# Self-Supervised Learning for Speech Recognition using wav2vec 2.0

Authors:

**Sri Teja Muthangi (W862g253)**

**Kavya Sri Modepu (N572v597)**

A logo with black text

Description automatically generated

Date of Submission: 5th December 2025

## Abstract

This project evaluates four modern speech recognition models - Wav2Vec2-Base, Wav2Vec2-Small, Whisper-Small, and XLSR-English-under real-world conditions. We tested them on CommonVoice2, LJSpeech, and a few custom English audio clips and measured word error rate (WER), character error rate (CER), inference time, and memory usage.

We kept preprocessing and evaluation consistent across all models. Wav2Vec2-Base achieved the highest overall accuracy, Wav2Vec2-Small ran faster and used less memory, XLSR-English handled varied accents and noise well, and Whisper-Small excelled on clean audio but needed more computational resources.

These results provide practical guidance Wav2Vec2-Base for general use, Wav2Vec2-Small for speed or low-resource devices, XLSR-English for noisy or accented speech, and Whisper-Small for studio-quality recordings.

Table of Contents

[Self-Supervised Learning for Speech Recognition using wav2vec 2.0 1](#_Toc215999315)

[Abstract 2](#_Toc215999316)

[1. Introduction 5](#_Toc215999317)

[1.1 Background and Motivation 5](#_Toc215999318)

[1.2 Project Objectives and Scope 5](#_Toc215999319)

[1.3 Significance and Applications 5](#_Toc215999320)

[1.4 User Interface (UI) Real Time Development Overview 6](#_Toc215999321)

[2. Methodology 7](#_Toc215999322)

[2.1 Deep Learning Models Used 7](#_Toc215999323)

[2.2 Data Collection and Preprocessing 7](#_Toc215999324)

[2.3 Model Architecture and Parameters 8](#_Toc215999325)

[2.4 Training Process and Hyperparameter Tuning 8](#_Toc215999326)

[3. Data 9](#_Toc215999327)

[3.1 Datasets Used 9](#_Toc215999328)

[3.2 Data Exploration and Visualization 9](#_Toc215999329)

[4. Experiments and Results 10](#_Toc215999330)

[4.1 Presentation of Experimental Setup 10](#_Toc215999331)

[4.2 Training and Evaluation Metrics 10](#_Toc215999332)

[4.3 Results of Model Training and Testing 10](#_Toc215999333)

[4.4 Performance Comparisons 12](#_Toc215999334)

[5. Discussion 15](#_Toc215999335)

[5.1 Analysis and Interpretation of Results 15](#_Toc215999336)

[5.2 Addressing Challenges Encountered 15](#_Toc215999337)

[5.3 Comparisons with Previous Work or Benchmarks 16](#_Toc215999338)

[6. Conclusion 17](#_Toc215999339)

[6.1 Summary of Key Findings 17](#_Toc215999340)

[6.2 Limitations of the Project 17](#_Toc215999341)

[6.3 Future Work and Recommendations 17](#_Toc215999342)

[7. References 18](#_Toc215999343)

[8. Appendices 20](#_Toc215999344)

[Appendix A - System Information 20](#_Toc215999345)

[Appendix B - Installed Package Versions 20](#_Toc215999346)

[Appendix C - Additional Figures (Real time deployment) 21](#_Toc215999347)

[Appendix D - Code Snippets 23](#_Toc215999348)

[Appendix E - Project Setup & Commands 29](#_Toc215999349)

[Appendix F - GitHub Repository 30](#_Toc215999350)

[9. Author Information 31](#_Toc215999351)

[Team Members and Contributions 31](#_Toc215999352)

## 1. Introduction

### 1.1 Background and Motivation

Automatic speech recognition (ASR) is used in voice assistants, accessibility tools, and many ways people interact with computers. Today’s models use deep learning and transfer learning to produce transcriptions that are almost as accurate as humans. Since pretrained models are easy to access, it’s important to see how well they perform and where they might fall short in real-world situations.

### 1.2 Project Objectives and Scope

We aimed to evaluate leading ASR models using consistent metrics and datasets. The scope included:

* Implementing four state-of-the-art ASR models.
* Standardized evaluation on public and custom datasets.
* Measuring WER, CER, inference time, and memory usage.
* Providing practical recommendations for model selection.

### 1.3 Significance and Applications

ASR technology is used in education, healthcare, conversational AI, assistive devices, and automatic transcription. This study compares different models to help researchers and practitioners choose the right one based on accuracy, speed, computing requirements, and how well it handles different accents or noisy conditions.

### 1.4 User Interface (UI) Real Time Development Overview

Additionally, we build an custom web interface that lets users upload audio files, run speech recognition in real time, and see the model’s predictions. The interface connects the backend ASR system with a simple, responsive frontend.

#### Frontend

* Built using React and Material UI.
* Key features include:
  + Audio upload widget
  + Real-time transcription display
  + Model selection dropdown
  + Progress indicators
  + Error handling and validation

#### Backend

* Implemented using FastAPI with a uvicorn server.
* Responsible for audio preprocessing, model loading, and inference.
* Returns transcriptions to the frontend in real time.

## 2. Methodology

### 2.1 Deep Learning Models Used

We evaluated several transformer-based ASR models:

* **Wav2Vec2-Base (960h):** Trained on 960 hours of Librispeech, this model gives good accuracy without needing too much computing power.
* **Wav2Vec2-Small (100h):** A smaller, quicker version that works well for low-resource setups or real-time tasks.
* **Whisper-Small:** This handles multiple languages and noisy audio well and trained to deal with varied speech patterns.
* **XLSR-English (XLSR-53):** It was pretrained on 53 languages and fine-tuned for English, this model manages accents and background noise effectively.

### 2.2 Data Collection and Preprocessing

* We resampled all audio to 16 kHz to keep inputs consistent across models.
* We applied standard torch audio transformations to normalize the audio.
* For Whisper, we converted audio into log-mel spectrograms to match its input format.
* We cleaned up transcripts by removing punctuation and converting everything to lowercase, making WER and CER calculations consistent.

### 2.3 Model Architecture and Parameters

* All models use transformer encoder architectures.
* Wav2Vec2 models rely on CTC decoding for transcription.
* Whisper uses an encoder-decoder transformer and beam search to improve transcription accuracy.

### 2.4 Training Process and Hyperparameter Tuning

We tested all ASR models as they were, without any additional fine-tuning, to check how they perform right out of the box. We also kept main hyperparameters the same across models to make the comparison fair. Beam search width, chunk sizes, and batch sizes were kept consistent to ensure decoding strategies, handling of varying audio lengths, and computational scaling didn’t influence results. This setup made sure that any differences we observed reflected the models’ true capabilities rather than variations in training or inference settings.

## 3. Data

### 3.1 Datasets Used

1. **Custom Speech Samples**
   1. We used two English audio files (eng1.wav and eng2.wav) representing natural conversational speech.
2. **CommonVoice2 Subset**
   1. 50 randomly selected samples capturing a diverse set of speakers, accents, and recording conditions.
3. **LJSpeech Subset**
   1. 50 samples from a single professional speaker, featuring high-quality, clean recordings.

### 3.2 Data Exploration and Visualization

* The average audio length across all datasets is approximately 4.5 seconds.
* The custom dataset contains natural background noise, while LJSpeech features clean, studio-quality recordings.
* CommonVoice2 shows the highest variability, with diverse speakers, accents, and recording conditions.

## 4. Experiments and Results

### 4.1 Presentation of Experimental Setup

We were done all the tests on a CPU-only setup since many real-world systems don’t have GPUs. We configured our system with PyTorch 2.9.0+CPU, Torchaudio 2.9.0, and Transformers 4.57.1, running on a machine with 4 CPU threads and 1.5 GB of free memory. We used FFmpeg 8.0 to handle audio decoding and preprocessing. This constrained setup enabled meaningful evaluation of inference time and memory usage, which are critical considerations for practical speech recognition applications.

### 4.2 Training and Evaluation Metrics

We evaluated performance using standard Automatic Speech Recognition (ASR) metrics. **Word Error Rate (WER)** measured the proportion of incorrectly predicted words, while **Character Error Rate (CER)** provided a more detailed assessment of transcription accuracy at the character level. In addition to accuracy, we recorded inference time per audio sample and memory usage during model execution to evaluate efficiency and resource consumption. Together, these metrics offer a comprehensive view of both the accuracy and computational requirements of each model.

### 4.3 Results of Model Training and Testing

#### 4.3.1 Results on Custom Audio Dataset

We were initially evaluated models on a small set of custom English audio files. When tested on **Wav2Vec2-Small** we got moderate accuracy, with WER values ranging from 0.20 to 0.08. **Wav2Vec2-Base** performed significantly better, achieving lower error rates and even perfect transcription on some samples. **Whisper-Small** also attained perfect accuracy on certain files but required considerably longer inference times, highlighting the higher computational demands of its encoder–decoder architecture. XLSR-English achieved intermediate performance, with WERs of 0.133 and 0.083 on the two test files, showing better accuracy than Wav2Vec2-Small but slightly lower than Wav2Vec2-Base. its inference time and memory usage were higher than Wav2Vec2 models but lower than Whisper-Small.

**Table 4.1 Performance on Custom Audio Samples**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Model** | **Language** | **WER** | **CER** | **Time (s)** | **Memory (MB)** | **File** |
| wav2vec2-small | English | 0.2000 | 0.1216 | 0.9268 | 345.73 | eng1.wav |
| wav2vec2-small | English | 0.0833 | 0.0286 | 1.0775 | 4.13 | eng2.wav |
| wav2vec2-base | English | 0.0667 | 0.0541 | 2.2503 | 358.08 | eng1.wav |
| wav2vec2-base | English | 0.0000 | 0.0000 | 1.1633 | 1.32 | eng2.wav |
| whisper-small | English | 0.0000 | 0.0000 | 15.5782 | 1023.33 | eng1.wav |
| xlsr-english | English | 0.133 | 0.068 | 5.74 | 1196.5 | eng1.wav |
| xlsr-english | English | 0.083 | 0.014 | 3.06 | 48.9 | eng2.wav |

#### 4.3.2 Results on CommonVoice2 and LJSpeech

Broader evaluations were conducted on the CommonVoice2 and LJSpeech datasets. we got strong and consistent performance on Wav2Vec2-Base across both datasets. In contrast, Wav2Vec2-Small had higher error rates on CommonVoice2, likely due to the dataset’s accent and noise variability. Whisper-Small achieved excellent accuracy on LJSpeech but struggled with CommonVoice2 and required significantly more computation time. XLSR-English performed well on CommonVoice2, benefiting from its multilingual pretraining, though it used more memory than the other models.

**Table 4.2 Performance on CommonVoice2 and LJSpeech**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Model** | **Dataset** | **WER** | **CER** | **Time (s)** | **Memory (MB)** |
| wav2vec2-base | CommonVoice2 | 0.1958 | 0.0661 | 1.1107 | 7.41 |
| wav2vec2-base | LJSpeech | 0.0667 | 0.0325 | 1.2248 | -0.28 |
| wav2vec2-small | CommonVoice2 | 0.2886 | 0.0984 | 1.0186 | 9.98 |
| wav2vec2-small | LJSpeech | 0.0960 | 0.0423 | 1.2101 | 0.69 |
| whisper-small | CommonVoice2 | 0.5731 | 0.3485 | 12.7623 | 12.24 |
| whisper-small | LJSpeech | 0.0351 | 0.0115 | 20.8801 | 0.13 |
| xlsr-english | CommonVoice2 | 0.0895 | 0.0278 | 1.9891 | 23.10 |
| xlsr-english | LJSpeech | 0.0681 | 0.0341 | 2.3813 | 1.19 |

### 4.4 Performance Comparisons

The comparative analysis across models highlights clear trade-offs between accuracy, efficiency, and resource requirements. **XLSR-English** achieved the lowest WER and CER on CommonVoice2, while **Whisper-Small** delivered the highest accuracy on LJSpeech. However, Whisper’s inference time was substantially longer than all other models, making it less suitable for low-resource or real-time applications. **Wav2Vec2-Base** offered the most balanced performance, combining strong accuracy with reasonable efficiency, whereas **Wav2Vec2-Small** emerged as the fastest model overall, ideal for scenarios where speed and low resource usage are critical.

A group of blue boxes with text

AI-generated content may be incorrect.

Figure 4.1 Comparison Across Models for Custom Audio datasets

A graph of different types of lines

AI-generated content may be incorrect.

Figure 4.2 Comparison Across Models for CommonVoice2 and LJSpeech

A graph of different colored squares

AI-generated content may be incorrect.

Figure 4.3 Average WER and CER by Model and Dataset

A comparison of different colors

AI-generated content may be incorrect.

Figure 4.4 Relative WER and CER improvement vs Wav2vec2-Base

## 5. Discussion

### 5.1 Analysis and Interpretation of Results

The results show **Wav2Vec2-Base** is the most reliable model overall, delivering consistently strong performance across the datasets and real-world audio samples. When tested, the model showed low error rates which make it well-suited for general-purpose speech recognition tasks. **Wav2Vec2-Small** achieved reasonable accuracy but got the fastest inference speed, making it highly applicable for deployment on low-power or resource-constrained devices. **Whisper-Small** demonstrated excellent robustness and precision on clean audio, but its high computational cost limits its practicality on CPU-only systems. **XLSR-English** exhibited strong cross-lingual generalization and performed particularly well on CommonVoice2, reflecting the benefits of its multilingual pretraining.

### 5.2 Addressing Challenges Encountered

We ran into several challenges during testing. Whisper-Small demanded a lot of memory and processing power, which caused delays and occasional slowdowns on CPU-only systems. The CommonVoice2 dataset added complexity with its wide range of accents, background noise, and varying recording quality, which increased the WER and CER for all models. On top of that, Wav2Vec2 and Whisper required different preprocessing steps-CTC-based models like Wav2Vec2 needed one approach, while encoder - decoder models like Whisper needed another. We had to carefully normalize inputs to ensure fair comparisons.

### 5.3 Comparisons with Previous Work or Benchmarks

Our results are in line with previous research. Baevski et al. (2020) showed Wav2Vec2 works well for English speech recognition, which matches the strong performance we observed. Whisper models also proved robust to noise and multilingual inputs, echoing findings from Radford et al. (2022). XLSR-English performed well on accented and diverse speech, supporting prior benchmarks that highlight its cross-lingual strength. Overall, our study confirms these trends and reinforces the reliability of these modern ASR models.

## 6. Conclusion

### 6.1 Summary of Key Findings

We tested modern speech recognition models on custom audio and public datasets. Wav2Vec2-Base was most accurate on custom data, Wav2Vec2-Small was fastest, Whisper-Small excelled on clean audio, and XLSR-English handled noisy and accented speech well. On CommonVoice2, XLSR-English performed best, while Whisper-Small was top on LJSpeech, showing trade-offs between accuracy, speed, and memory.

### 6.2 Limitations of the Project

This study was constrained by CPU-only experimentation, which may not fully capture the performance of large models that benefit from GPU acceleration. Dataset sizes were limited to subsets due to storage and computational constraints. Also, we ran all evaluations in zero-shot mode, so the models weren’t fine-tuned for any specific type of audio.

### 6.3 Future Work and Recommendations

We suggest future research should explore fine-tuning models on domain-specific datasets to improve accuracy and robustness. We think testing the models on multilingual datasets could give us a better idea of how well they work across different languages. Running experiments on GPU systems would show more realistic performance for big models like Whisper. Also, trying hybrid or ensemble approaches might improve transcription by combining the strengths of different models.

## 7. References

#### Research Papers

* Baevski, A., Zhou, Y., Mohamed, A., & Auli, M. (2020). *wav2vec 2.0: A framework for self-supervised learning of speech representations.* Advances in Neural Information Processing Systems (NeurIPS).
* Radford, A., Kim, J. W., Xu, T., Brockman, G., McLeavey, C., & Zaremba, W. (2022). *Whisper: Robust speech recognition via large-scale weak supervision.* OpenAI.
* Schneider, S., Baevski, A., Collobert, R., & Auli, M. (2019). *wav2vec: Unsupervised pre-training for speech recognition.* Interspeech.
* Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2018). *BERT: Pre-training of deep bidirectional transformers for language understanding.* arXiv:1810.04805.
* Park, D. S., Chan, W., Zhang, Y., Chiu, C.-C., Zoph, B., Cubuk, E. D., & Le, Q. V. (2019). *SpecAugment: A simple data augmentation method for automatic speech recognition.* Interspeech.

#### Datasets

* Mozilla Foundation. (n.d.). *CommonVoice Dataset.* Retrieved December 1, 2025, from <https://www.kaggle.com/datasets/danielgraham1997/commonvoice2>
* Ito, K., & Johnson, L. (2017). *The LJSpeech Dataset.* Retrieved December 1, 2025, from <https://www.kaggle.com/datasets/dromosys/ljspeech>
* Custom audio dataset (2024). Randomly downloaded speech samples for ASR evaluation.

#### Frontend Technologies

* Facebook Open Source. (2024). *React Documentation.* Retrieved December 1, 2025, from <https://react.dev/>
* Material-UI Team. (2024). *MUI Component Library Documentation.* Retrieved December 1, 2025, from <https://mui.com/material-ui/>

#### Backend Technologies

* FastAPI Team. (2024). *FastAPI Documentation.* Retrieved December 1, 2025, from <https://fastapi.tiangolo.com/>
* Flask Developers. (2024). *Flask Documentation.* Retrieved December 1, 2025, from <https://flask.palletsprojects.com/>
* Uvicorn Developers. (2024). *Uvicorn ASGI Server Documentation.* Retrieved December 1, 2025, from <https://www.uvicorn.org/>

## 8. Appendices

We have provided code snippets for audio preprocessing, model evaluation, and metric calculation and additional figures and visualizations illustrating the real-time deployment of the ASR system below.

### Appendix A - System Information

|  |  |
| --- | --- |
| **Component** | **Details** |
| **PyTorch Version** | 2.9.0+cpu |
| **Available System Memory** | 1.5 GB |
| **CPU Threads** | 4 |

### Appendix B - Installed Package Versions

|  |  |
| --- | --- |
| **Package** | **Version** |
| protobuf | 6.33.0 |
| torch | 2.9.0+cpu |
| torchaudio | 2.9.0+cpu |
| transformers | 4.57.1 |
| ffmpeg | 8.0 (essentials build – gyan.dev) |

### Appendix C - Additional Figures (Real time deployment)

A computer screen shot of a person's face

AI-generated content may be incorrect.

Figure 8.1 User Interface with Models dropdown

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8.2 Prediction by wav2vec2-large model

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8.3 Prediction by Whisper small model

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8.4 Transcribing by XLSR-53-large model

A screenshot of a computer

AI-generated content may be incorrect.

Figure 8.5 Prediction by XLSR-53-large Model

### Appendix D - Code Snippets

All supplementary technical code included below:

#### D.1 Model Wrapper

class ModelWrapper:

def \_\_init\_\_(self, model\_config):

self.config = model\_config

self.model = None

self.processor = None

def load(self):

"""Load model with memory optimization"""

print(f"\nLoading {self.config['name']}...")

try:

if self.config["type"]=="wav2vec2":

self.processor = Wav2Vec2Processor.from\_pretrained(self.config["name"])

self.model = Wav2Vec2ForCTC.from\_pretrained(

self.config["name"],

low\_cpu\_mem\_usage=True

)

else:

# whisper

self.processor = WhisperProcessor.from\_pretrained(self.config["name"])

self.model = WhisperForConditionalGeneration.from\_pretrained(

self.config["name"],

low\_cpu\_mem\_usage=True

)

self.model.eval()

print(f"Loaded {self.config['name']}")

return self

except Exception as e:

print(f"Error loading {self.config['name']}: {str(e)}")

raise

def transcribe(self, audio, sr):

"""transcribe audio"""

if self.config["type"] == "wav2vec2":

if audio.ndim > 1:

audio = audio.mean(axis=1)

inputs = self.processor(audio, sampling\_rate=sr, return\_tensors="pt", padding=True)

with torch.no\_grad():

logits = self.model(inputs.input\_values).logits

ids = torch.argmax(logits, dim=-1)

return self.processor.batch\_decode(ids)[0]

else:

inputs = self.processor(audio, sampling\_rate=sr, return\_tensors="pt")

with torch.no\_grad():

generated\_ids = self.model.generate(inputs.input\_features)

return self.processor.batch\_decode(generated\_ids, skip\_special\_tokens=True)[0]

def unload(self):

"""Free memory"""

try:

del self.model

del self.processor

torch.cuda.empty\_cache()

import gc

gc.collect()

print(f"Unloaded model and freed memory")

except Exception as e:

print(f"Error during unload: {str(e)}")

#### D.2 Audio Loader

def load\_test\_audio(audio\_path, max\_duration=30):

"""Load test audio file with duration limit"""

try:

try:

audio, sr = librosa.load(audio\_path, sr=16000, duration=max\_duration)

print(f"Loaded {audio\_path} with librosa")

return audio, sr

except Exception as e:

print(f"Librosa load failed, trying soundfile: {str(e)}")

audio, sr = sf.read(audio\_path)

if len(audio) > sr \* max\_duration:

audio = audio[:sr \* max\_duration]

print(f"Truncated {audio\_path} to {max\_duration}s")

if len(audio.shape) > 1:

audio = librosa.to\_mono(audio.T)

if sr != 16000:

audio = librosa.resample(audio, orig\_sr=sr, target\_sr=16000)

sr = 16000

print(f"Resampled to {sr}Hz")

return audio, sr

except Exception as e:

print(f"Error loading {audio\_path}: {str(e)}")

return None, None

#### D.3 Comparing models on Common Voice 2 and LJSpeech datsets

# Common Voice 2 + LJSpeech

SAMPLE\_SIZE = 50

SAMPLING\_RATE = 16000 #use None to keep the original value

def load\_audio\_librosa(file\_path):

audio, sr = librosa.load(file\_path, sr=SAMPLING\_RATE)

return audio, sr

def prepare\_dataset(df, text\_col="text", path\_col="audio\_path"):

# filtering files that exist

df = df[df[path\_col].apply(os.path.exists)]

n\_samples = min(SAMPLE\_SIZE, len(df))

df = df.sample(n\_samples, random\_state=42)

dataset = []

for \_, row in df.iterrows():

audio, sr = load\_audio\_librosa(row[path\_col])

dataset.append({

"speech": audio,

"sampling\_rate": sr,

"text": row[text\_col]

})

return dataset

# Common Voice 2

cv2\_df = pd.read\_csv("commonvoice2/train/train.tsv", sep="\t")

cv2\_df["audio\_path"] = cv2\_df["path"].apply(lambda x: f"./commonvoice2/train/clips/{x}.wav")

cv2\_df = cv2\_df[["audio\_path", "sentence"]].rename(columns={"sentence": "text"})

cv2\_ds = prepare\_dataset(cv2\_df)

# LJSpeech

lj\_df = pd.read\_csv("./LJSpeech-1.1/metadata.csv", sep="|", header=None, names=["path","text","\_"])

lj\_df["audio\_path"] = lj\_df["path"].apply(lambda x: f"./LJSpeech-1.1/wavs/{x}.wav")

lj\_df = lj\_df[["audio\_path", "text"]]

lj\_ds = prepare\_dataset(lj\_df)

print(len(cv2\_ds), len(lj\_ds))

# model evalutation

results = []

for model\_name, config in MODELS.items():

wrapper = ModelWrapper(config).load()

for dataset\_name, dataset in {"CommonVoice2": cv2\_ds, "LJSpeech": lj\_ds}.items():

for i, sample in enumerate(dataset):

try:

metrics = evaluate\_model(wrapper, sample["speech"], sample["sampling\_rate"], sample["text"])

if metrics:

metrics.update({

"model": model\_name,

"dataset": dataset\_name,

"sample\_idx": i

})

results.append(metrics)

except Exception as e:

print(f"Error on sample {i}: {e}")

wrapper.unload()

D.4 FastAPI Backend (Real-Time Deployment)  
A FastAPI backend was implemented to enable real-time deployment of the ASR models, allowing audio inputs to be transcribed on-the-fly. This lightweight backend demonstrates the practical applicability of pretrained models beyond offline evaluation.  
Repository: <https://github.com/sriteja-28/StateOfArt_SpeechRecognition/tree/main/backend>

D.5 React Frontend (Vite + MUI, Real-Time Deployment)  
A React-based frontend was developed using Vite and Material-UI (MUI) to provide an interactive interface for real-time speech transcription. The frontend connects with the FastAPI backend, enabling users to upload audio and view transcriptions instantly.  
Repository: <https://github.com/sriteja-28/StateOfArt_SpeechRecognition/tree/main/frontend>

### Appendix E - Project Setup & Commands

#### Backend Setup

python -m venv mlstateofartenv

.\mlstateofartenv\Scripts\Activate.ps1

pip install -r requirements.txt

uvicorn app.main:app --reload --port 8000

#### Frontend Setup

cd frontend

npm install

npm run dev

### Appendix F - GitHub Repository

Project Source Code:

The complete source code for this project - including model evaluation scripts, the FastAPI backend, and the React frontend - is available on GitHub: <https://github.com/sriteja-28/StateOfArt_SpeechRecognition.git> .

## 9. Author Information

### Team Members and Contributions

**Sri Teja Muthangi** led the technical development of the speech recognition system. He designed and implemented the ASR inference pipeline, developed evaluation scripts for WER, CER, inference time, and memory usage, and conducted experiments on custom datasets as well as CommonVoice2 and LJSpeech. He also performed analysis, generated visualizations, prepared tables, and contributed to writing the project report. Additionally, He also designed and implemented the full system architecture, including the React-based frontend with Material-UI and the FastAPI backend, enabling real-time speech transcription when user uploads an audio file offline.

**Kavya sri Modepu** managed data collection and preprocessing across all datasets, including Kaggle, CommonVoice2, LJSpeech, and custom audio samples. She ensured clean, consistent audio by trimming, normalizing, and validating samples. Kavya assisted in reviewing model outputs and assessing transcription quality. She also supported documentation of datasets, preprocessing steps, and challenges encountered during the project.

We coordinated regularly through weekly meetings, reviewed each other’s code, debugged issues together, and worked jointly to compile the final report and prepare the submission.

#### Contact Information

Sri Teja Muthangi – [w862g253@wichita.edu](mailto:w862g253@wichita.edu)  
Kavya – [N572v597@wichita.edu](mailto:N572v597@wichita.edu)

**Department:**

School of Computing,

A logo with black text

Description automatically generated  
Wichita, Kansas