

Real-time Speech to Text to Speech: Building Your AI-Based Alexa

Special Topics: Generative AI-Driven Intelligent Apps
Development

Ademilton Marcelo da Cruz Nunes (19679)

Links

Github:

<https://github.com/ademiltonnunes/Machine-Learning/tree/main/ChatGPT/Real-time%20Speech%20to%20Text%20to%20Speech>

Table of Content

- Introduction
- Introduction - Whisper
- Introduction - Google Text to Speech (GTTS)
- Introduction - Click Library
- Implementation Steps
- Implementation Steps - Recording the audio
- Implementation Steps - Transcribing the audio
- Implementation Steps - Submitting prompt to ChatGPT
- Implementation Steps - Replying ChatGPT response
- Recording the audio
- Transcribing the audio
- Submitting prompt to ChatGPT
- Replying ChatGPT response
- Conclusion

Introduction

This project aims to implement a AI-Based “Alexa”, which is the possibility of speaking with ChatGPT with voice using Whisper and Google Text to Speech (GTTS).

The system was built in Python and used the ChatGPT OpenAI text-davinci-002 model.

Introduction - Whisper

Whisper is an ASR system (automatic speech recognition) and a speech recognition model trained by OpenAI on 680,000 hours of multilingual and multitask data collected from the web.

Whisper main function is to transcribe voice-audio into text. It has five models sizes, four of which have English-only versions. Each whisper model offers a tradeoff between speed and accuracy. The performance varies depending on the language.

Whisper has two modes to integrate:

- Batch: Transcribes recorded audio into text
- Real-time: Transcribes real-time audio into text.

For this project, I used Real-time integration mode

Introduction - Google Text to Speech (GTTS)

The gTTS (Google Text-to-Speech) library is a powerful tool for converting text to speech in Python. With gTTS, you can easily add speech functionality to your Python scripts, allowing you to create audio files from any text.

Introduction - Click Library

Click is a Python library for creating command line interfaces in a composable way with as little code as necessary. It's highly configurable but comes with sensible defaults out of the box.

I used click library in this project to enable the assignment of parameter values via the command line and also to set the default values of the attributes.

Implementation Steps

The implementation consists of 4 main steps:

1. Recording the audio
2. Transcribing the audio
3. Submitting prompt to ChatGPT
4. Replying ChatGPT response

Implementation Steps - Recording the audio

This process starts with recording user's voice. The voice is captured by user's microphone. A wake word must be spoken, so that the system starts recording the audio. Some other parameters can be configured so that the recorded audio has more qualities, including:

- Energy: configures the magnitude of the sound or signal in a specific segment audio.
- Pause: Configures how long there should be a pause between words, if the pause is longer than the configured value, the audio is submitted to the next steps.
- Dynamic: Configures changes in loudness or intensity over time.

This audio will be saved in a audio query which will be the input to the step of "Transcribing the audio".

Implementation Steps - Transcribing the audio

Having the audio query from the previous step, it will be saved and transcribed to a text format. Whisper will transcribe audio to text in this step. At the time of transcription, some words may be disregarded and excluded, such as:

- Wake word: Wake word does not need to be in the audio transcription.
- Stop word: are words that have no semantic meaning but can be present in the audio, an example is the word "like", some people with language addiction can say this word all the time when they are speaking.

Text format will be the input of the next step.

Implementation Steps - Submitting prompt to ChatGPT

Getting the text transcribed from the previous step, the text will be the prompt submitted to ChatGPT. ChatGPT response, which also is in text format, will be the input of the next step.

Implementation Steps - Replying ChatGPT response

Getting the ChatGPT response from the previous step, the response text will be transcribed from text to audio using GTTS. This audio will be returned to the user.

ChatGPT response can answer user prompt or it may have default responses if the user's audio doesn't make sense.^a

Default response are:

- "I'm sorry, I don't know the answer to that"
- "I'm not sure I understand"
- "I'm not sure I can answer that",
- "Please repeat the question in a different way"

Recording the audio

The first thing I implemented was the click library, with the audio recording configuration parameters. There, default parameters for:

- Language
- Energy and pause time
- Dynamic
- Wake up word
- Verbose: If it will respond or not case user audio is too verbose

```
@click.command()
@click.option("--model", default="base", help="Model to use", type=click.Choice(["tiny", "base", "small", "medium", "large"]))
@click.option("--english", default=False, help="Whether to use the English model", is_flag=True, type=bool)
@click.option("--energy", default=300, help="Energy level for the mic to detect", type=int)
@click.option("--pause", default=0.8, help="Pause time before entry ends", type=float)
@click.option("--dynamic_energy", default=False, is_flag=True, help="Flag to enable dynamic energy", type=bool)
@click.option("--wake_word", default="hey computer", help="Wake word to listen for", type=str)
@click.option("--verbose", default=False, help="Whether to print verbose output", is_flag=True, type=bool)
```

Ln 105, Col 60 Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit

00:40 07/11/2023

Recording the audio

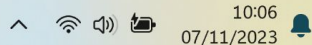
The `record_audio` method continuously records audio from the user microphone using the SpeechRecognition library (`sr`). The recorded audio is then converted to a format suitable for further processing using PyTorch. The processed audio data is then put into the `audio_queue` for further processing by other parts of the system.

```
def record_audio(audio_queue, energy, pause, dynamic_energy) -> NoReturn:
    r = sr.Recognizer()
    r.energy_threshold = energy
    r.pause_threshold = pause
    r.dynamic_energy_threshold = dynamic_energy

    with sr.Microphone(sample_rate=16000) as source:
        print("Listening...")
        i = 0
        while True:
            audio = r.listen(source)
            torch_audio = torch.from_numpy(np.frombuffer(audio.get_raw_data(), np.int16).flatten().astype(np.float32) / 32768.0)
            audio_data = torch_audio
            audio_queue.put_nowait(audio_data)
            i += 1
```

158 0 0 Live Share

Ln 46, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit



Transcribing the audio

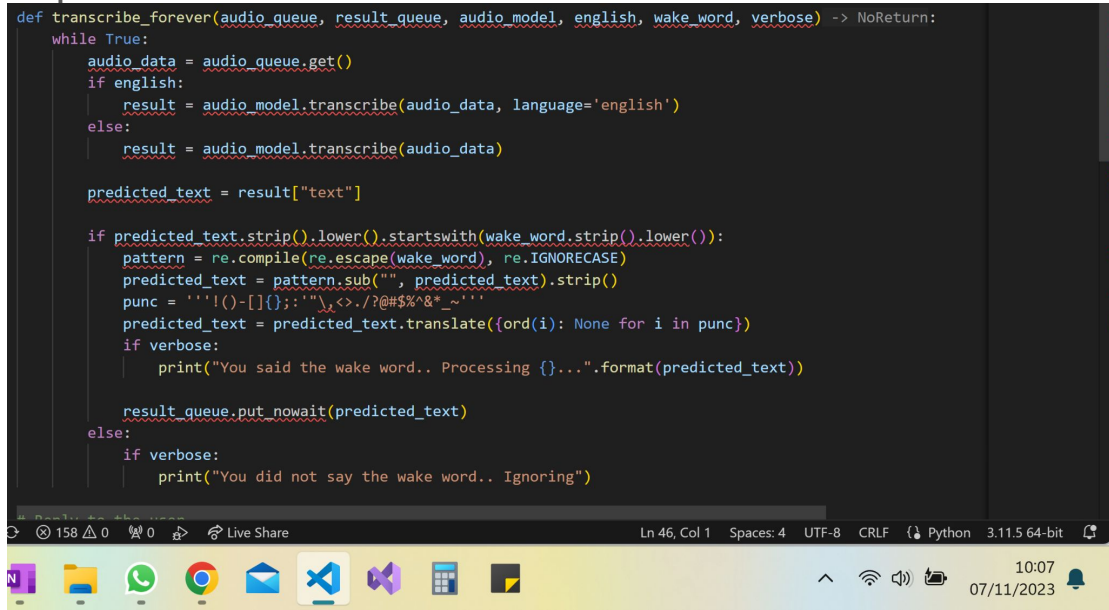
The method `transcribe_forever` continuously transcribes audio data obtained from a queue (`audio_queue`) to text. The transcription results are then processed to identify a wake word, and if the wake word is detected. The relevant information is extracted and put into another queue in text.

```
def transcribe_forever(audio_queue, result_queue, audio_model, english, wake_word, verbose) -> NoReturn:
    while True:
        audio_data = audio_queue.get()
        if english:
            result = audio_model.transcribe(audio_data, language='english')
        else:
            result = audio_model.transcribe(audio_data)

        predicted_text = result["text"]

        if predicted_text.strip().lower().startswith(wake_word.strip().lower()):
            pattern = re.compile(re.escape(wake_word), re.IGNORECASE)
            predicted_text = pattern.sub("", predicted_text).strip()
            punc = ' '!()-[]{};:_"\<>./?@#%&*~_''
            predicted_text = predicted_text.translate({ord(i): None for i in punc})
            if verbose:
                print("You said the wake word.. Processing {}...".format(predicted_text))

            result_queue.put_nowait(predicted_text)
        else:
            if verbose:
                print("You did not say the wake word.. Ignoring")
```



Submitting prompt to ChatGPT

With the extracted text we will submit it to LLM ChatGPT to answer customer prompt.

```
def reply(result_queue, verbose) -> NoReturn:
    while True:
        question = result_queue.get()
        # We use the following format for the prompt: "Q: ?\nA:"
        prompt = "Q: {}?\nA:".format(question)

        data = openai.Completion.create(
            model="text-davinci-002",
            prompt=prompt,
            temperature=0.5,
            max_tokens=100,
            n=1,
            stop=["\n"]
        )
```

158 0 0 0 Live Share Ln 98, Col 27 Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit

Replying ChatGPT response

Having the GPT answer, I used GTTS to transcribe ChatGPT response to audio and play it, also some default choices are given case any exception happens:

```
try:
    answer = data["choices"][0]["text"]
    mp3_obj = gTTS(text=answer, lang="en", slow=False)
except Exception as e:
    choices = [
        "I'm sorry, I don't know the answer to that",
        "I'm not sure I understand",
        "I'm not sure I can answer that",
        "Please repeat the question in a different way"
    ]
    mp3_obj = gTTS(text=choices[np.random.randint(0, len(choices))], lang="en", slow=False)
    if verbose:
        print(e)

# In both cases, we play the audio
mp3_obj.save("reply.mp3")
reply_audio = AudioSegment.from_mp3("reply.mp3")
play(reply_audio)
```

58 0 0 0 Live Share Ln 119, Col 1 Spaces: 4 UTF-8 CRLF Python 3.11.5 64-bit

10:08 07/11/2023

Conclusion

This project demonstrated how to implement a AI-Based “Alexa”, which is the possibility of speaking with ChatGPT with voice using Whisper and Google Text to Speech (GTTS).

I applied techniques and showed solid examples of how to Recording the audio, Transcribing the audio, Submitting prompt to ChatGPT, and Replying ChatGPT response.