RevoltronX Technologies Pvt. Ltd

Automatic Speech Recognition (ASR) model

A Project Report submitted in partial fulfillment of the requirements of RevoltronX Technologies Pvt. Ltd

by - Shruti Gupta
Department of Computer Science and Engineering

Email id: shrutigupta.cse37@gmail.com

Abstract

The ability of Automatic Speech Recognition (ASR) systems to handle diverse accents and dialects remains a critical challenge in the pursuit of equitable and accurate voice-based technologies. This report presents the design, implementation, and evaluation of a real-time speech recognition system using two prominent ASR solutions—Whisper (by OpenAI) and AssemblyAI. The aim is to assess their capabilities in transcribing speech accurately across varying dialects, enhance transcript quality through error correction, and explore post-processing methods using Natural Language Processing (NLP). Through comparative analysis, this report outlines the strengths, limitations, and applicability of each approach for dialect adaptation tasks and proposes improvements to build a robust and adaptable ASR pipeline.

Table of Contents

1. Introduction

Automatic Speech Recognition (ASR) is a transformative technology that enables computers to understand spoken language. Despite impressive progress, challenges such as dialect variability and accent-related errors persist. This project addresses these issues by evaluating and implementing two ASR systems Whisper and AssemblyAI - focusing on transcription quality, dialect handling, and error correction mechanisms.

2. Objectives

- Fine-tune or utilize existing ASR models to transcribe speech with varying dialects.
- Compare offline (Whisper) and cloud-based (AssemblyAI) transcription methods.
- Implement error correction for grammar and transcript clarity.
- Explore NLP-driven post-processing for misinterpretations.

3. Approach & Methodology

Two ASR pipelines are developed:

- **Whisper-Based (Offline):** Real-time microphone capture → Temporary WAV file → Transcription → Grammar correction using LanguageTool.
- **AssemblyAI-Based (Cloud):** Audio recorded and saved → Uploaded via REST API → Transcription polled → Optional grammar correction.

Comparative evaluation is conducted based on:

- Accuracy for accented speech.
- Latency and responsiveness.
- Ease of integration and deployment.
- Privacy and resource usage.

4. Tools and Technologies Used

Technology	Purpose
Whisper (OpenAI)	Local ASR model
AssemblyAI API	Cloud-based ASR service
sounddevice, pyaudio	Microphone-based recording
wave, os, tempfile	Audio file handling
language_tool_python	Grammar and text correction
Python	Programming language

5. Data Preprocessing & Selection

- **Recording Parameters:** Mono audio, 16-bit, 16kHz sample rate, 10-second duration.
- **Preprocessing:** Recorded audio saved as .wav and optionally normalized.
- **Selection:** Dataset consists of real-time user input and Mozilla Common Voice samples (for future testing).
- **Grammar Correction:** Detected using LanguageTool, corrected via its check and correct methods.

6. Model Architecture & Tuning Process

Whisper:

- **Base Model Used:** base variant for performance balance.
- **Architecture:** Transformer-based encoder-decoder.
- **Tuning:** Whisper was not fine-tuned in this phase, but the code supports switching to medium or large models for improved accuracy.
- **Limitations:** No streaming support; transcribes after recording completes.

AssemblyAI:

- **Model:** Proprietary API-based deep learning ASR model.
- **Customization:** Automatic detection of language and dialect; no local tuning needed.
- **Polling Strategy:** Implements delay between upload and transcript via API.

7. Implementation Details

7.1 Whisper-Based Transcription (Offline)

This version captures audio in real time and transcribes it using OpenAI's Whisper base model. It then applies grammar correction with the language_tool_python library to improve transcript quality.

Code Explanation:

- **Audio Recording**: The script uses the **sounddevice** module to record audio from the default microphone for 10 seconds at a sample rate of 16kHz:

```
audio = sd.rec(int(duration * samplerate), samplerate=samplerate, channels=1, dtype='int16')
```

- **Temporary File Handling**: The audio is saved to a temporary WAV file using **scipy.io.wavfile** and **tempfile**

```
with tempfile.NamedTemporaryFile(suffix=".wav", delete=False) as tmpfile:
wavfile.write(tmpfile.name, samplerate, audio)
```

- **Transcription:** The Whisper model transcribes the audio:

```
result = model.transcribe(temp_filename)
```

- **Grammar Correction:** Using **language_tool_python**, grammar issues in the transcript are corrected:

```
matches = tool.check(text)
corrected = language_tool_python.utils.correct(text, matches)
```

- **Loop and Cleanup**: The process loops continuously, deleting temp files after each transcription.

Pros:

- Fully local - no data sent externally
- Customizable with different model sizes
- Works offline, ensuring privacy

Cons:

- Limited built-in handling for dialects
- Slightly lower accuracy for accented English
- Real-time streaming not implemented (transcribes chunks)

7.2 AssemblyAI-Based Transcription (Cloud)

This script captures and uploads audio to AssemblyAI, then polls their API for transcriptions. It is ideal for hands-off transcription with high accuracy.

Code Explanation:

- **Audio Recording**: Using pyaudio, the script records 10 seconds of mono audio at 16kHz and writes it to a WAV file:

```
stream = p.open(...)
data = stream.read(chunk)
wf.writeframes(b".join(frames))
```

- **File Upload**: The audio file is uploaded to AssemblyAI's server using chunked POST requests

```
response = requests.post(upload_endpoint, headers=headers, data=read_file(filename))
```

- **Transcription Request**:The uploaded audio URL is submitted to the transcription endpoint:

```
response = requests.post(transcript_endpoint, json={'audio_url': audio_url}, headers=headers)
```

- **Polling for Results**: A polling loop checks the transcription status every few seconds until completed:

```
while True:
    response = requests.get(polling_endpoint, headers=headers)
    if data['status'] == 'completed':
        return data['text']
```

- **Loop and Cleanup**: The WAV file is deleted after each iteration to free resources

Pros:

- Excellent transcription quality
- Better handling of diverse accents
- API is easy to integrate and use

Cons:

- Requires internet connection
- Introduces latency due to upload/polling
- Privacy concerns due to cloud upload

8. Comparison Between Whisper and AssemblyAI

Feature	**Whisper (Local)**	**AssemblyAI (Cloud)**
Deployment	Local	Cloud-based
Accent/Dialect Adaptation	Requires fine-tuning	Built-in support
Internet Requirement	No	Yes
Transcript Accuracy	Moderate to High	High
Post-Processing	Implemented manually	Not built-in
Privacy	High	Moderate
Cost	Free (Open-source)	Paid (free tier available)

9. Evaluation Against Project Goals

Goal	**Whisper**	**AssemblyAI**
Dialect-sensitive transcription	Limited	Good
Real-time capability	Partial (batch chunks)	Delayed due to polling
Grammar correction	Implemented	Not implemented
NLP - based post processing	Planned	Planned

10. Recommendations and Improvements

For Whisper:

- Use larger models (medium, large) for better accuracy
- Fine-tune with dialect-rich datasets like Mozilla Common Voice
- Add streaming and speaker segmentation capabilities

For AssemblyAI:

- Implement grammar correction after transcription.
- Improve user interface with live transcript display.
- Use asynchronous API handling to reduce wait time.

General Suggestions:

- Pre-process with accent classifiers
- Apply noise reduction for cleaner input
- Build a web dashboard (Streamlit or Flask) for ease of use

11. Conclusion

This project demonstrates a comparative implementation of Whisper and AssemblyAI for speech recognition with a focus on dialect adaptation. Whisper offers full control and privacy, making it ideal for local deployment and fine-tuning, whereas AssemblyAI provides high-quality results out-of-the-box, especially for various accents. Both approaches can be enhanced with grammar correction and NLP-based post-processing to further improve transcript quality. Future work will include accent classification, multilingual support, and integration into user-friendly interfaces.

12. References

- OpenAI. *Whisper: Robust Speech Recognition via Large-Scale Weak Supervision*. 2022, https://github.com/openai/whisper
- Hugging Face. *Automatic Speech Recognition*. Hugging Face, https://huggingface.co/docs/transformers/tasks/asr
- Thompson, Dylan. *What Is ASR?* AssemblyAI, 30 Aug. 2022, https://www.assemblyai.com/blog/what-is-asr/
- Morris, J. X. *language-tool-python: Grammar and Style Checker in Python*. 2021, https://github.com/jxmorris12/language-tool-python.