B.TECH PROJECT - 1

Project Report

SPEECH TRANSCRIPTION VERIFICATION

Team 1
Shreeya Singh - 2020102011
Srujana Vanka - 2020102005

Github Link

Table of Contents

Introduction

The process of converting speech into text is a crucial component in many applications such as automatic captioning, audio search, and speech recognition. Speech transcription can be a time-consuming and labor-intensive task, especially for large audio files. Moreover, the accuracy of the transcription is vital for the effectiveness of the application. Inaccuracies in the transcription can lead to misunderstandings and misinterpretations. Therefore, there is a need for an efficient and reliable system for speech transcription verification that can ensure the accuracy of the transcription while minimizing the effort required by transcribers.

Our project aims to develop such a system by creating an interface that allows transcribers to verify and correct the transcription of smaller audio segments. The system will ensure that each audio segment is reviewed only onc. This will ensure that the transcription process is more manageable and efficient, while also improving the accuracy of the transcription.

In this report, we will discuss the design and implementation of our system, the challenges we faced, and the results of our testing. We will also discuss the potential applications and future improvements of our system.

Overview

Our project aims to develop a system for verifying speech transcriptions by splitting a large audio file into smaller segments, converting each segment into text using the speech-to-text API provided, and presenting it to the user for verification and correction.

- Develop an interface that displays the transcript text and allows the user to play the corresponding audio chunk and make edits to the transcript.
- The user will have the option save or discard the transcription.
- The system will ensure that each audio chunk is reviewed only once i.e should not appear again once the application is restarted. The user can then move on to the next chunk once he has verified the transcription
- Overall, our project aims to verify the efficiency of speech transcriptions and make the process more manageable for transcribers through an easy to use interface.

Objective

To develop a system to verify speech transcriptions by splitting a large audio file into smaller chunks, transcribing each chunk, and presenting them to the user for verification and correction. Enabling them to either "save" or "discard" the transcriptions and move to the "next" one.

There are 3 main components:

1. Audio Splitting
2. Speech-to-text conversion
3. Transcription verification Interface

Implementation

1. Audio Splitting

Audio splitting is the process of dividing a large audio file into smaller segments or chunks. Audio splitting is done for several reasons:

- **Easier processing**: By splitting into smaller chunks, it becomes easier to process each chunk separately and in parallel, which can speed up the overall processing time.
- **Improved accuracy**: It improves the accuracy of speech recognition and transcription and makes it easier to isolate and analyze each segment separately, which can lead to more accurate transcriptions.
- **User interface**: Makes it more manageable for the user to review and verify the transcriptions for smaller chunks.

Overall, splitting a large audio file into smaller chunks is done to make processing more efficient, improve accuracy, and make the transcription review process more manageable for users.

Methodology

- Audio splitting is done by splitting an audio file into smaller segments based on silence using the **Pydub** and **WebRTC Voice Activity Detection (VAD) libraries.**

```
from pydub import AudioSegment
from pydub.silence import split_on_silence
import webrtcvad
```

- The WebRTC Voice Activity Detector (VAD) is used to filter out non-speech sections of audio to improve the accuracy of the silence detection.
- The input audio file path and the output directory path are taken as inputs. This audio file is read and we create frames where the VAD object is used to determine which frames contain speech and which contain silence, and collect consecutive speech frames into a list.
- When a sufficient number of consecutive speech frames have been collected, the collected frames are presented as a single audio segment.
- These chunks are saved in the Public folder `"./frontend/speech-transcription-app/public/Original data/audio_chunks"`
- as they are created to be further processed for speech-to-text conversion and be displayed on the interface.
- This task is done by the python script `audio-split.py` in the `main` folder.

2. Speech-to-text conversion

The process of converting spoken words into written text is called Speech-to-Text conversion. Each of these audio chunks created is further processed to obtain transcripts.

Methodology

- Each audio chunk is converted into text transcripts using the **Speech-to-Text API** provided.
- The API is hosted at: https://asr.iiit.ac.in/ssmtapi/
- Each audio chunk is sent to the Speech-to-Text API for transcription, and saves the output in text files.
- A POST request is made to the API with the audio file as input and the response from the API is in JSON format, which is loaded into a Python dictionary using the json.loads() method.

```
-   output = subprocess.check_output("curl -k -X 'POST'
    'https://asr.iiit.ac.in/ssmtapi//'    -H 'accept: application/json'
    -H 'Content-Type: multipart/form-data'    -F
    'uploaded_file=@./frontend/speech-transcription-app/public/Original
    data/audio_chunks/'chunk"+str("{:04d}".format(i))+".wav';type=audio/
    x-wav'    -F 'lang=eng' ", shell=True)
-      dict = json.loads(output.decode('utf-8'))
```

- The text transcription is stored in a list of dictionaries under the "transcript" key in the dictionary. We then extract the text from each dictionary and concatenate them to form the complete transcription of the audio chunk.
- These transcripts are then saved in the path `“./frontend/speech-transcription-app/public/Original data/transcripts”`
- This task is done by the python script `transcription.py` in the `main` folder.

3. Transcription Verification Interface

Backend Server

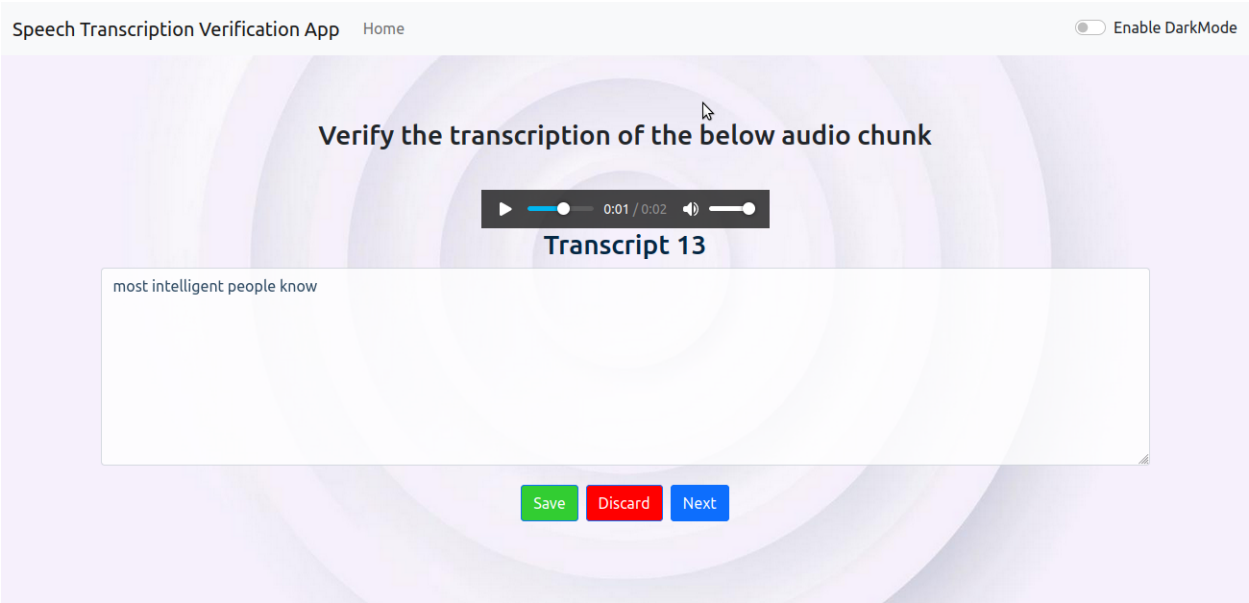
First task is setting up the server using **Node.js** and **Express.js** framework. The `` `child_process` `` module is used to execute two Python scripts called `` `audio-split.py` `` and `` `transcription.py` `` using `` `exec()` `` function.

Once the Python scripts have finished running, the server is started and listens for incoming requests on **port 5000** using the `` `body-parser` `` middleware to parse incoming requests with JSON payloads. The `` `cors` `` middleware is also added to allow cross-origin requests from other domains. The server is started on port 5000 using the `` `listen()` `` function, and a message is logged to the console indicating that the server is running.

```
// Start the server after both scripts have finished
  const PORT = 5000;
  app.listen(PORT, () => {
    console.log(`Server running on port ${PORT}`);
  });
```

This design allows the backend server to **handle the audio splitting and transcription processes separately from the frontend interface**, making the application more scalable and efficient. Additionally, using Node.js and Python together allows for greater flexibility and functionality in the overall system design.

Interface



System Design

The interface involves multiple components working together to provide a seamless experience for the user.

1. The transcript text is presented to the user along with the option to play the corresponding audio chunk.

2. The user can make edits to the transcript if required, and has the option to either save or discard the transcription.

3. The interface ensures that the transcript number is saved even on application restarts and that each audio chunk is reviewed only once, the system moves on to the next chunk once the user clicks next.

The interface is built using React, a popular frontend library for building user interfaces. The user interface is simple and intuitive, with controls for playing the audio, editing the transcript, and saving or discarding the transcription.

Methodology

1. **The transcript text is presented to the user along with the option to play the corresponding audio chunk.**

The transcript text is presented to the user in a textarea element, which is initially set to display "Loading...".

```
const [text, setText] = useState('Loading...');
```

The text displayed in the textarea is fetched from a text file in the `"./Original data/transcripts"` folder using Axios and the current transcript number state.

In addition, if the current transcript corresponds to an audio file, a HTML5 audio element is displayed below the textarea element. The audio file is fetched from a wav file in the `"Original data/audio_chunks"` folder using Axios and the current transcript number state. The audio is played and paused using the standard controls provided by the audio element.

2. **The user can make edits to the transcript if required, and has the option to either save or discard the transcription.**

The user can make edits to the transcript by typing directly into the `` `textarea` `` element. Any changes the user makes will be reflected in the `` `text` `` state variable, which is then displayed in the `` `textarea` `` element through the `` `value` `` attribute.

The `` **`onChange`** `` event listener is responsible for updating the `text` state variable whenever the user types in the `textarea`. When the `value` of the `textarea` changes, the `` `onChange` `` event is triggered, which updates the `text` state variable to the current value of the `textarea`.

Thus, this allows the user to make edits to the transcript and have those changes reflected in the UI.

There are also three buttons displayed below the textarea and audio elements: **`"Save", "Discard",`** and **`"Next"`**. All three buttons are initially disabled when the transcript text is set to "Loading...". The "Save" and "Discard" buttons send an **HTTP POST** request to a local server using Axios, with the current transcript text and transcript number as data.

```
const handleSave = () => {

      axios.post('http://localhost:5000/save-text', {
```

```
            text: text,
            transcriptNumber: transcriptNumber // Add this line
        })
            .then(response => {
                console.log(response.data);
            })
            .catch(error => {
                console.log(error);
            });
    };
```

When the user clicks the "Save" button, the `**handleSave**` function is called. This function makes a POST request to the `**http://localhost:5000/save-text**` endpoint using the Axios library, passing in two parameters: the `text` of the transcript, and the `transcriptNumber`.If the request is successful, the response data is logged to the console. If there is an error, the error is also logged to the console.

The discard button works similarly. In summary, the user has the option to save or discard changes made to a transcript using these functions, which make requests to the server with the updated transcript data and transcript identifier.

3. **The interface ensures that the transcript number is saved even on application restarts and that each audio chunk is reviewed only once, the system moves on to the next chunk once the user clicks next.**

The code uses the `**useState**` hook to create a state variable called `transcriptNumber`, which is initially set to the value retrieved from `**localStorage**` or 1 if there is no stored value. This variable keeps track of the transcript number that the user is currently reviewing.

```
    // Using localStorage API to store the last transcript number and
retrieve it when the application starts again
    const [transcriptNumber, setTranscriptNumber] =
useState(parseInt(localStorage.getItem('transcriptNumber')) || 1);
```

When the user clicks the "Next" button, the `**handleNext**` function is called. First, the code updates the `localStorage` to store the next transcript number (`transcriptNumber + 1`). Then, the `setTranscriptNumber` function is called to update the `transcriptNumber` state variable, which triggers a re-render of the component.

```
  // Next button
   const handleNext = async () => {
       localStorage.setItem('transcriptNumber', transcriptNumber + 1);
       setTranscriptNumber((prevState) => prevState + 1);

       // Fetch the new transcript text
       try {
           const response = await axios.get(`./Original
data/transcripts/transcript${(transcriptNumber + 1).toString().padStart(4,
'0')}.txt`);
           setText(response.data);
       } catch (error) {
           console.error(error);
       }
   };
```

After updating the transcript number, the code fetches the text for the next transcript using the `axios` library, based on the incremented `transcriptNumber`. The `try...catch` block is used to catch any errors that might occur during the network request. Once the text for the next transcript is retrieved successfully, it is set as the value of the `text` state variable using the `setText` function.

By incrementing the `transcriptNumber` and fetching the next transcript in this way, the code ensures that each audio chunk is reviewed only once, and the system keeps track of the transcript number and moves on to the next chunk once the user clicks the "Next" button.

Tech Stack

- Built with: Python3, React, Javascript
- Frontend UI built using: Creat-React-App
- Backend server built using: Node.js and Server.js
- Python scripts used for Audio Chunking and Transcripts Generation

Modules used:

1. Audio Splitting:
    - Pydub
    - WebRTC Voice Activity Detector (VAD)

2. Speech-to-text conversion:
    - Speech-to-text API provided - Click here

3. Transcription verification:
    - Proptypes
    - Axios
    - LocalStorage API
    - Fetch API

Major Challenges

1. Integration with audio recording and playback: The app needs to be able to record and play back audio files to enable the transcription process. Integrating the app with audio recording and playback can be a complex task, and it requires a good understanding of the audio file formats and APIs.

2. Handling large audio files: Audio files can be quite large, and processing them can be resource-intensive. The app needs to be optimized to handle large audio files efficiently, without consuming too much memory or CPU resources.

3. User interface design: A user-friendly and intuitive interface is essential for the success of the app. Designing a user interface that is easy to use, functional, and visually appealing can be challenging, and it requires a good understanding of the target audience and their needs.

4. Cross-browser compatibility: The app needs to work seamlessly across different web browsers, operating systems, and devices. Ensuring cross-browser compatibility can be a challenging task, and it requires extensive testing and debugging.

Future Scope

1. **Enhancing Navigation**:

- Incorporating a previous button to allow users to navigate backward in the transcripts

- Adding a reset button to enable users to go back to transcript 1

- Allowing users to set the transcript number to any value and navigate to it

2. **Multiple Audio Files:**

- Accommodating multiple audio files and allowing users to choose which audio file to work on

- Enabling users to upload their own audio files for transcription and verification

3. **Language Support:**

- Supporting the application for different languages to enable transcription and verification in a wider range of contexts

- Adding language detection functionality to identify the language of the audio automatically

4. **Exporting Transcripts:**

- Adding a feature for downloading transcripts in different formats to enable easy sharing and integration with other tools

5. **Verification:**

- Verifying the transcriber by implementing checks to determine whether the transcriber has actually listened to the audio or not

- Providing feedback to the user on their accuracy to improve future transcription and verification efforts

Acknowledgements

We would like to express my sincere gratitude towards our project guide, Mr. Anil Kumar Vupalla, for his invaluable guidance, encouragement, and support throughout the project. His insights, suggestions, and constant motivation helped us to complete this project successfully.

We would also like to extend my heartfelt thanks to our project mentor, Mr. Kowshik Motepalli, who went above and beyond to answer our doubts, provide guidance, and help us whenever we faced any difficulties. Their presence and support have been instrumental in shaping our project.

We would like to thank the panel members, Ms. Parameswari Krishnamurthy and Mr. Karthik Vaidhyanathan, for evaluating our project during BTP-1. Their feedback and suggestions were invaluable and will helps us in improving our project and taking it to the next level.

References

1. https://create-react-app.dev/docs/getting-started/
2. https://pypi.org/project/webrtcvad/
3. https://axios-http.com/docs/intro
4. https://www.freecodecamp.org/news/http-request-methods-explained/
5. https://github.com/Jason-Oleana/How-to-run-a-python-script-by-clicking-on-an-html-button/blob/master/README.md
6. https://medium.com/swlh/run-python-script-from-node-js-and-send-data-to-browser-15677fcf199f