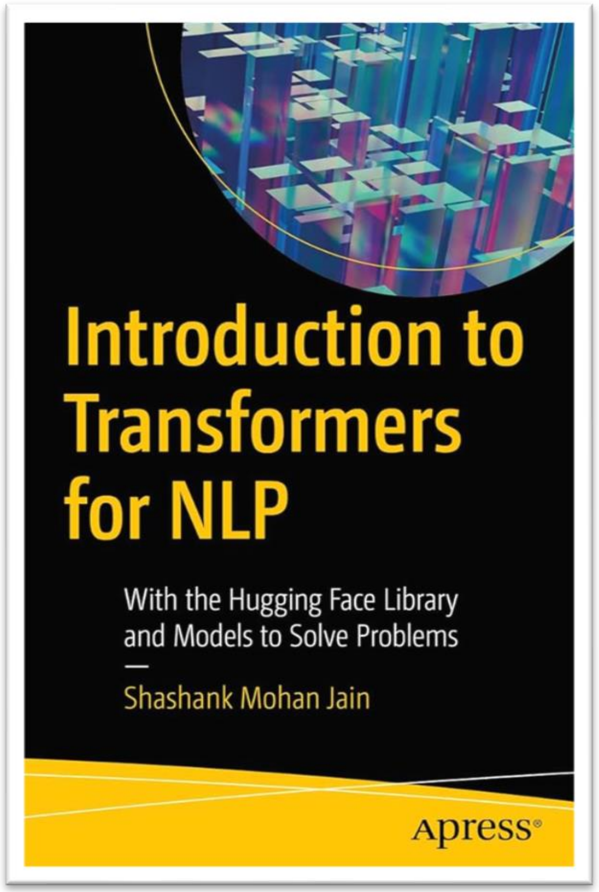
Examples and Applications

With Hugging Face Library


Introduction to Transformers for NLP

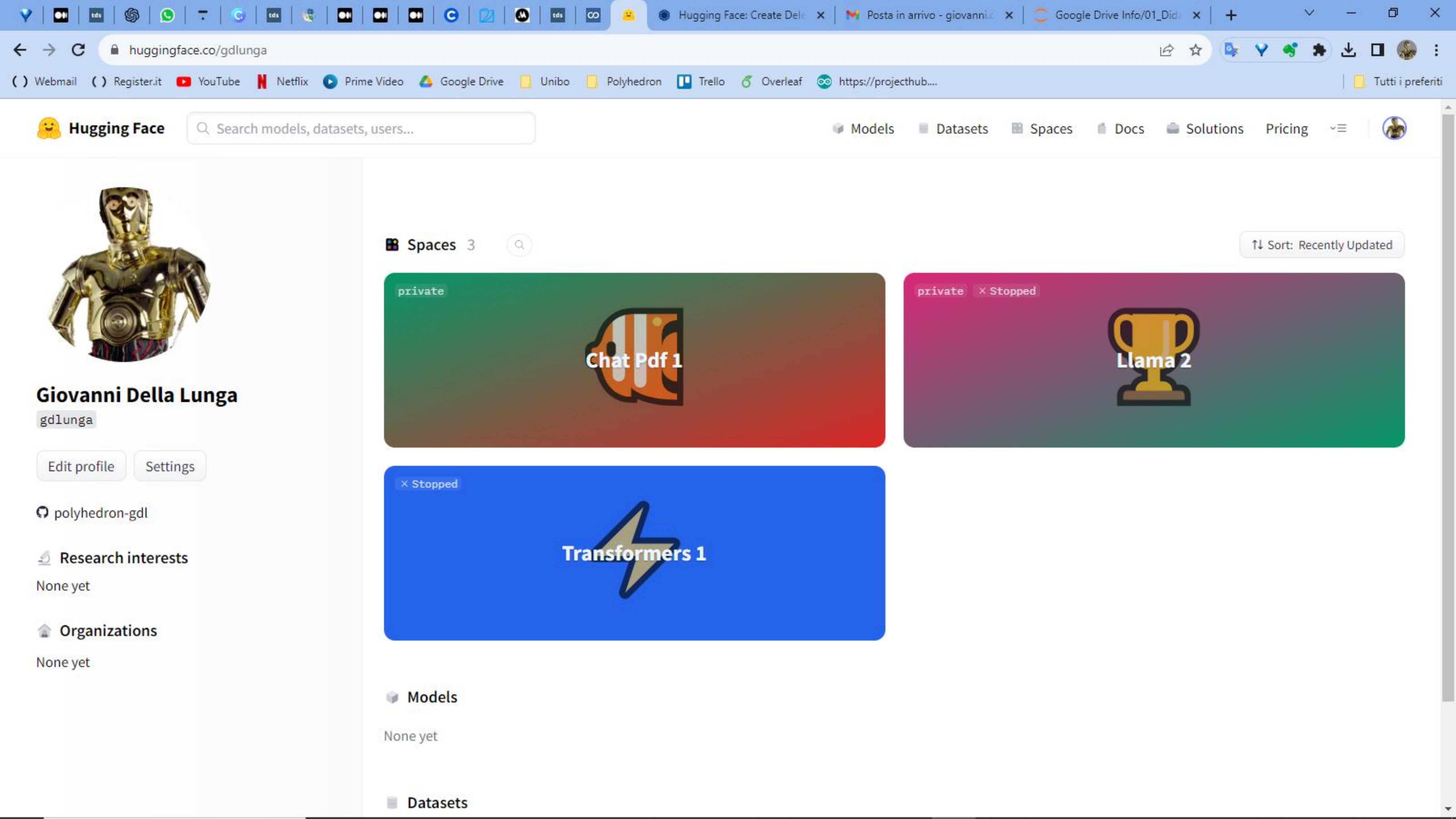
With the Hugging Face Library and Models to Solve Problems

Shashank Mohan Jain

Apress

Using Transformers With Hugging Face Library

Reference: Shashank Mohan Jain «Introduction to Transformers for NLP» APress



Giovanni Della Lunga
gdlunga

polyhedron-gdl

Research interests

None yet

Organizations

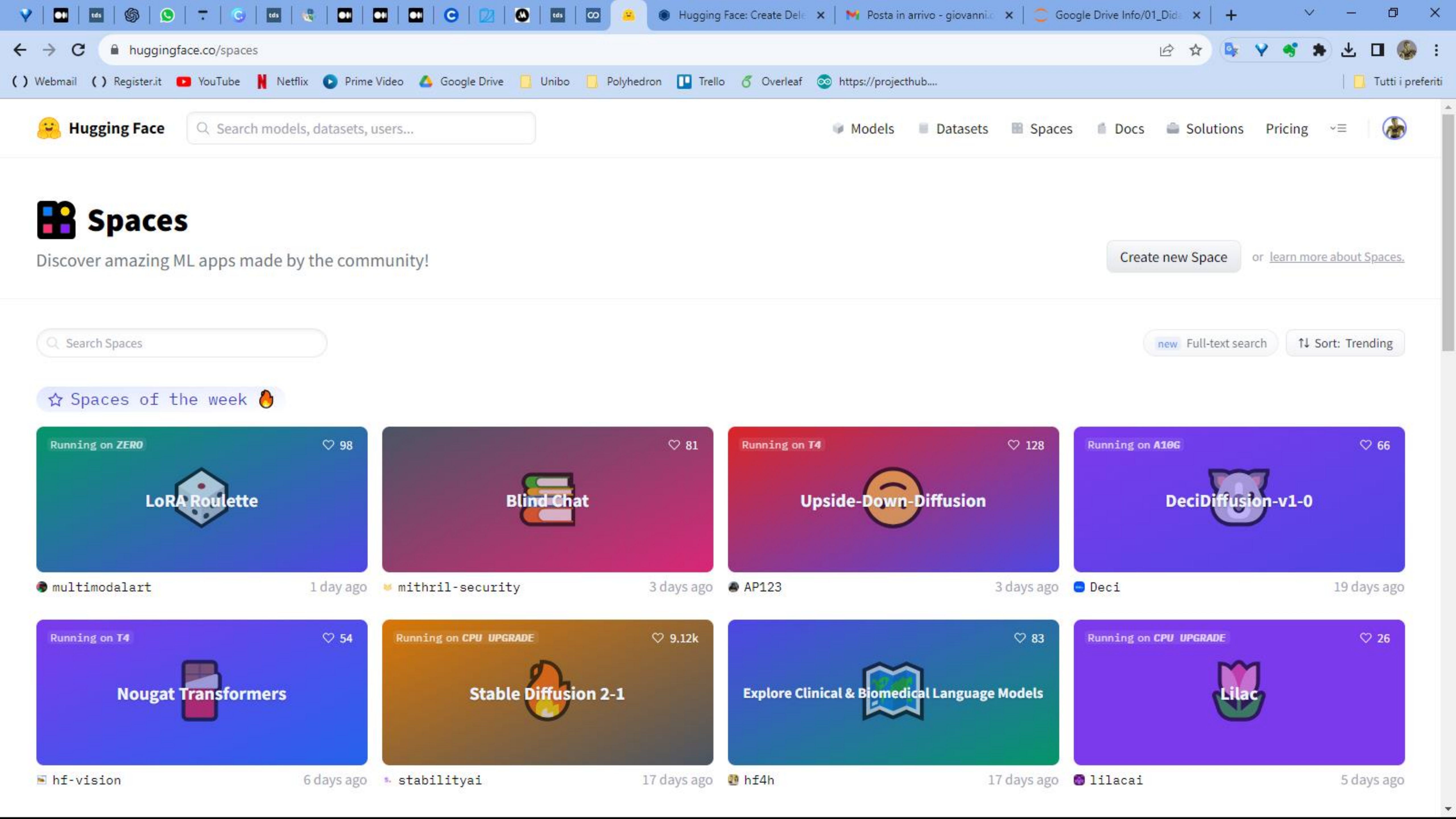
None yet

Spaces 3

Models

None yet

Datasets


huggingface.co/spaces

Hugging Face | Models | Datasets | Spaces | Docs | Solutions | Pricing

Spaces

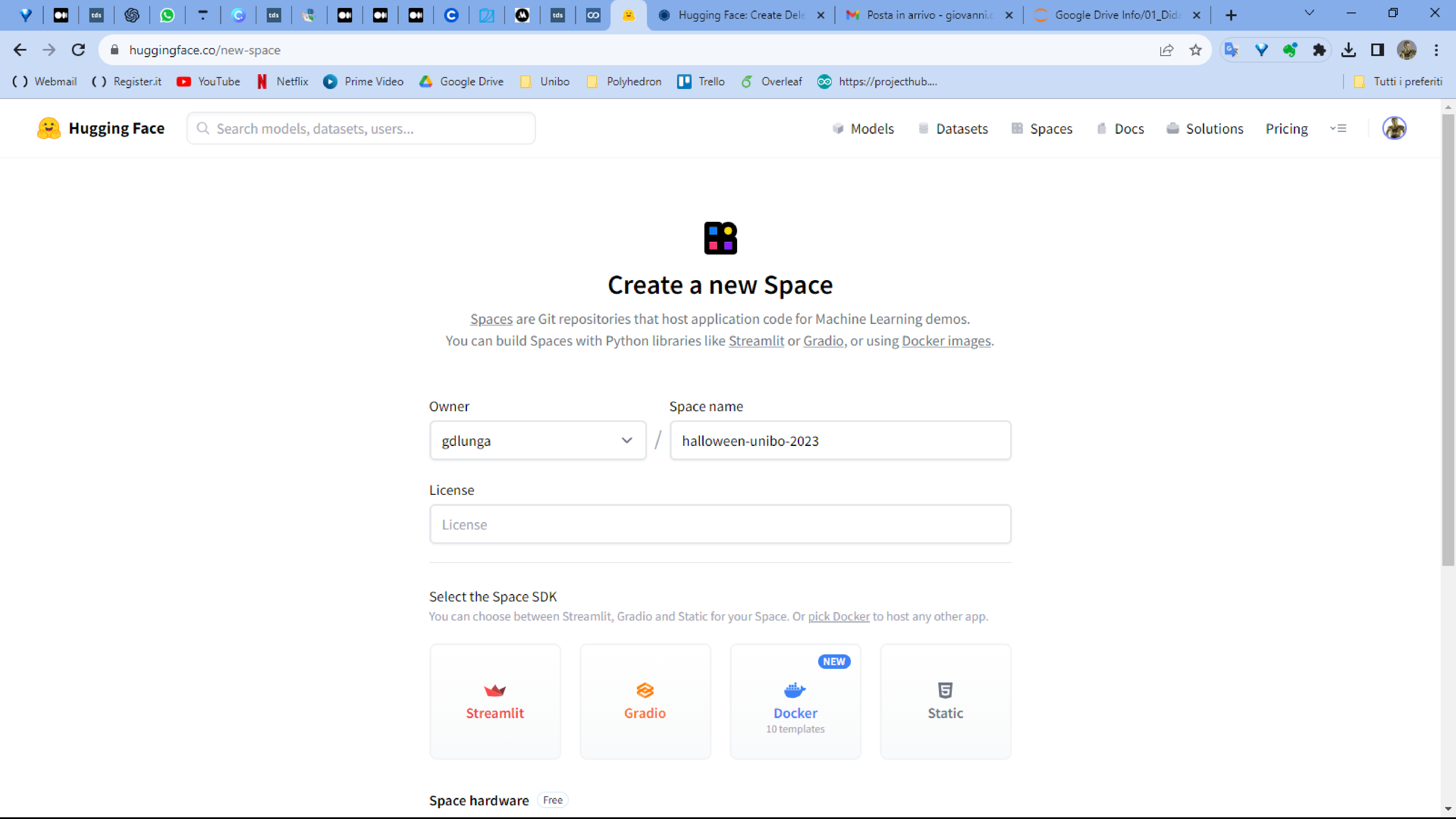
Discover amazing ML apps made by the community!

Create new Space or learn more about Spaces.

Search Spaces | new Full-text search | Sort: Trending

Spaces of the week 🔥

Space	Hardware	Likes	Owner	Updated
LoRA Roulette	Running on ZERO	98	multimodalart	1 day ago
Blind Chat		81	mithril-security	3 days ago
Upside-Down-Diffusion	Running on T4	128	AP123	3 days ago
DeciDiffusion-v1-0	Running on A10G	66	Deci	19 days ago
Nougat Transformers	Running on T4	54	hf-vision	6 days ago
Stable Diffusion 2-1	Running on CPU UPGRADE	9.12k	stabilityai	17 days ago
Explore Clinical & Biomedical Language Models		83	hf4h	17 days ago
Lilac	Running on CPU UPGRADE	26	lilacai	5 days ago



Create a new Space

Spaces are Git repositories that host application code for Machine Learning demos.
You can build Spaces with Python libraries like Streamlit or Gradio, or using Docker images.

Owner	Space name
gdlunga	halloween-unibo-2023

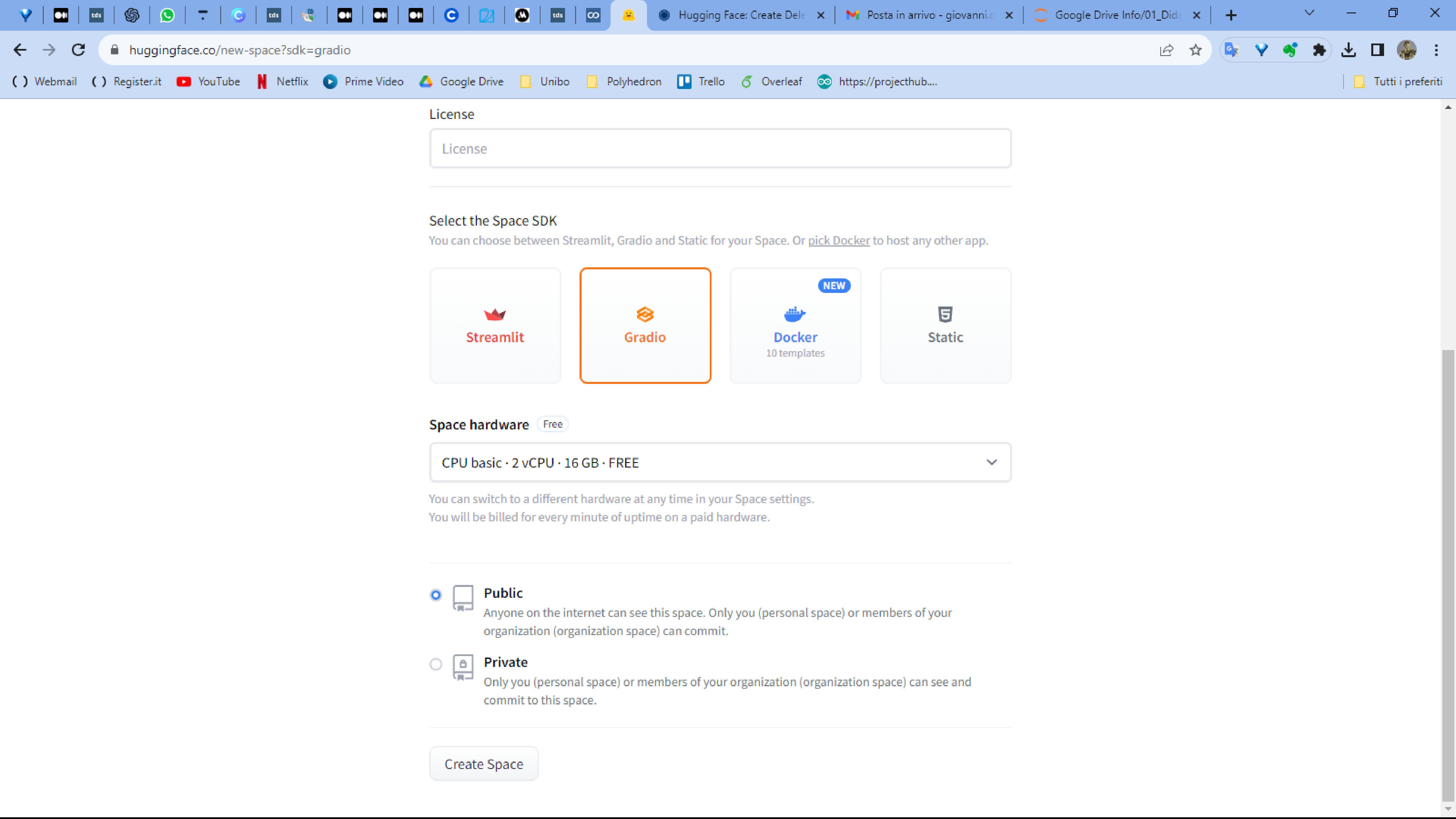
License

License

Select the Space SDK

You can choose between Streamlit, Gradio and Static for your Space. Or pick Docker to host any other app.

Space hardware Free



License

License

Select the Space SDK

You can choose between Streamlit, Gradio and Static for your Space. Or pick Docker to host any other app.

Streamlit
[✓ Gradio]
NEW Docker 10 templates
Static

Space hardware Free

CPU basic · 2 vCPU · 16 GB · FREE

You can switch to a different hardware at any time in your Space settings.
You will be billed for every minute of uptime on a paid hardware.

[✓ **Public**]
Anyone on the internet can see this space. Only you (personal space) or members of your organization (organization space) can commit.

Private
Only you (personal space) or members of your organization (organization space) can see and commit to this space.

Create Space

Gradio: An Introduction

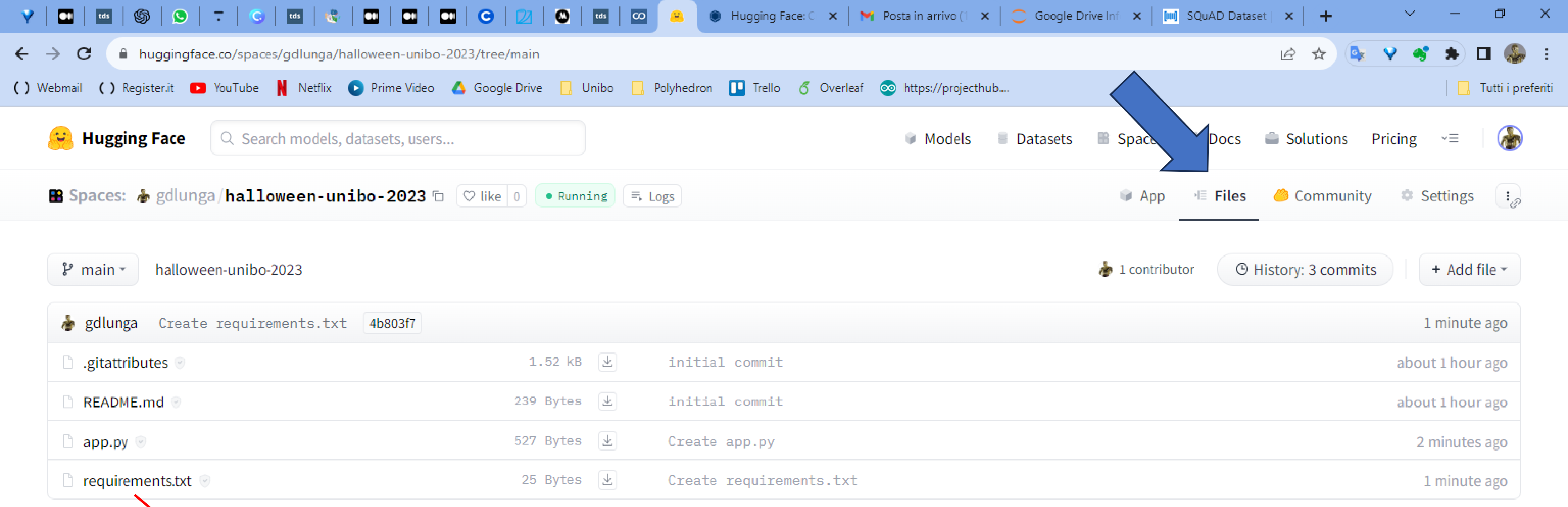
- Gradio is a web framework specially built for deploying and inferencing machine learning models;
- Gradio allows us to have our ML models exposed quickly over a web interface without a requirement of learning too much coding;

Hugging Face Tasks – Question and Answering

- The input to the model would be a paragraph and a quesion from within that paragraph;
- The output of the model inference would be the answer to the question;
- The model we are used are trained on the SQuAD dataset;

Hugging Face Tasks – Question and Answering

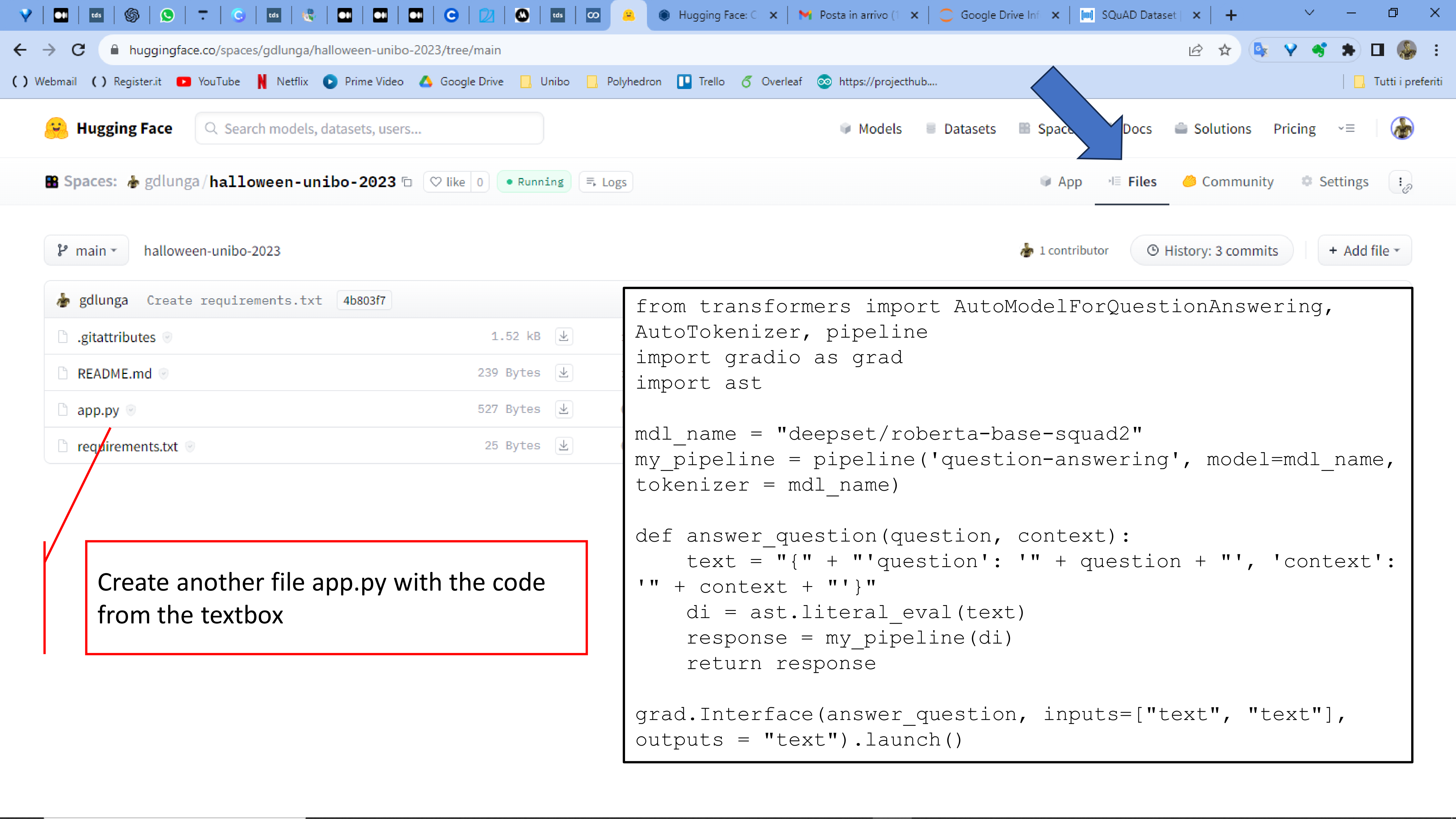
- The Stanford Question Answering Dataset (SQuAD) is a collection of question-answer pairs derived from Wikipedia articles.
- In SQuAD, the correct answers of questions can be any sequence of tokens in the given text.
- Because the questions and answers are produced by humans through crowdsourcing, it is more diverse than some other question-answering datasets.
- SQuAD 1.1 contains 107,785 question-answer pairs on 536 articles.
- SQuAD2.0 (open-domain SQuAD, SQuAD-Open), the latest version, combines the 100,000 questions in SQuAD1.1 with over 50,000 un-answerable questions written adversarially by crowdworkers in forms that are similar to the answerable ones.



gdlunga	Create requirements.txt 4b803f7		1 minute ago
.gitattributes	1.52 kB	initial commit	about 1 hour ago
README.md	239 Bytes	initial commit	about 1 hour ago
app.py	527 Bytes	Create app.py	2 minutes ago
requirements.txt	25 Bytes	Create requirements.txt	1 minute ago

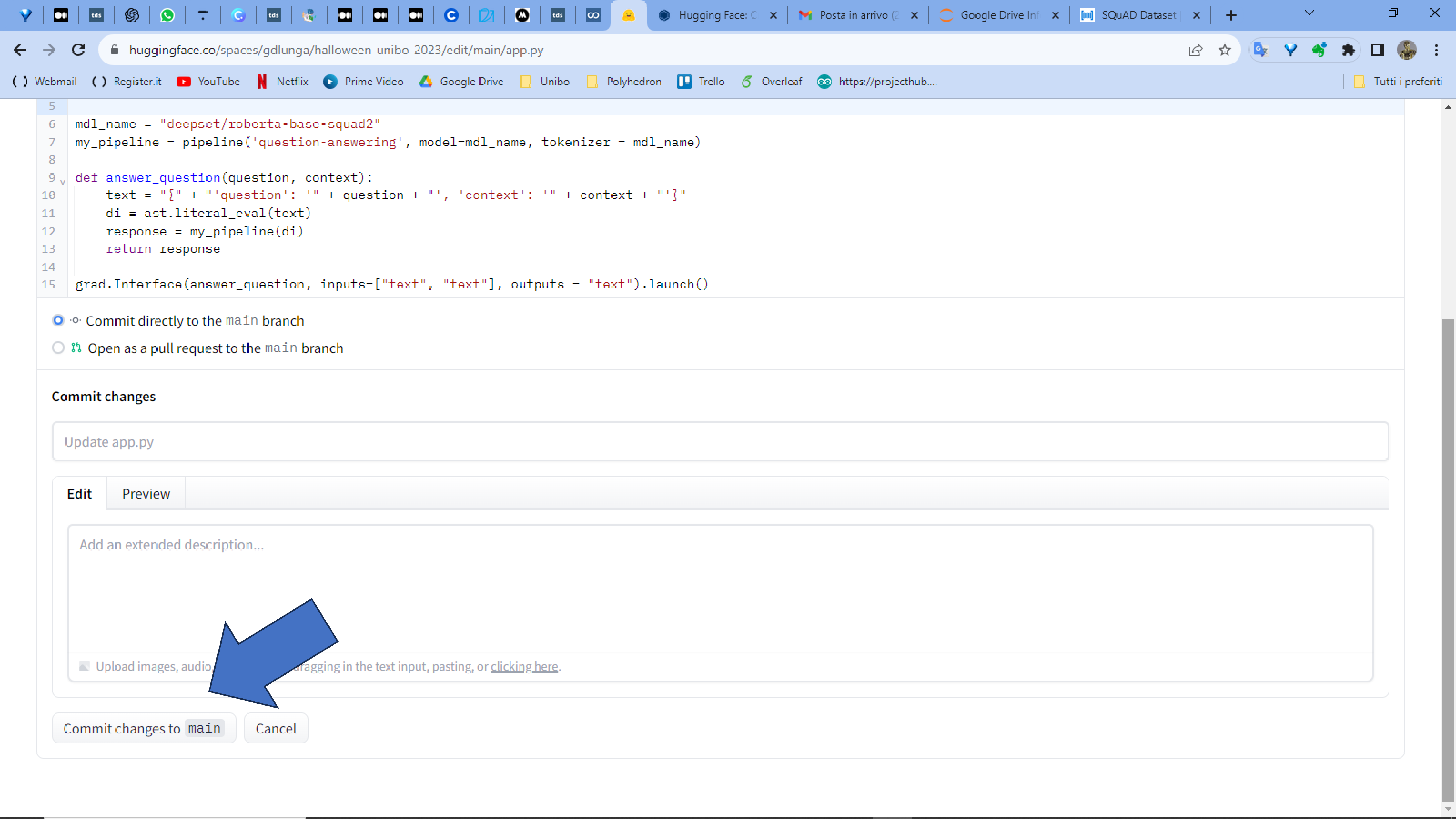
Create a file requirements.txt with the following content:

```
gradio
transformers
torch
```



```
from transformers import AutoModelForQuestionAnswering,
AutoTokenizer, pipeline
import gradio as grad
import ast

mdl_name = "deepset/roberta-base-squad2"
my_pipeline = pipeline('question-answering', model=mdl_name,
tokenizer = mdl_name)

def answer_question(question, context):
    text = "{" + "'question': '" + question + "', 'context':
'" + context + "'}"
    di = ast.literal_eval(text)
    response = my_pipeline(di)
    return response

grad.Interface(answer_question, inputs=["text", "text"],
outputs = "text").launch()
```

Create another file app.py with the code from the textbox



```python

mdl_name = "deepset/roberta-base-squad2"
my_pipeline = pipeline('question-answering', model=mdl_name, tokenizer = mdl_name)

def answer_question(question, context):
    text = "{" + "'question': '" + question + "', 'context': '" + context + "'}"
    di = ast.literal_eval(text)
    response = my_pipeline(di)
    return response

grad.Interface(answer_question, inputs=["text", "text"], outputs = "text").launch()
```

[✓ Commit directly to the main branch]

Open as a pull request to the main branch

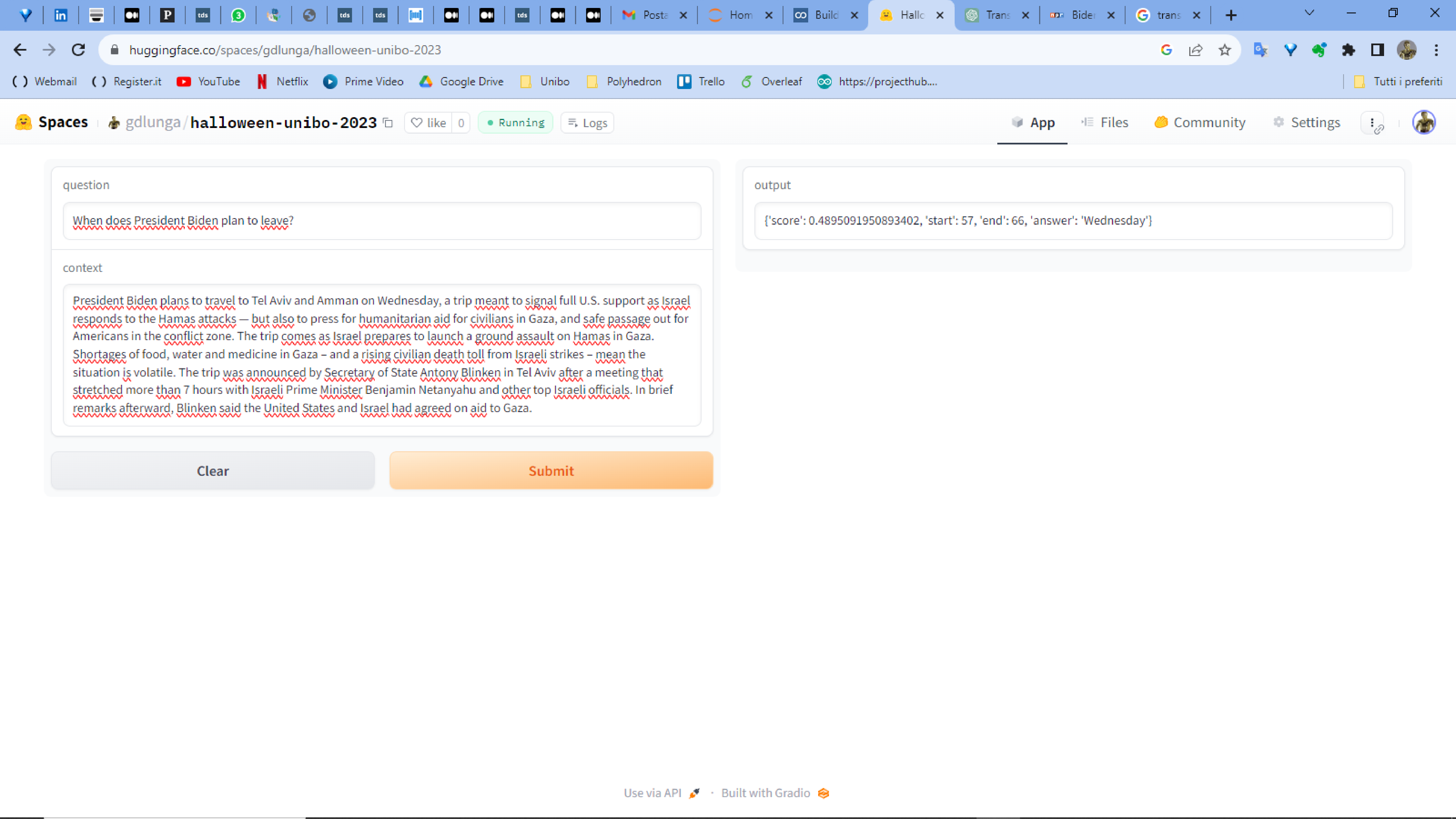
Commit changes

Update app.py

Edit
Preview
Add an extended description...
Upload images, audio, ... dragging in the text input, pasting, or clicking here.
Commit changes to main
Cancel

Hugging Face Tasks – Question and Answering

- Pressing the commit button will trigger the build and deployment process, and one can click the See logs button to see the activity;
- Once this is done, click the App tab, which is to the left of the Files and Versions tab;
- This would present you the UI for keying in the inputs;
- Once inputs are provided, click the Submit button …



huggingface.co/spaces/gdlunga/halloween-unibo-2023

question

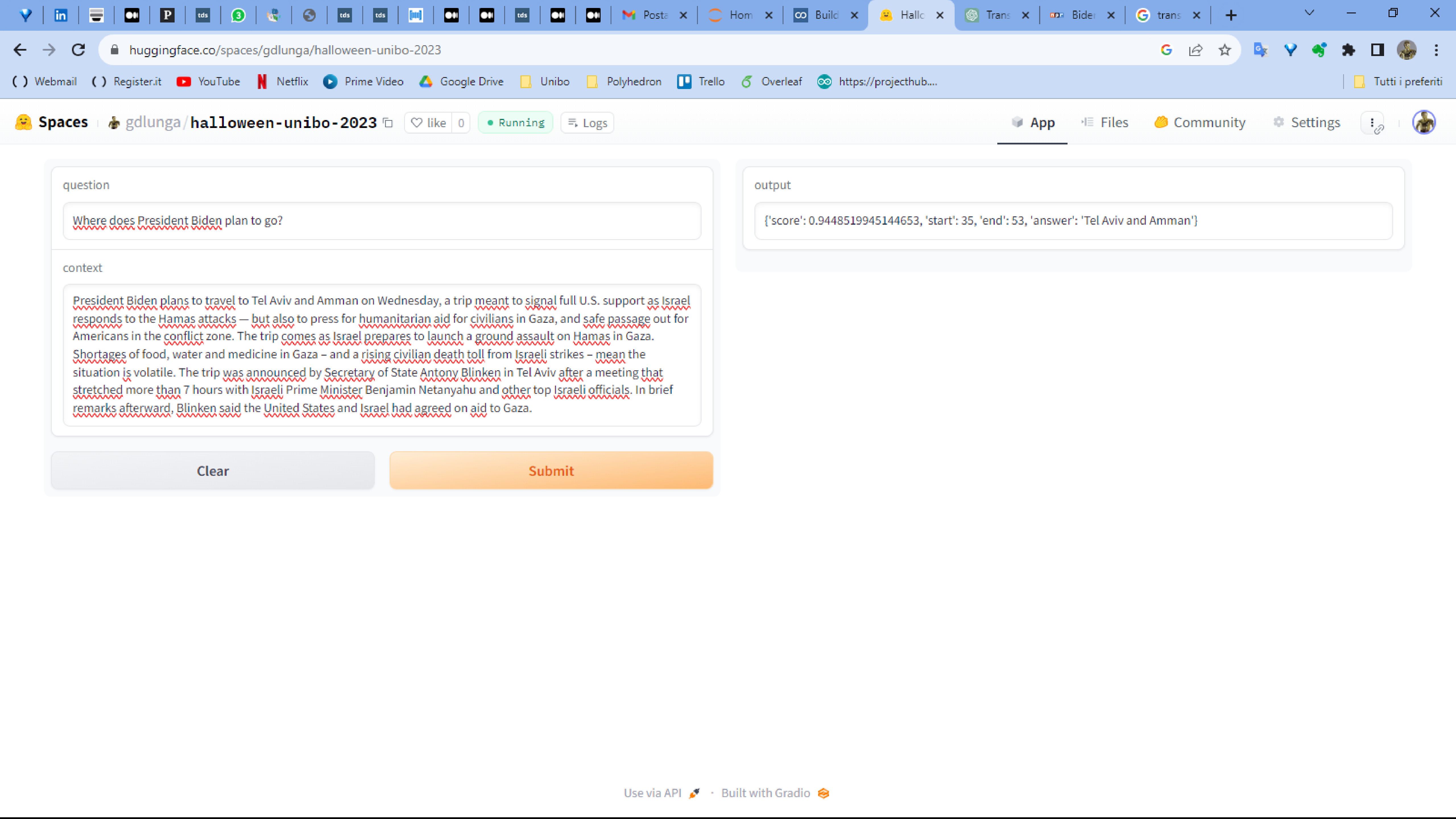
When does President Biden plan to leave?

context

President Biden plans to travel to Tel Aviv and Amman on Wednesday, a trip meant to signal full U.S. support as Israel responds to the Hamas attacks — but also to press for humanitarian aid for civilians in Gaza, and safe passage out for Americans in the conflict zone. The trip comes as Israel prepares to launch a ground assault on Hamas in Gaza. Shortages of food, water and medicine in Gaza – and a rising civilian death toll from Israeli strikes – mean the situation is volatile. The trip was announced by Secretary of State Antony Blinken in Tel Aviv after a meeting that stretched more than 7 hours with Israeli Prime Minister Benjamin Netanyahu and other top Israeli officials. In brief remarks afterward, Blinken said the United States and Israel had agreed on aid to Gaza.

output

```
{'score': 0.4895091950893402, 'start': 57, 'end': 66, 'answer': 'Wednesday'}
```


huggingface.co/spaces/gdlunga/halloween-unibo-2023
Spaces
gdlunga / halloween-unibo-2023
like 0
Running
Logs
App
Files
Community
Settings

question

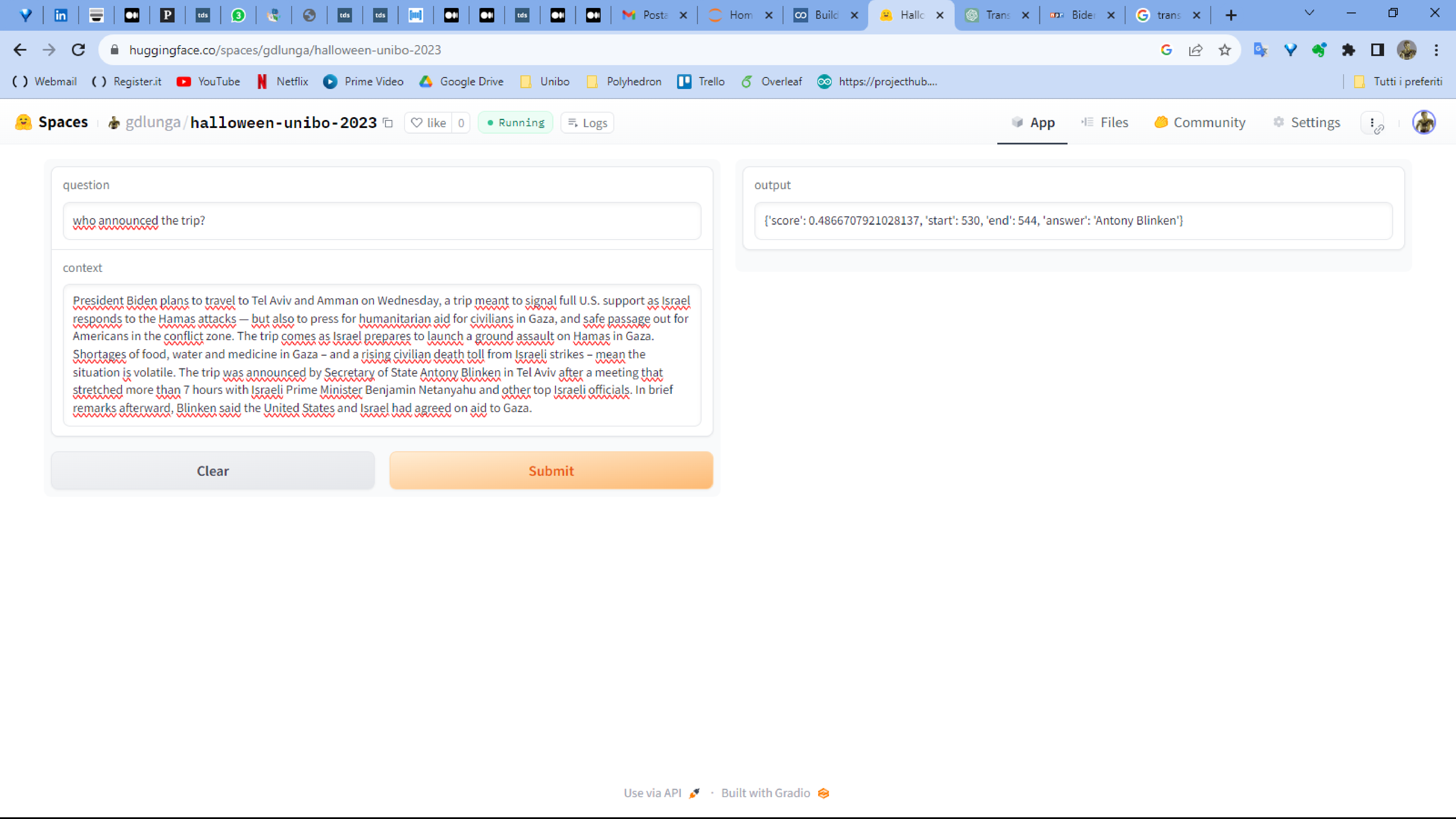
Where does President Biden plan to go?

context

President Biden plans to travel to Tel Aviv and Amman on Wednesday, a trip meant to signal full U.S. support as Israel responds to the Hamas attacks — but also to press for humanitarian aid for civilians in Gaza, and safe passage out for Americans in the conflict zone. The trip comes as Israel prepares to launch a ground assault on Hamas in Gaza. Shortages of food, water and medicine in Gaza – and a rising civilian death toll from Israeli strikes – mean the situation is volatile. The trip was announced by Secretary of State Antony Blinken in Tel Aviv after a meeting that stretched more than 7 hours with Israeli Prime Minister Benjamin Netanyahu and other top Israeli officials. In brief remarks afterward, Blinken said the United States and Israel had agreed on aid to Gaza.

output

```
{'score': 0.9448519945144653, 'start': 35, 'end': 53, 'answer': 'Tel Aviv and Amman'}
```


huggingface.co/spaces/gdlunga/halloween-unibo-2023
Spaces gdlunga / halloween-unibo-2023
like 0
Running
Logs
App
Files
Community
Settings

question

who announced the trip?

context

President Biden plans to travel to Tel Aviv and Amman on Wednesday, a trip meant to signal full U.S. support as Israel responds to the Hamas attacks — but also to press for humanitarian aid for civilians in Gaza, and safe passage out for Americans in the conflict zone. The trip comes as Israel prepares to launch a ground assault on Hamas in Gaza. Shortages of food, water and medicine in Gaza – and a rising civilian death toll from Israeli strikes – mean the situation is volatile. The trip was announced by Secretary of State Antony Blinken in Tel Aviv after a meeting that stretched more than 7 hours with Israeli Prime Minister Benjamin Netanyahu and other top Israeli officials. In brief remarks afterward, Blinken said the United States and Israel had agreed on aid to Gaza.

output

```
{'score': 0.4866707921028137, 'start': 530, 'end': 544, 'answer': 'Antony Blinken'}
```

PDF-Chat: Answer Questions from Your Own PDF

Load your Pdf and ask it questions and get answers from it.

PDF-Chat App

Credits: This example is based on the blog “Building a PDF-Chat App using LangChain, OpenAI API & Streamlit” by Youssef Hosni (https://medium.com/gitconnected/building-a-pdf-chat-app-using-langchain-openai-api-streamlit-3d95c27bda0)

Building a PDF-Chat App - Langchain

- LangChain is a framework built around LLMs. It can be used for chatbots, Generative Question-Answering (GQA), summarization, and much more.
- The core idea of the library is that we can "chain" together different components to create more advanced use cases around LLMs.
- The goal of this application is to use LangChain and OpenAI API to make the user load a certain pdf file and ask questions to be answered from this Pdf.

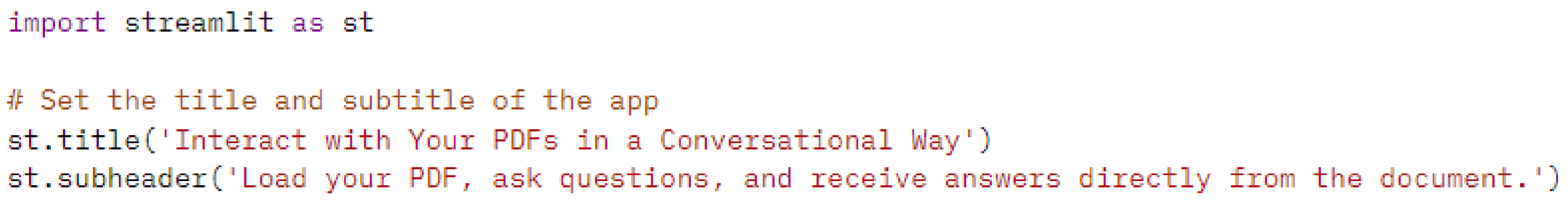
Building a PDF-Chat App - Langchain

Chains may consist of multiple components from several modules:

- **Prompt templates**: Prompt templates are templates for different types of prompts. Like "chatbot" style templates, ELI5 question-answering, etc
- **LLMs**: Large language models like GPT-3, BLOOM, etc
- **Agents**: Agents use LLMs to decide what actions should be taken. Tools like web search or calculators can be used, and all are packaged into a logical loop of operations.
- **Parser**: Parsers is on the opposite end of Prompts. It involves taking the output of these models and parsing it into a more structured format so that you can do things downstream with it.
- **Memory**: Short-term memory, long-term memory.

Building a PDF-Chat App – App Interface

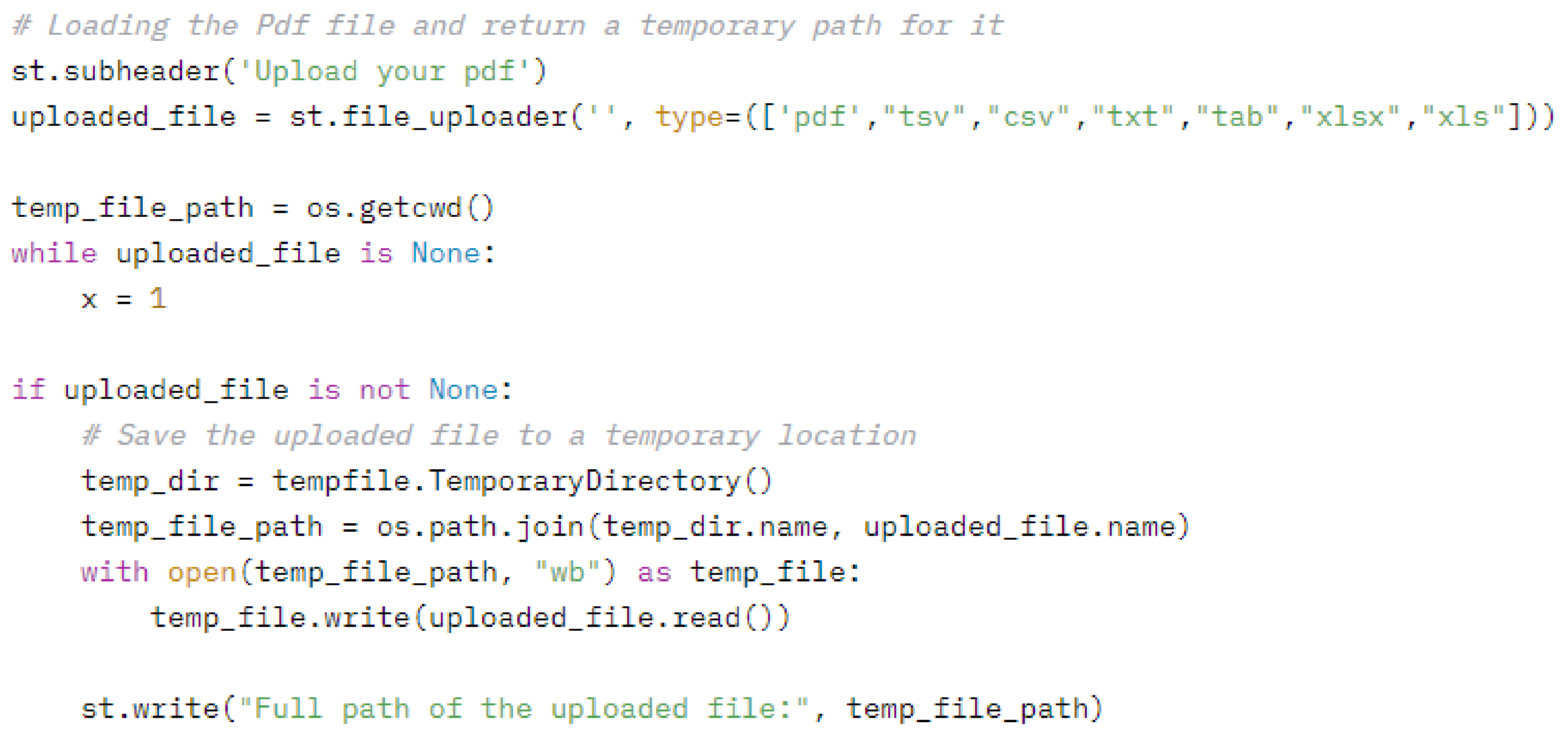
- \textbf{Design the Application Interface}
- We build the user interface by setting the main application page which will include setting the title, subtitle, and the main image of the application.
- This is done using the Streamlit package.
- we start by setting the title and subtitle for a PDF-Chat application interface.
- The title is "PDF-Chat: Interact with Your PDFs in a Conversational Way" and the subtitle is "Load your PDF, ask questions, and receive answers directly from the document."



```
import streamlit as st

# Set the title and subtitle of the app
st.title('Interact with Your PDFs in a Conversational Way')
st.subheader('Load your PDF, ask questions, and receive answers directly from the document.')
```

Building a PDF-Chat App – Upload PDF File

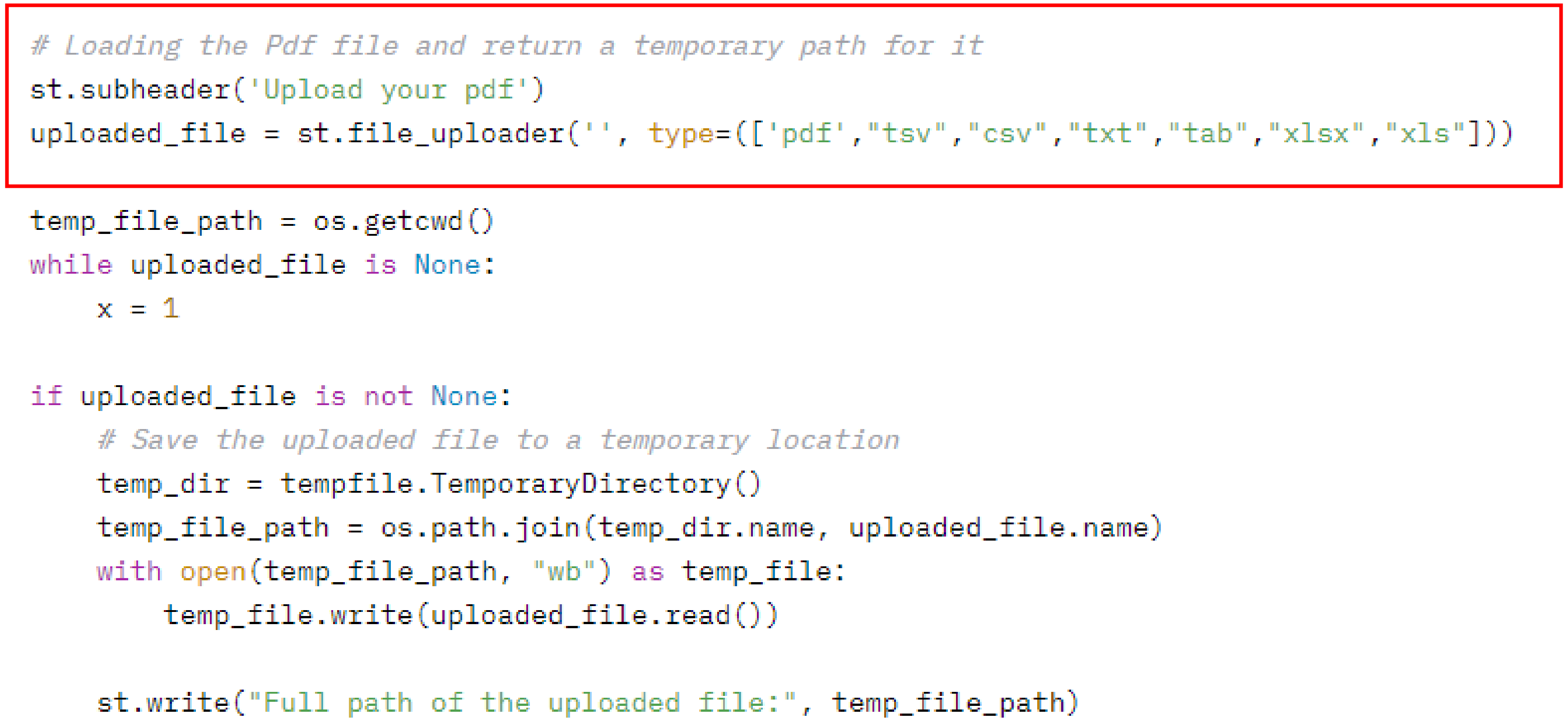
- Upload the Pdf file & Return the File Path
- The second step is to allow the user to load a pdf file and also to return a temporary path of this file to be able to load the file after that using LangChain with this temporary path.
- Let’s see the code step by step …



```
# Loading the Pdf file and return a temporary path for it
st.subheader('Upload your pdf')
uploaded_file = st.file_uploader('', type=(['pdf',"tsv","csv","txt","tab","xlsx","xls"]))

temp_file_path = os.getcwd()
while uploaded_file is None:
    x = 1

if uploaded_file is not None:
    # Save the uploaded file to a temporary location
    temp_dir = tempfile.TemporaryDirectory()
    temp_file_path = os.path.join(temp_dir.name, uploaded_file.name)
    with open(temp_file_path, "wb") as temp_file:
        temp_file.write(uploaded_file.read())

    st.write("Full path of the uploaded file:", temp_file_path)
```

Building a PDF-Chat App – Upload PDF File

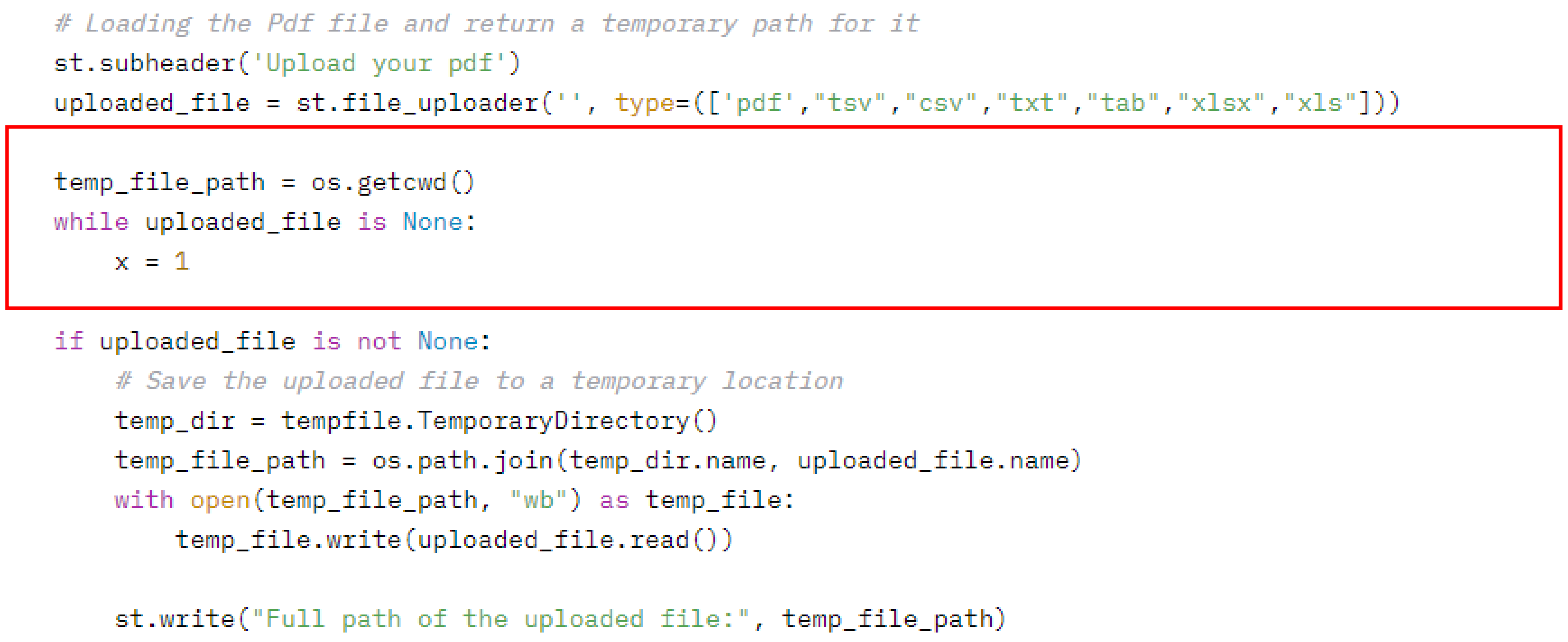
- The code displays a subheader with the text "Upload your pdf" to indicate that the user should upload a PDF file.
- The st.file_uploader() function creates a file uploader component where the user can select a file. It allows the user to upload files of various types, including PDF, TSV, CSV, TXT, TAB, XLSX, and XLS.



```python
# Loading the Pdf file and return a temporary path for it
st.subheader('Upload your pdf')
uploaded_file = st.file_uploader('', type=(['pdf',"tsv","csv","txt","tab","xlsx","xls"]))

temp_file_path = os.getcwd()
while uploaded_file is None:
    x = 1

if uploaded_file is not None:
    # Save the uploaded file to a temporary location
    temp_dir = tempfile.TemporaryDirectory()
    temp_file_path = os.path.join(temp_dir.name, uploaded_file.name)
    with open(temp_file_path, "wb") as temp_file:
        temp_file.write(uploaded_file.read())

    st.write("Full path of the uploaded file:", temp_file_path)
```

Building a PDF-Chat App – Upload PDF File

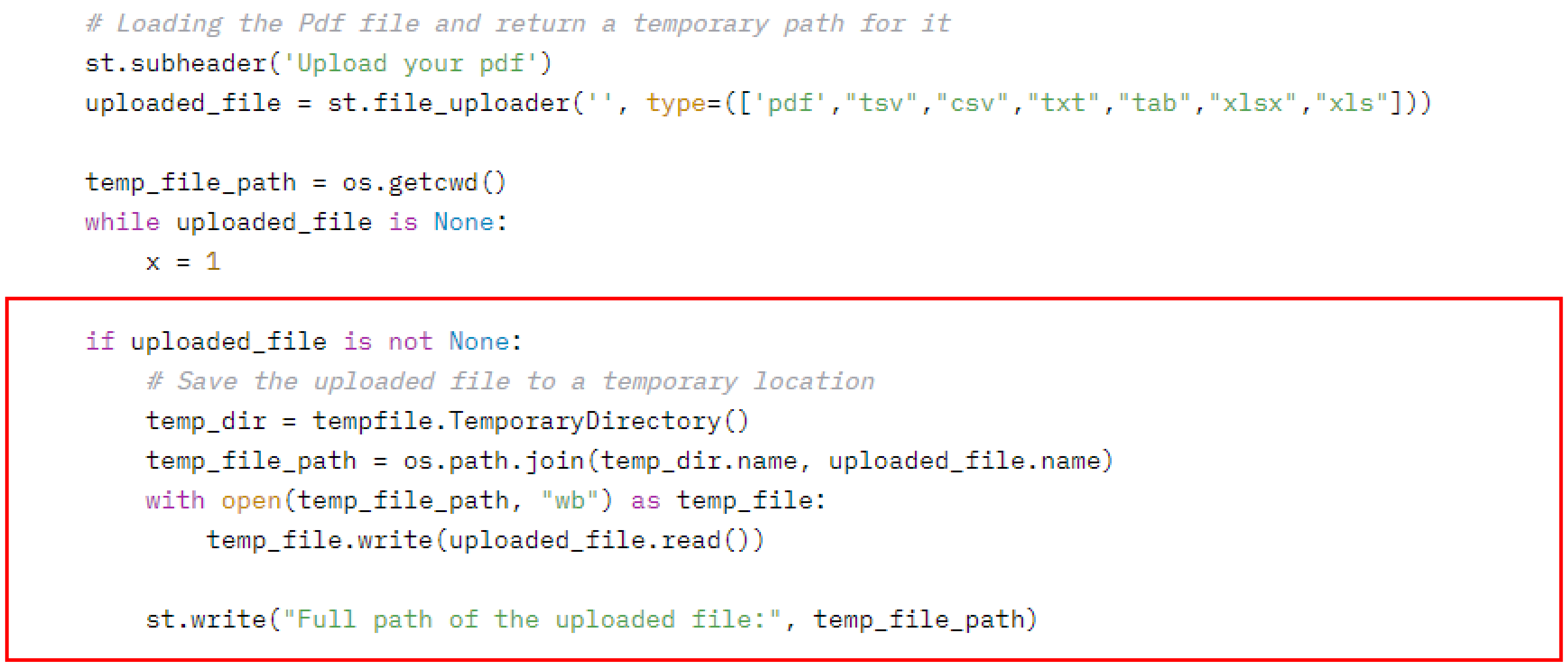
- The variable temp_file_path is initialized with the current working directory using os.getcwd(). This will serve as the default temporary file path if no file is uploaded.
- The code enters a loop that continues as long as uploaded_file is None, indicating that no file has been uploaded yet. Within the loop, a variable x is assigned the value 1, which doesn't appear to have any practical purpose in this snippet.



```python
# Loading the Pdf file and return a temporary path for it
st.subheader('Upload your pdf')
uploaded_file = st.file_uploader('', type=(['pdf',"tsv","csv","txt","tab","xlsx","xls"]))

temp_file_path = os.getcwd()
while uploaded_file is None:
    x = 1

if uploaded_file is not None:
    # Save the uploaded file to a temporary location
    temp_dir = tempfile.TemporaryDirectory()
    temp_file_path = os.path.join(temp_dir.name, uploaded_file.name)
    with open(temp_file_path, "wb") as temp_file:
        temp_file.write(uploaded_file.read())

    st.write("Full path of the uploaded file:", temp_file_path)
```

Building a PDF-Chat App – Upload PDF File

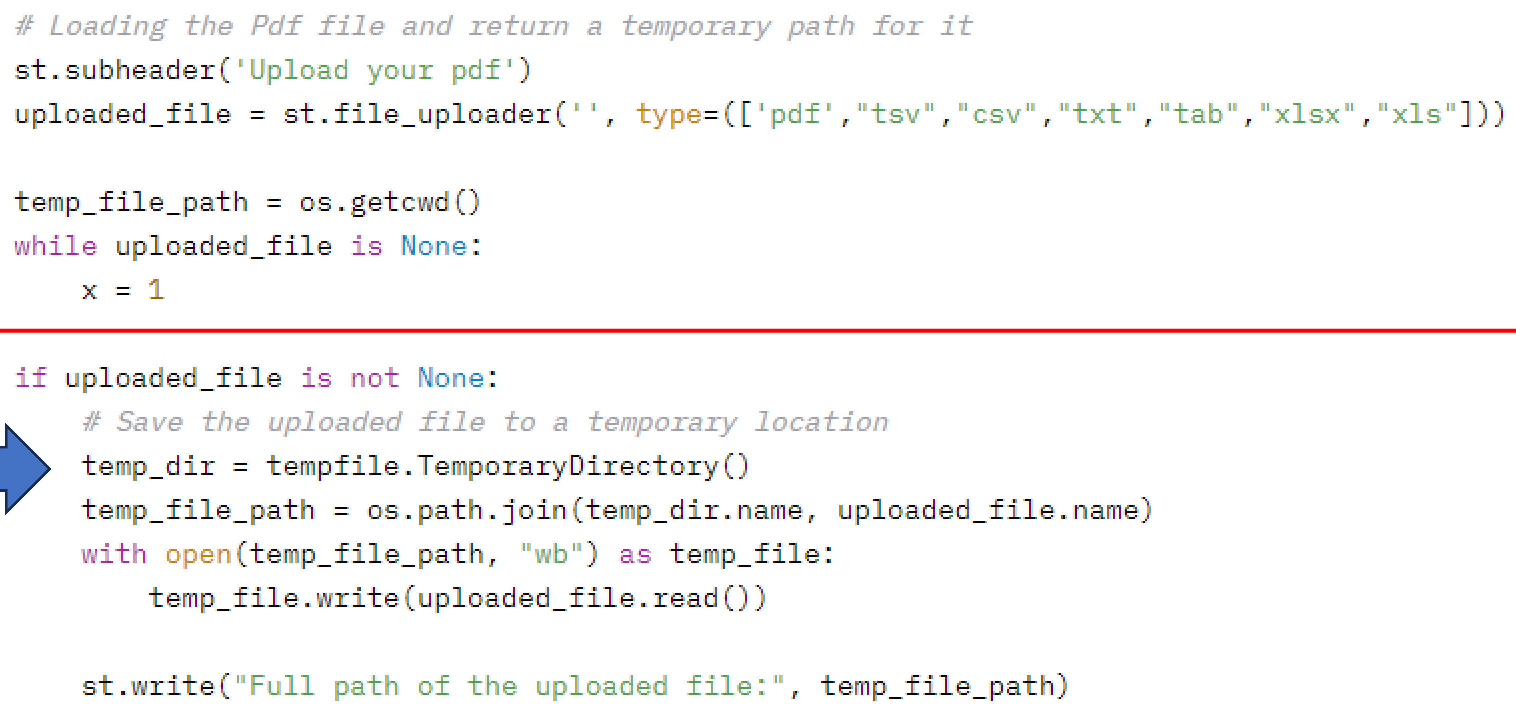
- The variable temp_file_path is initialized with the current working directory using os.getcwd(). This will serve as the default temporary file path if no file is uploaded.
- The code enters a loop that continues as long as uploaded_file is None, indicating that no file has been uploaded yet.



```
# Loading the Pdf file and return a temporary path for it
st.subheader('Upload your pdf')
uploaded_file = st.file_uploader('', type=(['pdf',"tsv","csv","txt","tab","xlsx","xls"]))

temp_file_path = os.getcwd()
while uploaded_file is None:
    x = 1

if uploaded_file is not None:
    # Save the uploaded file to a temporary location
    temp_dir = tempfile.TemporaryDirectory()
    temp_file_path = os.path.join(temp_dir.name, uploaded_file.name)
    with open(temp_file_path, "wb") as temp_file:
        temp_file.write(uploaded_file.read())

    st.write("Full path of the uploaded file:", temp_file_path)
```

Building a PDF-Chat App – Upload PDF File

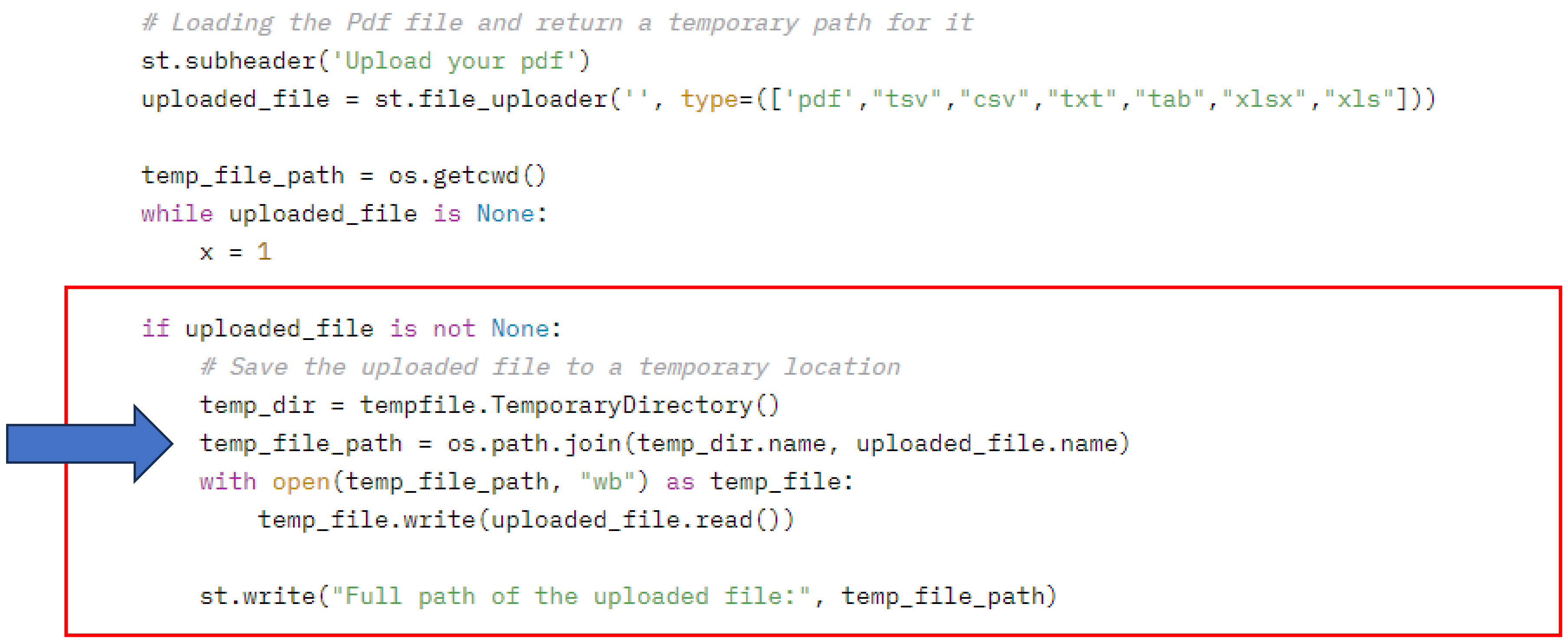
- Once a file is uploaded (uploaded_file is not None), the code creates a temporary directory using tempfile.TemporaryDirectory() and assigns it to the variable temp_dir.
- The full file path for the uploaded file is constructed by joining the temporary directory path (temp_dir.name) with the name of the uploaded file (uploaded_file.name). This resulting path is assigned to the variable temp_file_path.



```python
# Loading the Pdf file and return a temporary path for it
st.subheader('Upload your pdf')
uploaded_file = st.file_uploader('', type=(['pdf',"tsv","csv","txt","tab","xlsx","xls"]))

temp_file_path = os.getcwd()
while uploaded_file is None:
    x = 1

if uploaded_file is not None:
    # Save the uploaded file to a temporary location
    temp_dir = tempfile.TemporaryDirectory()
    temp_file_path = os.path.join(temp_dir.name, uploaded_file.name)
    with open(temp_file_path, "wb") as temp_file:
        temp_file.write(uploaded_file.read())

    st.write("Full path of the uploaded file:", temp_file_path)
```

Building a PDF-Chat App – Upload PDF File

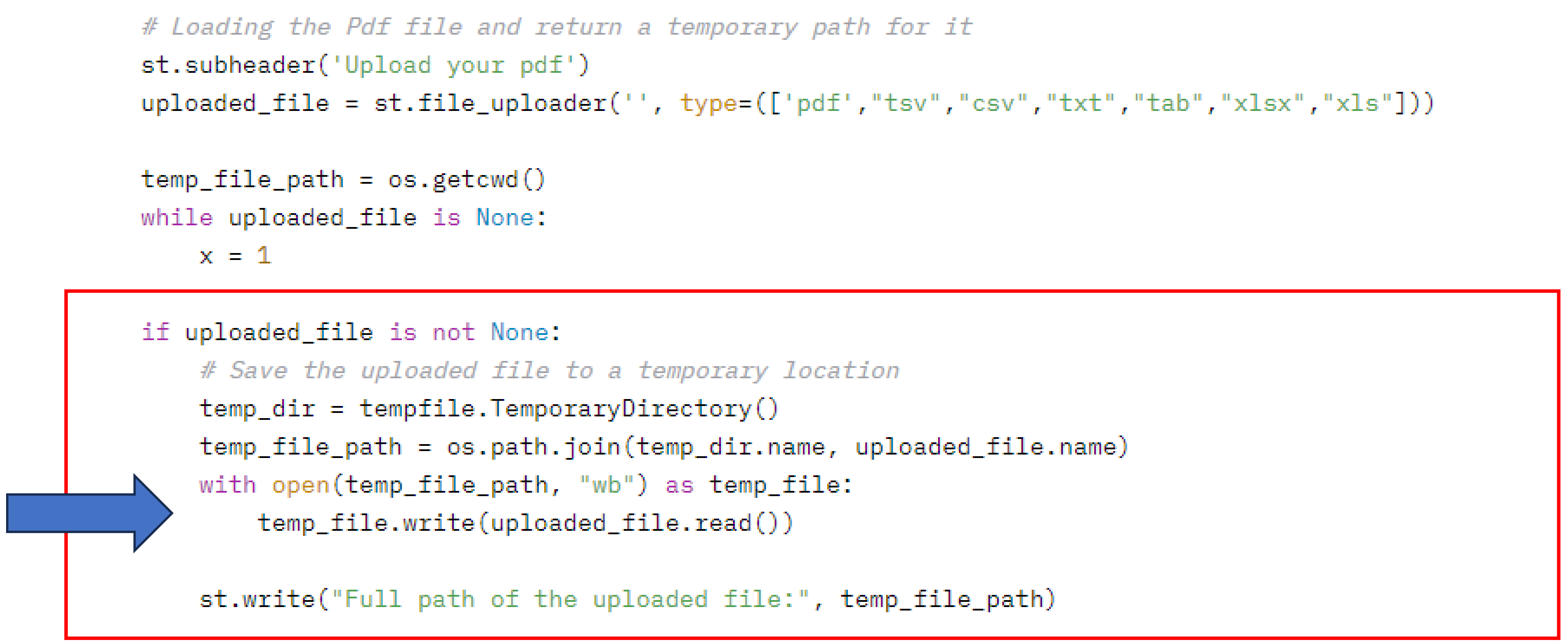
- Once a file is uploaded (uploaded_file is not None), the code creates a temporary directory using tempfile.TemporaryDirectory() and assigns it to the variable temp_dir.
- The full file path for the uploaded file is constructed by joining the temporary directory path (temp_dir.name) with the name of the uploaded file (uploaded_file.name). This resulting path is assigned to the variable temp_file_path.



```python
# Loading the Pdf file and return a temporary path for it
st.subheader('Upload your pdf')
uploaded_file = st.file_uploader('', type=(['pdf',"tsv","csv","txt","tab","xlsx","xls"]))

temp_file_path = os.getcwd()
while uploaded_file is None:
    x = 1

if uploaded_file is not None:
    # Save the uploaded file to a temporary location
    temp_dir = tempfile.TemporaryDirectory()
    temp_file_path = os.path.join(temp_dir.name, uploaded_file.name)
    with open(temp_file_path, "wb") as temp_file:
        temp_file.write(uploaded_file.read())

    st.write("Full path of the uploaded file:", temp_file_path)
```

Building a PDF-Chat App – Upload PDF File

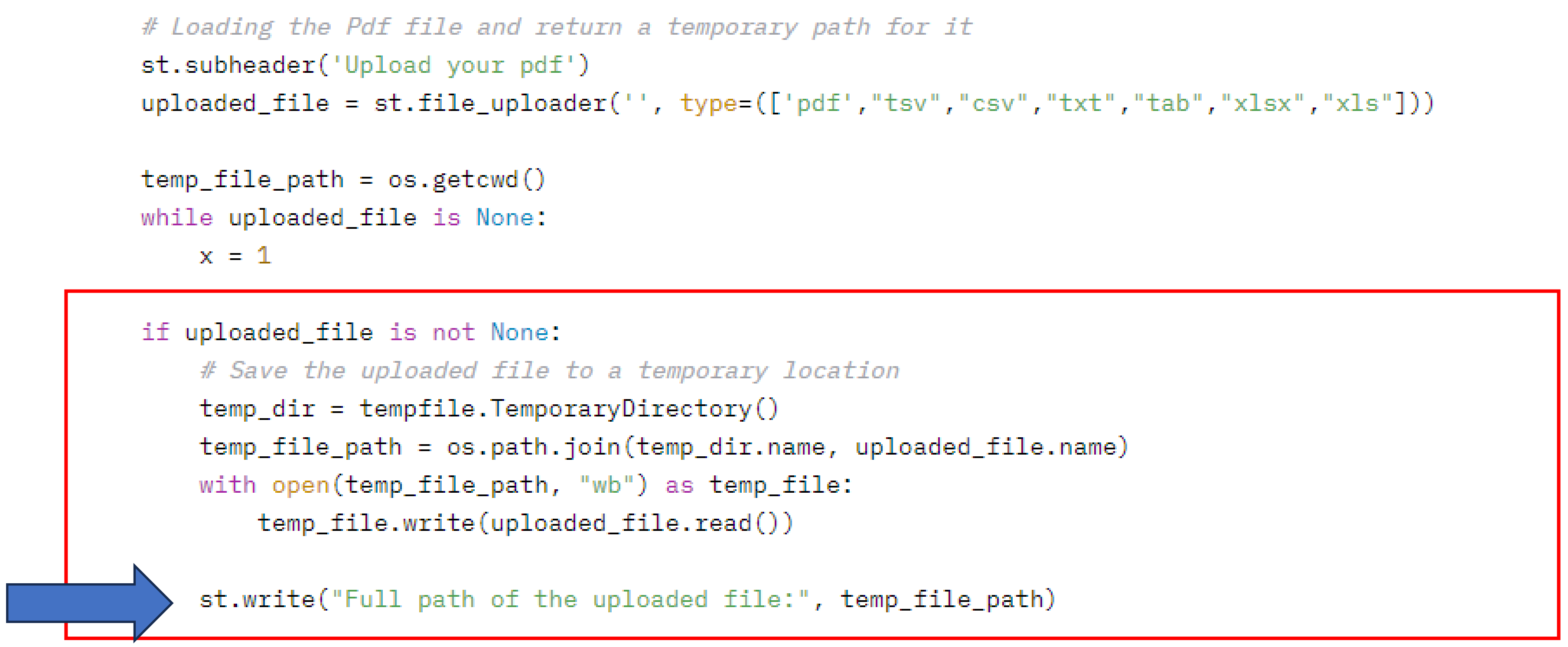
- The code opens the temporary file in write-binary mode ("wb") and writes the contents of the uploaded file to it.
- Finally, it displays the full path of the uploaded file using st.write().



```python
# Loading the Pdf file and return a temporary path for it
st.subheader('Upload your pdf')
uploaded_file = st.file_uploader('', type=(['pdf',"tsv","csv","txt","tab","xlsx","xls"]))

temp_file_path = os.getcwd()
while uploaded_file is None:
    x = 1

if uploaded_file is not None:
    # Save the uploaded file to a temporary location
    temp_dir = tempfile.TemporaryDirectory()
    temp_file_path = os.path.join(temp_dir.name, uploaded_file.name)
    with open(temp_file_path, "wb") as temp_file:
        temp_file.write(uploaded_file.read())

    st.write("Full path of the uploaded file:", temp_file_path)
```

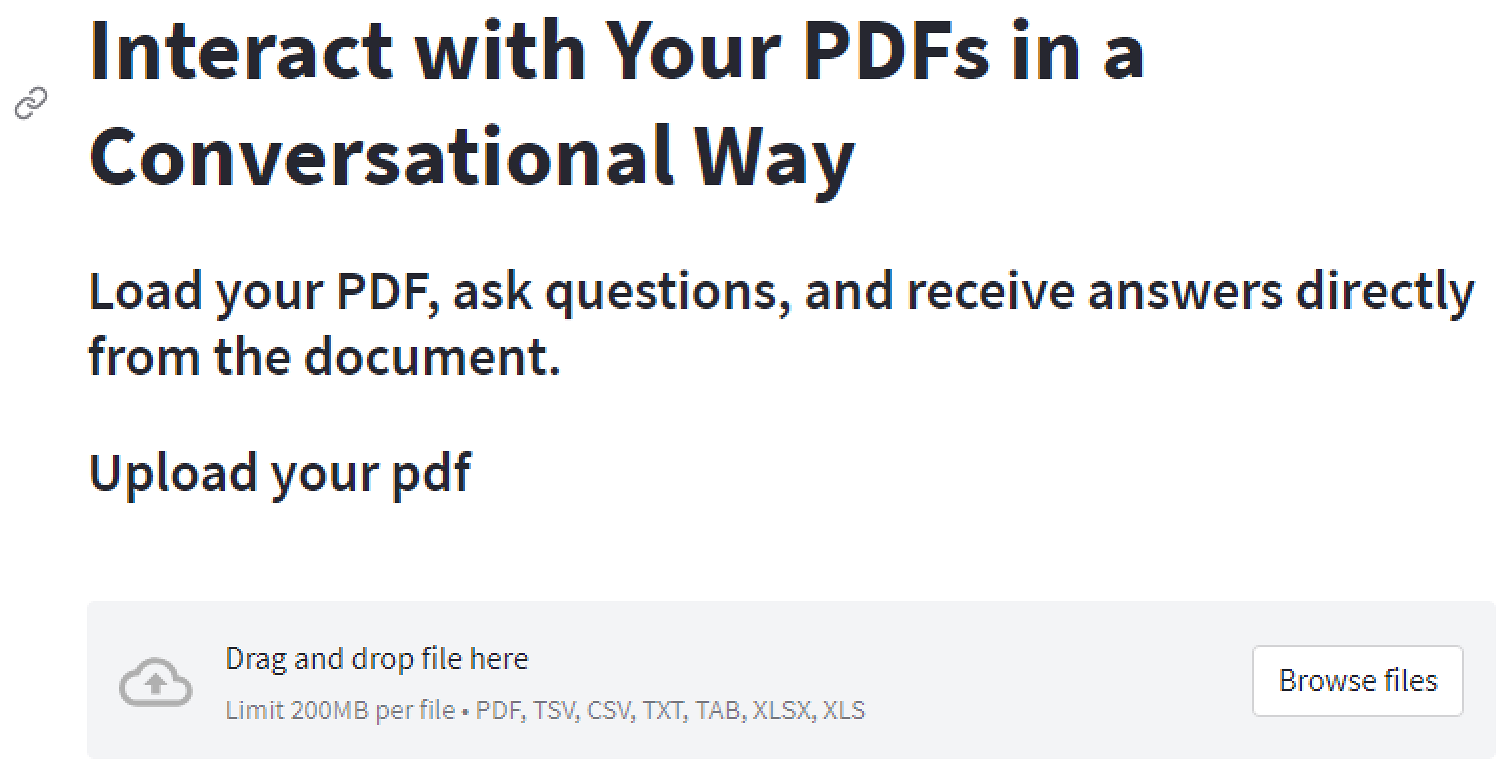
Building a PDF-Chat App – Upload PDF File

- The code opens the temporary file in write-binary mode ("wb") and writes the contents of the uploaded file to it.
- Finally, it displays the full path of the uploaded file using st.write().



```python
# Loading the Pdf file and return a temporary path for it
st.subheader('Upload your pdf')
uploaded_file = st.file_uploader('', type=(['pdf',"tsv","csv","txt","tab","xlsx","xls"]))

temp_file_path = os.getcwd()
while uploaded_file is None:
    x = 1

if uploaded_file is not None:
    # Save the uploaded file to a temporary location
    temp_dir = tempfile.TemporaryDirectory()
    temp_file_path = os.path.join(temp_dir.name, uploaded_file.name)
    with open(temp_file_path, "wb") as temp_file:
        temp_file.write(uploaded_file.read())

    st.write("Full path of the uploaded file:", temp_file_path)
```

Building a PDF-Chat App


Interact with Your PDFs in a Conversational Way

Load your PDF, ask questions, and receive answers directly from the document.

Upload your pdf

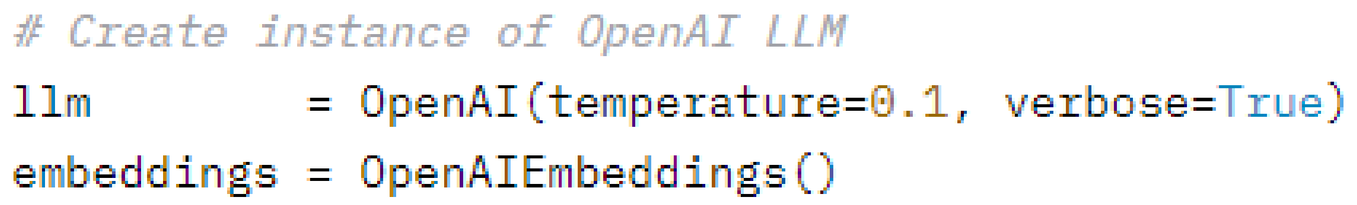
Drag and drop file here

Limit 200MB per file • PDF, TSV, CSV, TXT, TAB, XLSX, XLS

- After the file is uploaded and the file path is returned it is time to load the file and process it using LangChain…

Building a PDF-Chat App – Langchain and OpenAI API

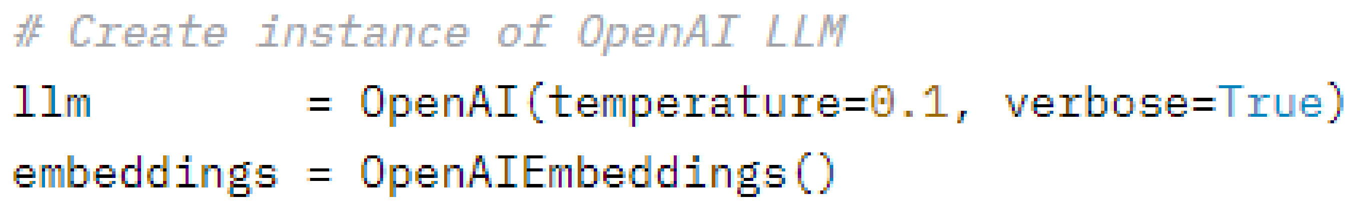
- **Setup OpenAI API**
- To be able to use the OpenAI API you will need to set up the API key for the OpenAI language model service and creates an instance of the OpenAI language model (LLM).
- **Please ensure that you have a valid API key and appropriate access to the OpenAI service to use this code effectively.**



```
# Create instance of OpenAI LLM
llm        = OpenAI(temperature=0.1, verbose=True)
embeddings = OpenAIEmbeddings()
```

Building a PDF-Chat App

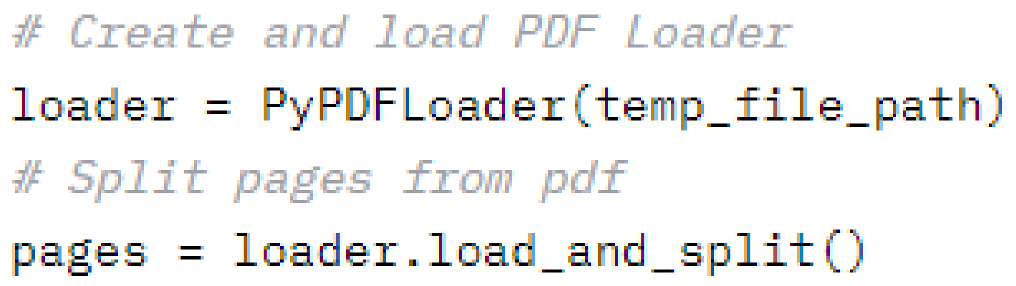
- An instance of the OpenAI language model is created using the **`OpenAI()`** constructor.
- The temperature parameter sets the randomness of the generated text (lower values make it more focused), and **`verbose=True`** enables verbose mode, which provides additional information during text generation.
- Additionally, an instance of **`OpenAIEmbeddings()`** is created.



```
# Create instance of OpenAI LLM
llm        = OpenAI(temperature=0.1, verbose=True)
embeddings = OpenAIEmbeddings()
```

Building a PDF-Chat App - Load and Processing Pdf file

- In this step, we will load the Pdf file and process it to make it ready to answer your questions.
- We will start first by creating a pdf file loader and loading the pdf file and after that, we will split it into separate pages.



```
# Create and load PDF Loader
loader = PyPDFLoader(temp_file_path)
# Split pages from pdf
pages = loader.load_and_split()
```

Building a PDF-Chat App - Load and Processing Pdf file

- The `PyPDFLoader` class is instantiated with the `temp_file_path` variable, which contains the path to the uploaded PDF file. This class is responsible for loading and handling the PDF file.
- The loader.load_and_split() method is called on the loader object. This method loads the PDF file and splits it into individual pages, returning a collection of these pages.
- The resulting pages are stored in the pages variable.

```
# Create and load PDF Loader
loader = PyPDFLoader(temp_file_path)
# Split pages from pdf
pages = loader.load_and_split()
```

Building a PDF-Chat App - Load and Processing Pdf file

- The PyPDFLoader class is instantiated with the temp_file_path variable, which contains the path to the uploaded PDF file. This class is responsible for loading and handling the PDF file.
- The **`loader.load_and_split()`** method is called on the loader object. This method loads the PDF file and splits it into individual pages, returning a collection of these pages.
- The resulting pages are stored in the **`pages`** variable.

```python
# Create and load PDF Loader
loader = PyPDFLoader(temp_file_path)
# Split pages from pdf
pages = loader.load_and_split()
```

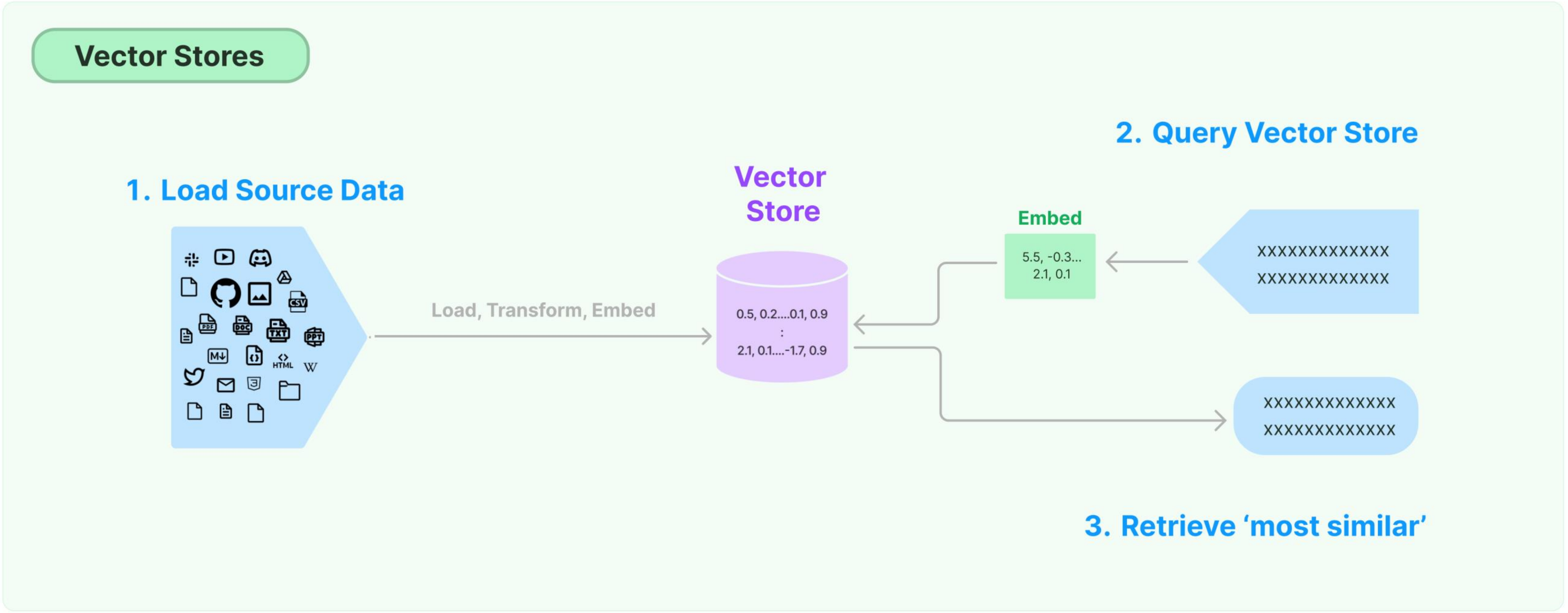
Building a PDF-Chat App - Load and Processing Pdf file

- Next, we will convert the individual pages of a PDF document into a vector database called ChromaDB and create a VectorStoreInfo object which will increase the search speed.
- Next, we will convert the document store into a LangChain toolkit.
- Finally, we will need to create an end-to-end LangChain agent executor to be able to extract the information from the pdf file.

Building a PDF-Chat App – What is a Vector DB?

- One of the most common ways to store and search over unstructured data is to embed it and store the resulting embedding vectors, and then at query time to embed the unstructured query and retrieve the embedding vectors that are 'most similar' to the embedded query.
- A vector store takes care of storing embedded data and performing vector search for you.
- Chroma DB is an open-source vector storage system (vector database) designed for the storing and retrieving vector embeddings.
- Its primary function is to store embeddings with associated metadata for subsequent use by extensive language models.
- Moreover, it can serve as a foundation for semantic search engines that operate on textual data.
- Vector database offers an ideal solution for managing large volumes of unstructured and semi-structured data.

Building a PDF-Chat App – What is a Vector DB

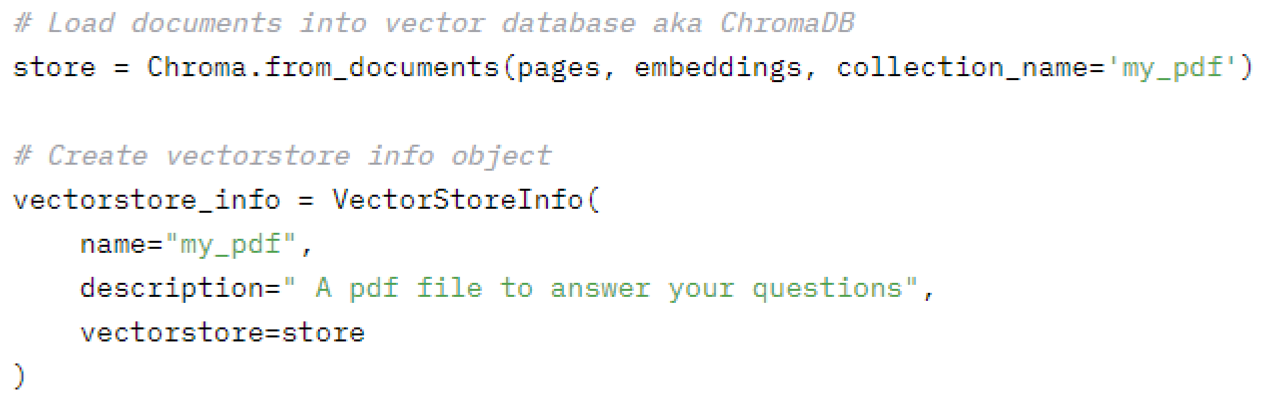


Building a PDF-Chat App – Langchain VectorStoreInfo

VectorStoreInfo is used to provide metadata and configuration details for a vector store when creating a VectorStoreRouterToolkit. It contains:

- name - The name of the vector store
- description - A description of the vector store and what kind of data it contains
- vectorstore - The actual VectorStore instance

So in summary, VectorStoreInfo holds metadata and configuration for each vector store, which is then passed to some Toolkit to enable other process



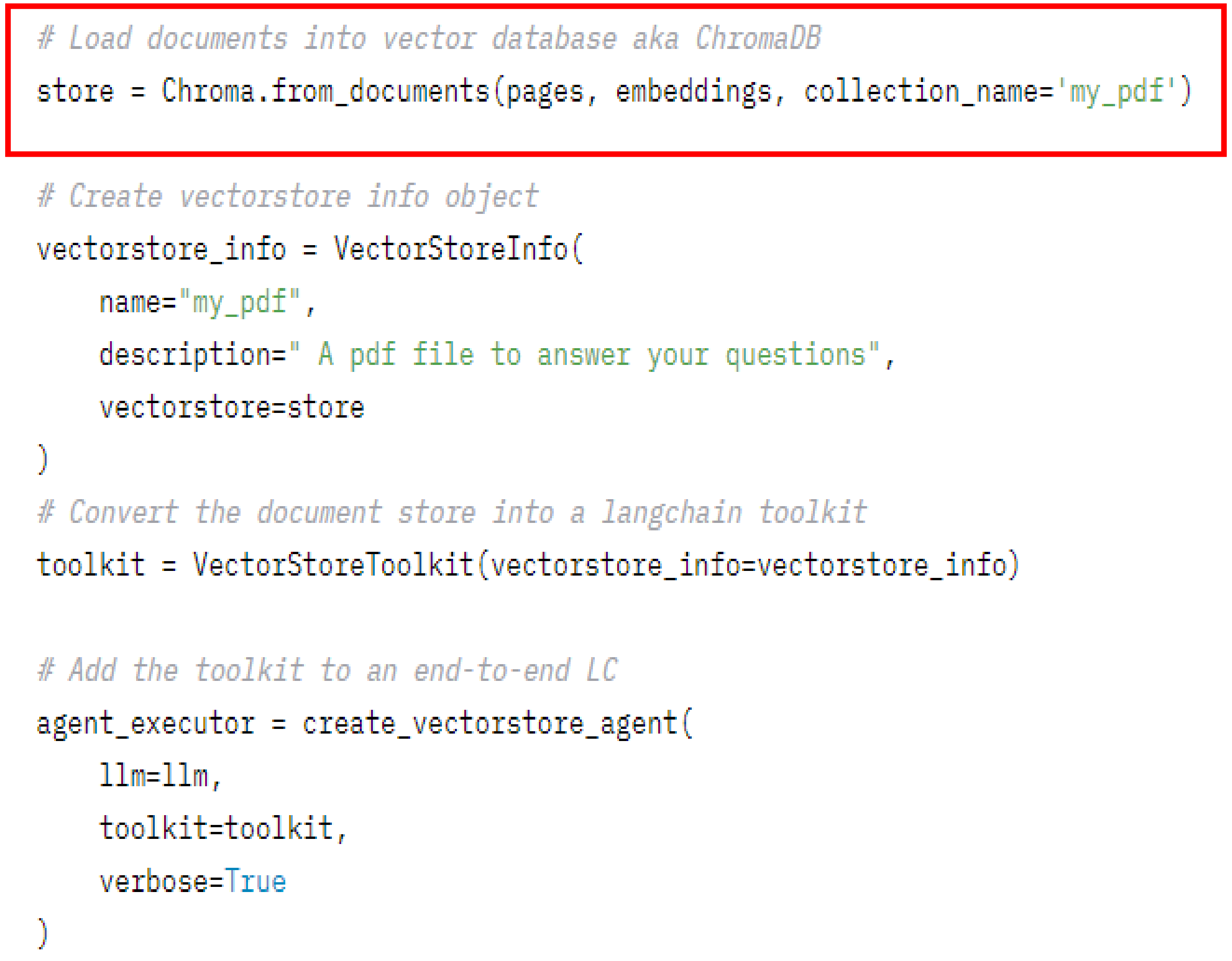
```
# Load documents into vector database aka ChromaDB
store = Chroma.from_documents(pages, embeddings, collection_name='my_pdf')

# Create vectorstore info object
vectorstore_info = VectorStoreInfo(
    name="my_pdf",
    description=" A pdf file to answer your questions",
    vectorstore=store
)
```

Building a PDF-Chat App – Langchain Toolkit and Agents

- In Langchain a “Tool” is a specific abstraction around a function that makes it easy for a language model to interact with it.
- Specificlaly, the interface of a tool has a single text input and a single text output.
- Some applications will require not just a predetermined chain of calls to LLMs/other tools, but potentially an unknown chain that depends on the user's input.
- In these types of chains, there is a “agent” which has access to a suite of tools.
- Depending on the user input, the agent can then decide which, if any, of these tools to call.
- An Agent Executor is an Agent and set of Tools.
- The agent executor is responsible for calling the agent, getting back and action and action input, calling the tool that the action references with the corresponding input, getting the output of the tool, and then passing all that information back into the Agent to get the next action it should take

Building a PDF-Chat App - Load and Processing Pdf file

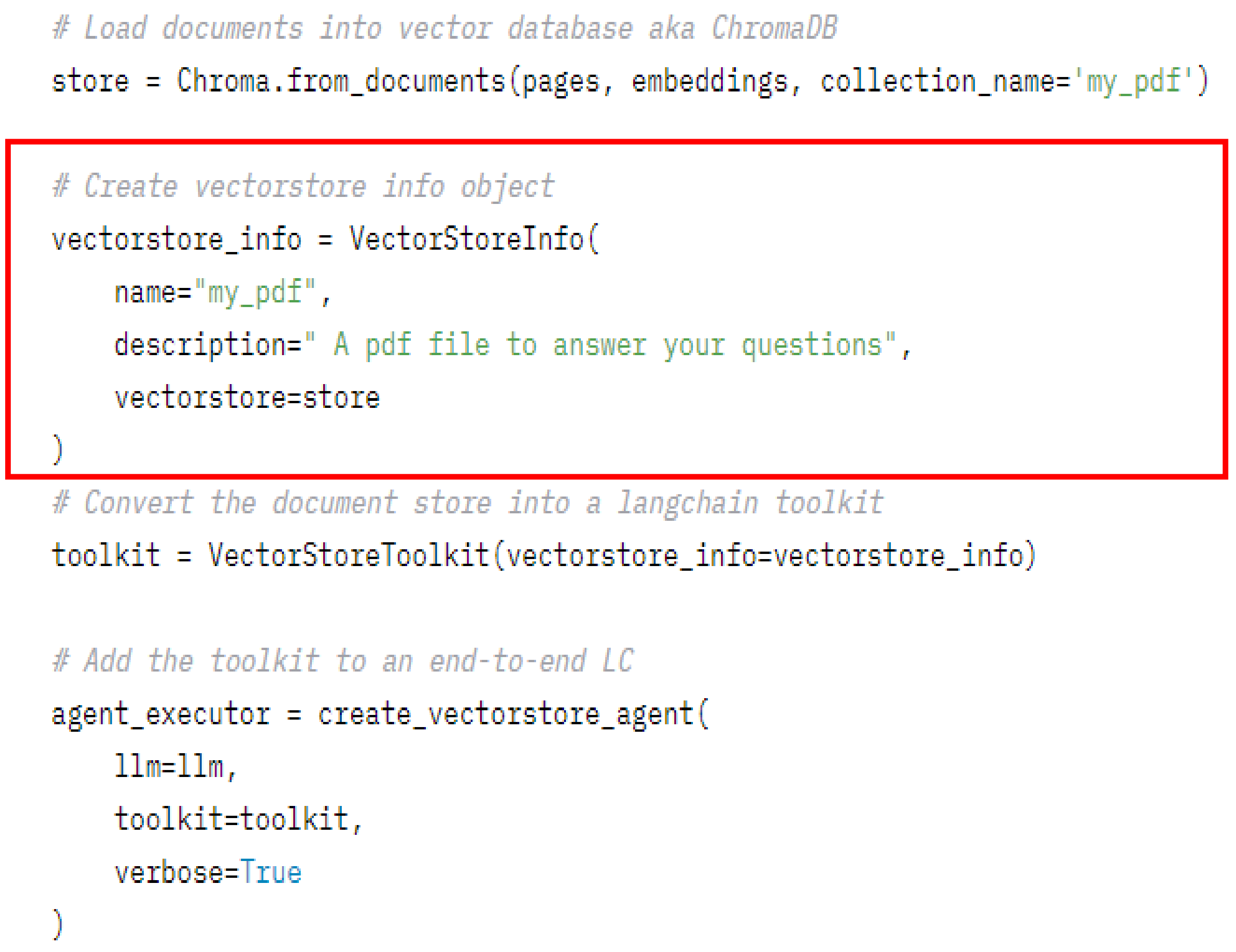
- The **`Chroma.from_documents()`** method is called on the pages variable, which contains the individual pages of the PDF.
- This method takes the pages and the embeddings object and creates a vector database or store.



```
# Load documents into vector database aka ChromaDB
store = Chroma.from_documents(pages, embeddings, collection_name='my_pdf')

# Create vectorstore info object
vectorstore_info = VectorStoreInfo(
    name="my_pdf",
    description=" A pdf file to answer your questions",
    vectorstore=store
)
# Convert the document store into a langchain toolkit
toolkit = VectorStoreToolkit(vectorstore_info=vectorstore_info)

# Add the toolkit to an end-to-end LC
agent_executor = create_vectorstore_agent(
    llm=llm,
    toolkit=toolkit,
    verbose=True
)
```

Building a PDF-Chat App - Load and Processing Pdf file

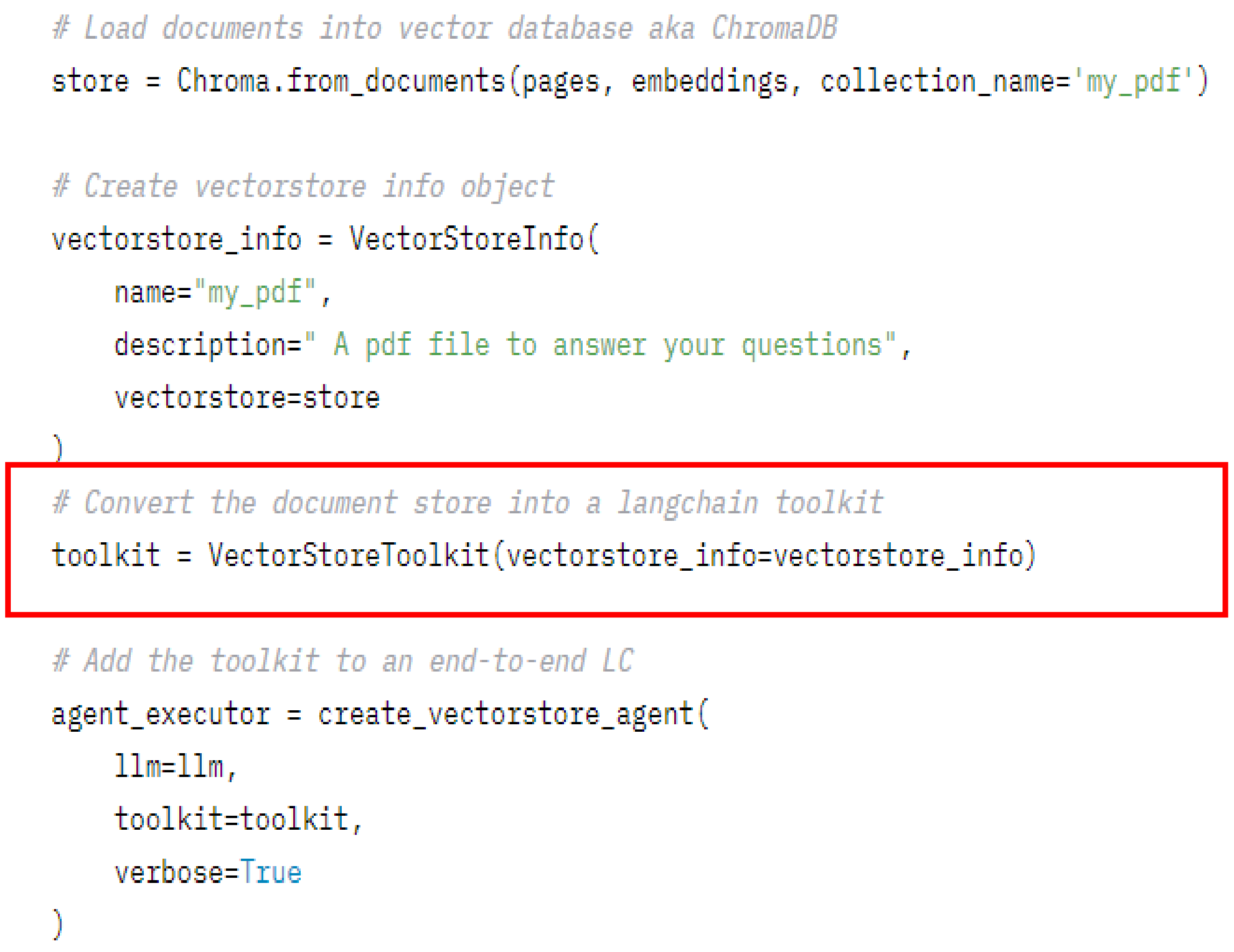
- A `VectorStoreInfo` object is created using the **`VectorStoreInfo()`** constructor.
- It provides information about the vector store, including its name, description, and the previously created store object.



```
# Load documents into vector database aka ChromaDB
store = Chroma.from_documents(pages, embeddings, collection_name='my_pdf')

# Create vectorstore info object
vectorstore_info = VectorStoreInfo(
    name="my_pdf",
    description=" A pdf file to answer your questions",
    vectorstore=store
)
# Convert the document store into a langchain toolkit
toolkit = VectorStoreToolkit(vectorstore_info=vectorstore_info)

# Add the toolkit to an end-to-end LC
agent_executor = create_vectorstore_agent(
    llm=llm,
    toolkit=toolkit,
    verbose=True
)
```

Building a PDF-Chat App - Load and Processing Pdf file

- A **`VectorStoreToolkit`** object is created using the **`VectorStoreToolkit()`** constructor, passing in the **`vectorstore_info`** object created in the previous step.
- This converts the document store into a LangChain toolkit.



```python
# Load documents into vector database aka ChromaDB
store = Chroma.from_documents(pages, embeddings, collection_name='my_pdf')

# Create vectorstore info object
vectorstore_info = VectorStoreInfo(
    name="my_pdf",
    description=" A pdf file to answer your questions",
    vectorstore=store
)
# Convert the document store into a langchain toolkit
toolkit = VectorStoreToolkit(vectorstore_info=vectorstore_info)

# Add the toolkit to an end-to-end LC
agent_executor = create_vectorstore_agent(
    llm=llm,
    toolkit=toolkit,
    verbose=True
)
```

Building a PDF-Chat App - Load and Processing Pdf file

- The create_vectorstore_agent() function is called to create an end-to-end LangChain agent executor.
- It takes the OpenAI language model (llm) and the toolkit object as inputs.
- Additionally, verbose=True enables verbose mode for the agent executor.

```
# Load documents into vector database aka ChromaDB
store = Chroma.from_documents(pages, embeddings, collection_name='my_pdf')

# Create vectorstore info object
vectorstore_info = VectorStoreInfo(
    name="my_pdf",
    description=" A pdf file to answer your questions",
    vectorstore=store
)
# Convert the document store into a langchain toolkit
toolkit = VectorStoreToolkit(vectorstore_info=vectorstore_info)

# Add the toolkit to an end-to-end LC
agent_executor = create_vectorstore_agent(
    llm=llm,
    toolkit=toolkit,
    verbose=True
)
```

Building a PDF-Chat App - Prompt Handling

- The final part of the code will handle the input prompt and generate answers to users' questions.
- First, a text input box is created using st.text_input('Input your prompt here'). The string 'Input your prompt here' is displayed as a placeholder text inside the input box.’
- The code block if prompt: checks if the variable prompt (the user's input) is not empty, indicating that the user has entered a prompt and pressed Enter.

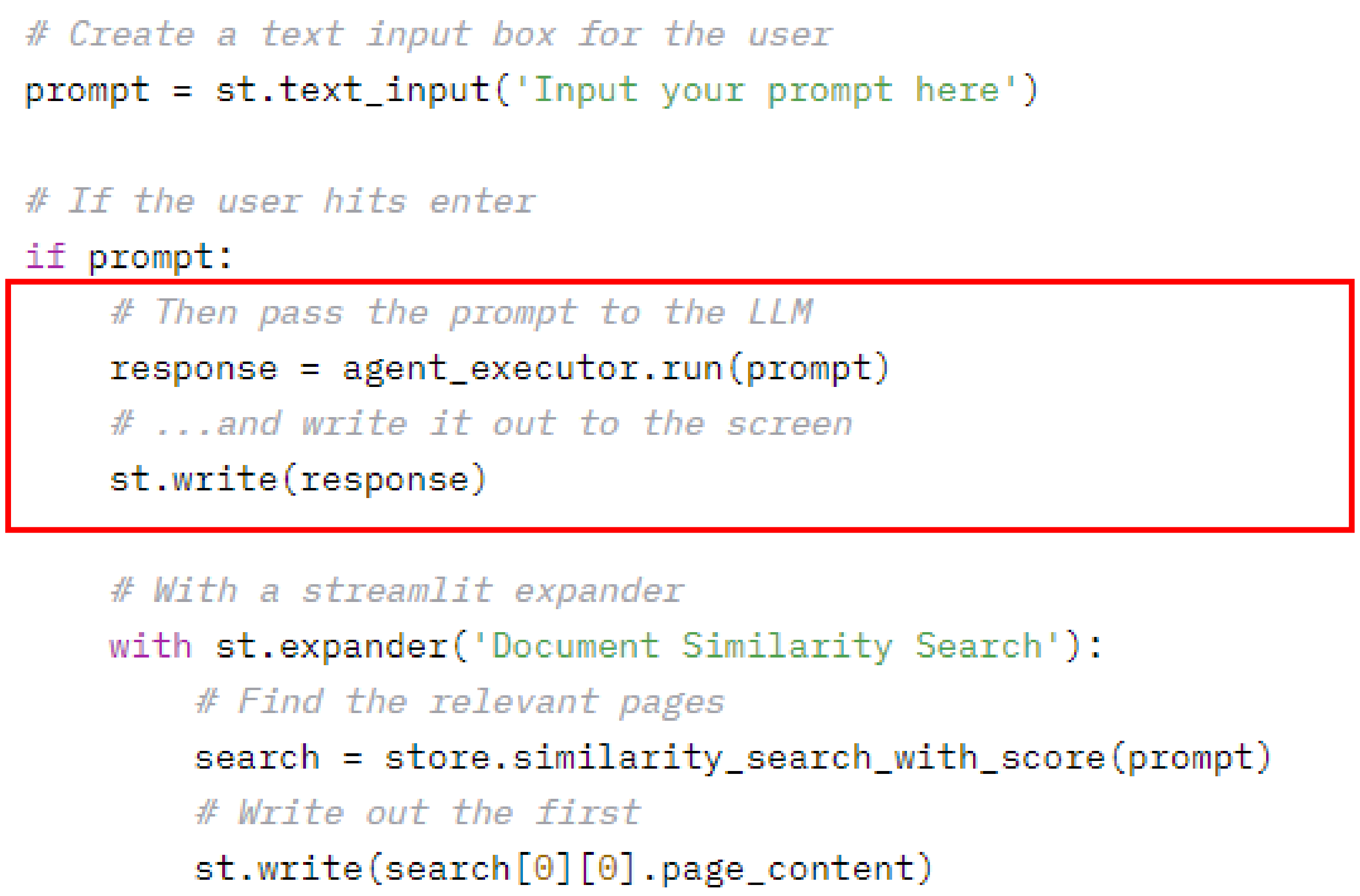
```
# Create a text input box for the user
prompt = st.text_input('Input your prompt here')
```

```
# If the user hits enter
if prompt:
    # Then pass the prompt to the LLM
    response = agent_executor.run(prompt)
    # ...and write it out to the screen
    st.write(response)

    # With a streamlit expander
    with st.expander('Document Similarity Search'):
        # Find the relevant pages
        search = store.similarity_search_with_score(prompt)
        # Write out the first
        st.write(search[0][0].page_content)
```

Building a PDF-Chat App - Prompt Handling

If the condition is true (i.e., the user has entered a prompt), the following actions are performed:

1. The prompt is passed to an "agent_executor" using agent_executor.run(prompt).
2. The response generated by the agent_executor is stored in the variable response.
3. The response is displayed on the screen using st.write(response).



```python
# Create a text input box for the user
prompt = st.text_input('Input your prompt here')

# If the user hits enter
if prompt:
    # Then pass the prompt to the LLM
    response = agent_executor.run(prompt)
    # ...and write it out to the screen
    st.write(response)

    # With a streamlit expander
    with st.expander('Document Similarity Search'):
        # Find the relevant pages
        search = store.similarity_search_with_score(prompt)
        # Write out the first
        st.write(search[0][0].page_content)
```

Building a PDF-Chat App - Prompt Handling

- The code block with st.expander ('Document Similarity Search'): creates a Streamlit expander with the header text 'Document Similarity Search'.
- Inside the expander, the following actions are performed:

1. The store.similarity_search_with_score (prompt) function is called to find relevant pages based on the input prompt.
2. The search result is stored in the variable search.
3. The content of the first page in the search result (retrieved with search[0][0].page_content) is displayed on the screen using st.write().

```python
# Create a text input box for the user
prompt = st.text_input('Input your prompt here')

# If the user hits enter
if prompt:
    # Then pass the prompt to the LLM
    response = agent_executor.run(prompt)
    # ...and write it out to the screen
    st.write(response)

    # With a streamlit expander
    with st.expander('Document Similarity Search'):
        # Find the relevant pages
        search = store.similarity_search_with_score(prompt)
        # Write out the first
        st.write(search[0][0].page_content)
```