Business AI Meeting Companion STT

Skills Network

Introduction

Consider you're attending a business meeting where all conversations are being captured by an advanced AI application. This application not only transcribes the discussions with high accuracy but also provides a concise summary of the meeting, emphasizing the key points and decisions made.

In our project, we'll use OpenAI's Whisper to transform speech into text. Next, we'll use IBM Watson's AI to summarize and find key points. We'll make an app with Hugging Face Gradio as the user interface.

Learning Objectives

After finishing this lab, you will able to:

- Create a Python script to generate text using a model from the Hugging Face Hub, identify some key parameters that influence the model's output, and have a basic understanding of how to switch between different LLM models.
- Use OpenAI's Whisper technology to convert lecture recordings into text, accurately.
- Implement IBM Watson's AI to effectively summarize the transcribed lectures and extract key points.
- Create an intuitive and user-friendly interface using Hugging Face Gradio, ensuring ease of use for students and educators.

Generated by DALLE-3

Preparing the environment

Let's start with setting up the environment by creating a Python virtual environment and installing the required libraries, using the following commands in the terminal:

```
pip3 install virtualenv
virtualenv my_env # create a virtual environment my_env
source my_env/bin/activate # activate my_env
```

Copied! Executed!

Then, install the required libraries in the environment (this will take time ☕☕):

```
# installing required libraries in my_env
pip install transformers==4.35.2 torch==2.1.1 gradio==4.44.0 langchain==0.0.343 ibm_watson_machine_learning==1.0.335 huggingface-hub==0.19.4
```

Copied! Executed!

Have a cup of coffee, it will take a few minutes.

```
       )  (
      (   ) )
       ) ( (
     _______)_
  .-'---------|
 ( C|/\/\/\/\/|
  '-./\/\/\/\/|
    '_________'
     '-------'
```

Copied!

We need to install `ffmpeg` to be able to work with audio files in python.

```
sudo apt update
```

Copied! Executed!

Then run:

```
sudo apt install ffmpeg -y
```

Copied! Executed!

> Whisper from OpenAI is available in github. Whisper's code and model weights are released under the MIT License. See LICENSE for further details.

Step 1: Speech-to-Text

Initially, we want to create a simple speech-to-text Python file using OpenAI Whisper.

You can test the sample audio file **Sample voice** **link to download.**

Create and open a Python file and call it `simple_speech2text.py` by clicking the link below:

Open **simple_speech2text.py** in IDE

Let's download the file first (you can do it manually, then drag and drop it into the file environment).

```python
import requests

# URL of the audio file to be downloaded
url = "https://cf-courses-data.s3.us.cloud-object-storage.appdomain.cloud/IBMSkillsNetwork-GPXX04C6EN/Testing%20speech%20to%20text.mp3"

# Send a GET request to the URL to download the file
response = requests.get(url)

# Define the local file path where the audio file will be saved
audio_file_path = "downloaded_audio.mp3"

# Check if the request was successful (status code 200)
if response.status_code == 200:
    # If successful, write the content to the specified local file path
    with open(audio_file_path, "wb") as file:
        file.write(response.content)
    print("File downloaded successfully")
else:
    # If the request failed, print an error message
    print("Failed to download the file")
```

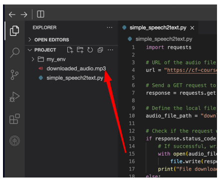
Copied!

Run the Python file to test it.

```
python3 simple_speech2text.py
```

Copied! Executed!

You should see the downloaded audio file in the file explorer.



Next, implement OpenAI Whisper for transcribing voice to speech.

You can override the previous code in the Python file.

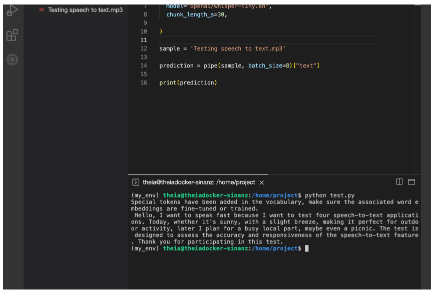
```
import torch
from transformers import pipeline

# Initialize the speech-to-text pipeline from Hugging Face Transformers
# This uses the "openai/whisper-tiny.en" model for automatic speech recognition (ASR)
# The `chunk_length_s` parameter specifies the chunk length in seconds for processing
pipe = pipeline(
  "automatic-speech-recognition",
  model="openai/whisper-tiny.en",
  chunk_length_s=30,
)

# Define the path to the audio file that needs to be transcribed
sample = 'downloaded_audio.mp3'

# Perform speech recognition on the audio file
# The `batch_size=8` parameter indicates how many chunks are processed at a time
# The result is stored in `prediction` with the key "text" containing the transcribed text
prediction = pipe(sample, batch_size=8)["text"]

# Print the transcribed text to the console
print(prediction)
```

Copied!

Run the Python file and you will get the output.

```
python3 simple_speech2text.py
```

Copied! Executed!



In the next step, we will utilize `Gradio` for creating interface for our app.

Gradio interface

Creating a simple demo

Through this project, we will create different LLM applications with Gradio interface. Let's get familiar with Gradio by creating a simple app:

Still in the `project` directory, create a Python file and name it `hello.py`.

Open `hello.py`, paste the following Python code and save the file.

```
import gradio as gr

def greet(name):
    return "Hello " + name + "!"

demo = gr.Interface(fn=greet, inputs="text", outputs="text")

demo.launch(server_name="0.0.0.0", server_port= 7860)
```

Copied!

The above code creates a **gradio.Interface** called `demo`. It wraps the `greet` function with a simple text-to-text user interface that you could interact with.

The **gradio.Interface** class is initialized with 3 required parameters:

- fn: the function to wrap a UI around
- inputs: which component(s) to use for the input (e.g. "text", "image" or "audio")
- outputs: which component(s) to use for the output (e.g. "text", "image" or "label")

The last line `demo.launch()` launches a server to serve our `demo`.

Launching the demo app

Now go back to the terminal and make sure that the `my_env` virtual environment name is displayed at the beginning of the line

Next, run the following command to execute the Python script.

```
python3 hello.py
```

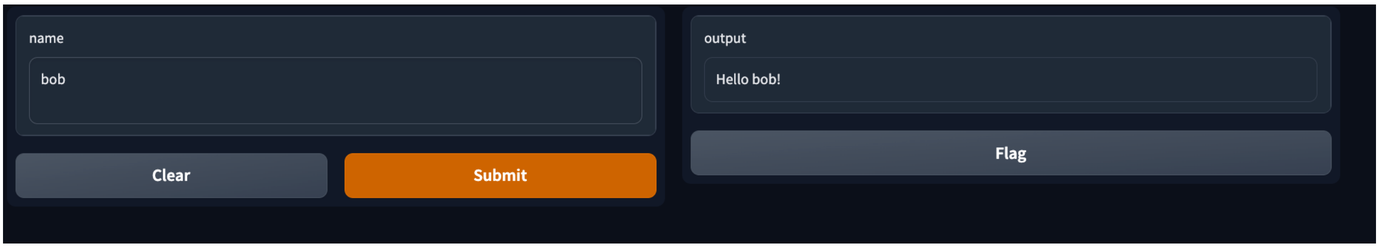
Copied! Executed!

As the Python code is served by a local host, click on the button below and you will be able to see the simple application we just created. Feel free to play around with the input and output of the web app!

Click here to see the application:

Web application

You should see the following, here we entered the name Bob:



If you finish playing with the app and want to exit, **press `Ctrl+c` in the terminal and close the application tab**.

If you wish to learn a little bit more about customization in Gradio, you are invited to take the guided project called **Bring your Machine Learning model to life with Gradio**. You can find it under **Courses & Projects** on cognitiveclass.ai!

For the rest of this project, we will use Gradio as an interface for LLM apps.

Step 2: Creating audio transcription app

Create a new python file `speech2text_app.py`

Open **speech2text_app.py** in IDE

Exercise: Complete the `transcript_audio` function.

From the step1: fill the missing parts in `transcript_audio` function.

```
import torch
from transformers import pipeline
import gradio as gr

# Function to transcribe audio using the OpenAI Whisper model
def transcript_audio(audio_file):
    # Initialize the speech recognition pipeline
    pipe = #-----> Fill here <----

    # Transcribe the audio file and return the result
    result = #-----> Fill here <----
    return result

# Set up Gradio interface
audio_input = gr.Audio(sources="upload", type="filepath")  # Audio input
output_text = gr.Textbox()  # Text output

# Create the Gradio interface with the function, inputs, and outputs
iface = gr.Interface(fn=transcript_audio,
                     inputs=audio_input, outputs=output_text,
                     title="Audio Transcription App",
                     description="Upload the audio file")

# Launch the Gradio app
iface.launch(server_name="0.0.0.0", server_port=7860)
```

Copied!

▾ Click here for the answer

```
import torch
from transformers import pipeline
import gradio as gr

# Function to transcribe audio using the OpenAI Whisper model
def transcript_audio(audio_file):
    # Initialize the speech recognition pipeline
    pipe = pipeline(
        "automatic-speech-recognition",
        model="openai/whisper-tiny.en",
        chunk_length_s=30,
    )
    # Transcribe the audio file and return the result
    result = pipe(audio_file, batch_size=8)["text"]
    return result

# Set up Gradio interface
audio_input = gr.Audio(sources="upload", type="filepath")  # Audio input
output_text = gr.Textbox()  # Text output

# Create the Gradio interface with the function, inputs, and outputs
iface = gr.Interface(fn=transcript_audio,
                     inputs=audio_input, outputs=output_text,
                     title="Audio Transcription App",
                     description="Upload the audio file")

# Launch the Gradio app
iface.launch(server_name="0.0.0.0", server_port=7860)
```

Copied!

Then, run your app:

```
python3 speech2text_app.py
```

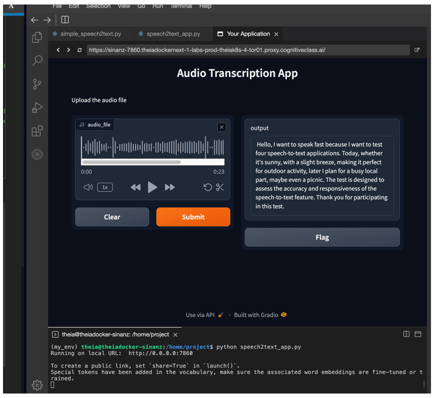
Copied! Executed!

And start the app:

Web application

You can download the sample audio file we've provided by right-clicking on it in the file explorer and selecting "Download." Once downloaded, you can upload this file to the app. Alternatively, feel free to choose and upload any MP3 audio file from your local computer.

The result will be:



Press Ctrl + C to stop the application.

Step 3: Integrating LLM: Using Llama 3 in WatsonX as LLM

Running simple LLM

Let's start by generating text with LLMs. Create a Python file and name it `simple_llm.py`. You can proceed by clicking the link below or by referencing the accompanying image.

Open **simple_llm.py** in IDE

In case, you want to use Llama 3 as an LLM instance, you can follow the instructions below:

> IBM WatsonX utilizes various language models, including Llama 3 by Meta, which is currently the strongest open-source language model.

Here's how the code works:

1. **Setting up credentials**: The credentials needed to access IBM's services are pre-arranged by the Skills Network team, so you don't have to worry about setting them up yourself.
2. **Specifying parameters**: The code then defines specific parameters for the language model. 'MAX_NEW_TOKENS' sets the limit on the number of words the model can generate in one go. 'TEMPERATURE' adjusts how creative or predictable the generated text is.
3. **Setting up Llama 3 model**: Next, the LLAMA3 model is set up using a model ID, the provided credentials, chosen parameters, and a project ID.
4. **Creating an object for Llama 3**: The code creates an object named llm, which is used to interact with the Llama 3 model. A model object, LLAMA3_model, is created using the Model class, which is initialized with a specific model ID, credentials, parameters, and project ID. Then, an instance of WatsonxLLM is created with LLAMA3_model as an argument, initializing the language model hub llm object.
5. **Generating and printing response**: Finally, 'llm' is used to generate a response to the question, "How to read a book effectively?" The response is then printed out.

```python
from ibm_watson_machine_learning.foundation_models import Model
from ibm_watson_machine_learning.foundation_models.extensions.langchain import WatsonxLLM
from ibm_watson_machine_learning.metanames import GenTextParamsMetaNames as GenParams

my_credentials = {
    "url"    : "https://us-south.ml.cloud.ibm.com"
}

params = {
        GenParams.MAX_NEW_TOKENS: 800, # The maximum number of tokens that the model can generate in a single run.
        GenParams.TEMPERATURE: 0.1,   # A parameter that controls the randomness of the token generation. A lower value makes the generation more deterministic, while a higher value introduces more randomness.
    }

LLAMA2_model = Model(
        model_id= 'meta-llama/llama-3-8b-instruct',
        credentials=my_credentials,
        params=params,
        project_id="skills-network",
        )

llm = WatsonxLLM(LLAMA2_model)

print(llm("How to read a book effectively?"))
```

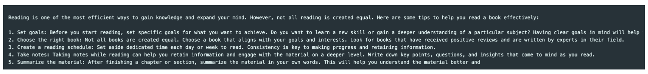
Copied!

You can then run this script in the terminal using the following command:

```
python3 simple_llm.py
```

Copied! Executed!

Upon running the script, you should see the generated text in your terminal, as shown below:


```
Reading is one of the most efficient ways to gain knowledge and expand your mind. However, not all reading is created equal. Here are some tips to help you read a book effectively:

1. Set goals: Before you start reading, set specific goals for what you want to achieve. Do you want to learn a new skill or gain a deeper understanding of a particular subject? Having clear goals in mind will help
2. Choose the right book: Not all books are created equal. Choose a book that aligns with your goals and interests. Look for books that have received positive reviews and are written by experts in their field.
3. Create a reading schedule: Set aside dedicated time each day or week to read. Consistency is key to making progress and retaining information.
4. Take notes: Taking notes while reading can help you retain information and engage with the material on a deeper level. Write down key points, questions, and insights that come to mind as you read.
5. Summarize the material: After finishing a chapter or section, summarize the material in your own words. This will help you understand the material better and
```

You can see how watsonx Llama 2 provides a good answer.

Step 4: Put them all together

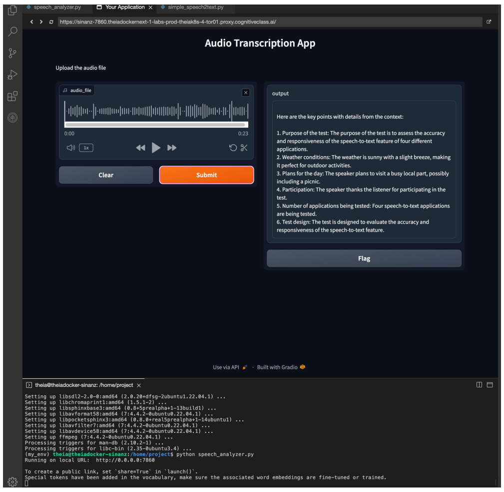
Create a new Python file and call it `speech_analyzer.py`

Open **speech_analyzer.py** in IDE

In this exercise, we'll set up a language model (LLM) instance, which could be IBM WatsonxLLM, HuggingFaceHub, or an OpenAI model. Then, we'll establish a prompt template. These templates are structured guides to generate prompts for language models, aiding in output organization (more info in langchain prompt template.

Next, we'll develop a transcription function that employs the OpenAI Whisper model to convert speech-to-text. This function takes an audio file uploaded through a Gradio app interface (preferably in .mp3 format). The transcribed text is then fed into an LLMChain, which integrates the text with the prompt template and forwards it to the chosen LLM. The final output from the LLM is then displayed in the Gradio app's output textbox.

The output should look:



Notice how the LLM corrected a minor mistake made by the speech-to-text model, resulting in a coherent and accurate output.

Exercise: Fill the missing parts:

```
import torch
import os
import gradio as gr
#from langchain.llms import OpenAI
from langchain.llms import HuggingFaceHub
from transformers import pipeline
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain
from ibm_watson_machine_learning.foundation_models.extensions.langchain import WatsonxLLM
from ibm_watson_machine_learning.foundation_models.utils.enums import DecodingMethods
from ibm_watson_machine_learning.metanames import GenTextParamsMetaNames as GenParams
from ibm_watson_machine_learning.foundation_models import Model

#######------------- LLM-------------####

# initiate LLM instance, this can be IBM WatsonX, huggingface, or OpenAI instance

llm = ###---> write your code here

#######------------- Prompt Template-------------####

# This template is structured based on LLAMA2. If you are using other LLMs, feel free to remove the tags
temp = """
<s><<SYS>>
List the key points with details from the context:
[INST] The context : {context} [/INST]
<</SYS>>
"""
# here is the simplified version of the prompt template
# temp = """
# List the key points with details from the context:
# The context : {context}
# """

pt = PromptTemplate(
    input_variables=["context"],
    template= temp)

prompt_to_LLAMA2 = LLMChain(llm=llm, prompt=pt)

#######------------- Speech2text-------------####

def transcript_audio(audio_file):
    # Initialize the speech recognition pipeline

    pipe = #------> write the code here

    # Transcribe the audio file and return the result
    transcript_txt = pipe(audio_file, batch_size=8)["text"]
    # run the chain to merge transcript text with the template and send it to the LLM
    result = prompt_to_LLAMA2.run(transcript_txt)

    return result

#######------------- Gradio-------------####

audio_input = gr.Audio(sources="upload", type="filepath")
output_text = gr.Textbox()

# Create the Gradio interface with the function, inputs, and outputs
iface = #---> write code here

iface.launch(server_name="0.0.0.0", server_port=7860)
```

Copied!

▾ Click here for the answer

```python
import torch
import os
import gradio as gr

#from langchain.llms import OpenAI
from langchain.llms import HuggingFaceHub

from transformers import pipeline
from langchain.prompts import PromptTemplate
from langchain.chains import LLMChain

from ibm_watson_machine_learning.foundation_models import Model
from ibm_watson_machine_learning.foundation_models.extensions.langchain import WatsonxLLM
from ibm_watson_machine_learning.metanames import GenTextParamsMetaNames as GenParams

my_credentials = {
    "url"    : "https://us-south.ml.cloud.ibm.com"
}
params = {
        GenParams.MAX_NEW_TOKENS: 800, # The maximum number of tokens that the model can generate in a single run.
        GenParams.TEMPERATURE: 0.1,   # A parameter that controls the randomness of the token generation. A lower value makes the generation more deterministic, while a higher value introduces more randomness.
    }

LLAMA2_model = Model(
        model_id= 'meta-llama/llama-3-8b-instruct',
        credentials=my_credentials,
        params=params,
        project_id="skills-network",
        )

llm = WatsonxLLM(LLAMA2_model)

#######------------- Prompt Template-------------####

temp = """
<s><<SYS>>
List the key points with details from the context:
[INST] The context : {context} [/INST]
<</SYS>>
"""

pt = PromptTemplate(
    input_variables=["context"],
    template= temp)

prompt_to_LLAMA2 = LLMChain(llm=llm, prompt=pt)

#######------------- Speech2text-------------####

def transcript_audio(audio_file):
    # Initialize the speech recognition pipeline
    pipe = pipeline(
        "automatic-speech-recognition",
        model="openai/whisper-tiny.en",
        chunk_length_s=30,
    )
    # Transcribe the audio file and return the result
    transcript_txt = pipe(audio_file, batch_size=8)["text"]
    result = prompt_to_LLAMA2.run(transcript_txt)

    return result

#######------------- Gradio-------------####

audio_input = gr.Audio(sources="upload", type="filepath")
output_text = gr.Textbox()

iface = gr.Interface(fn= transcript_audio,
                    inputs= audio_input, outputs= output_text,
                    title= "Audio Transcription App",
                    description= "Upload the audio file")

iface.launch(server_name="0.0.0.0", server_port=7860)
```

Copied!

Run your code:

```
python3 speech_analyzer.py
```

Copied! Executed!

If there is no error, run the web app:

Web application

Conclusion

Congratulations on completing this project! You have now laid a solid foundation for leveraging powerful Language Models (LLMs) for speech-to-text generation tasks. Here's a quick recap of what you've accomplished:

- Text generation with LLM: You've created a Python script to generate text using a model from the Hugging Face Hub, learned about some key parameters that influence the model's output, and have a basic understanding of how to switch between different LLM models.
- Speech-to-Text conversion: Utilize OpenAI's Whisper technology to convert lecture recordings into text, accurately.
- Content summarization: Implement IBM Watson's AI to effectively summarize the transcribed lectures and extract key points.
- User interface development: Create an intuitive and user-friendly interface using Hugging Face Gradio, ensuring ease of use for students and educators.

Author(s)

Sina Nazeri