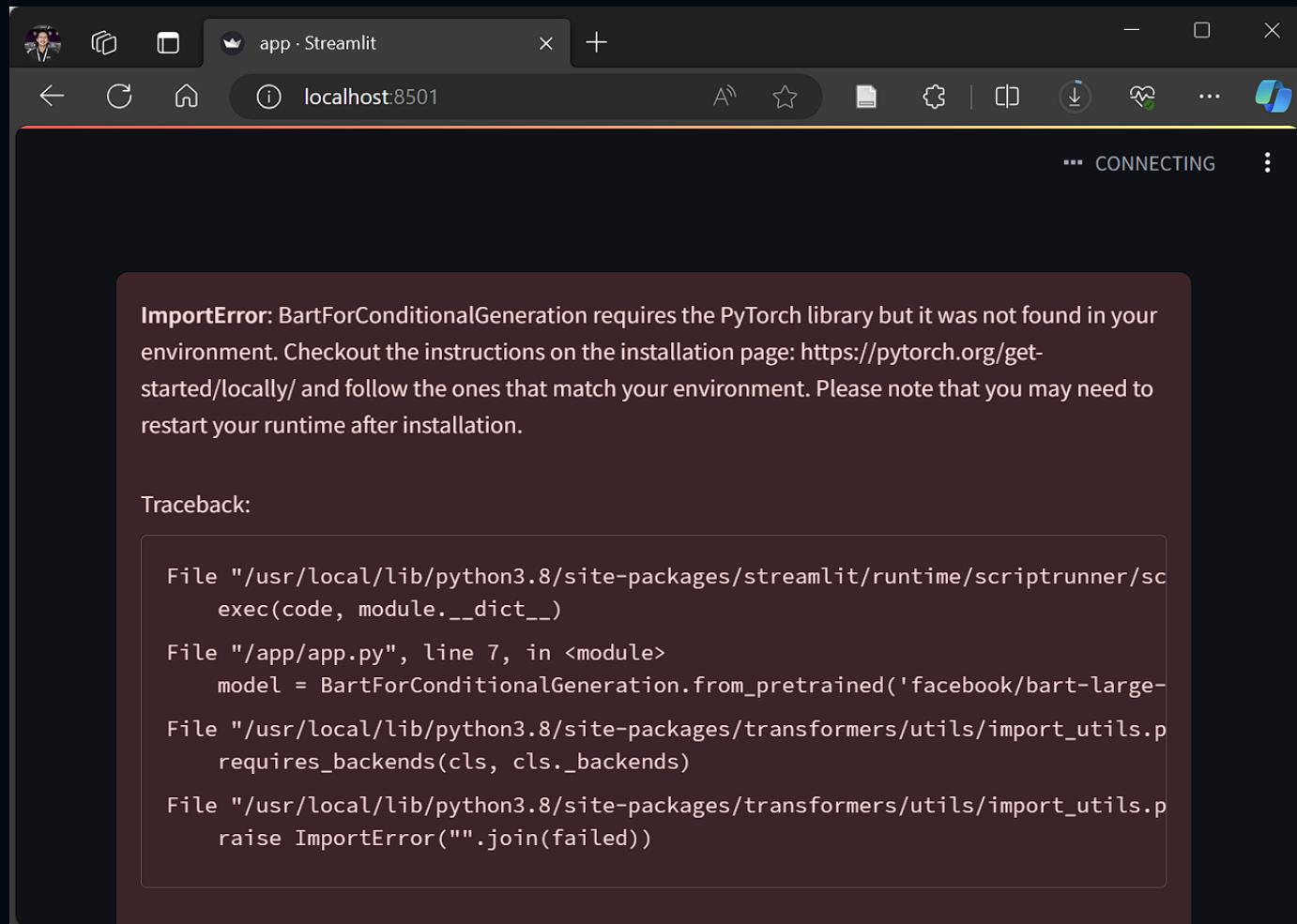
## INSTRUCTIONS

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.