EECE490 Project Report

Ahmad Hlayhel, Elias El Khoury, Houssam El Deen Soboh

December 2024

Project Overview

The project focuses on two main components: **Speech Recognition** and **Speaker Recognition**, along with countering these systems through adversarial audio manipulation.

1 Speech Recognition

1.1 Model Selection and Loading

We selected the Whisper model for its state-of-the-art transcription capabilities and robust architecture, which efficiently processes audio as log-Mel spectrograms through transformer-based encoders and decoders. The model is available in five variations: **tiny**, **base**, **small**, **medium**, and **large**, each differing in size, computational requirements, and accuracy. Due to hardware limitations and to balance efficiency with performance, we chose the **small** model, which offers reliable transcription accuracy while being resource-efficient. The model was loaded using the `whisper.load_model()` function.

1.2 Understanding the Whisper Model

The Whisper model follows an **end-to-end encoder-decoder architecture** designed for robust speech processing. Below is a breakdown of how it works and its key components (see Figure 1).

1.3 Audio Input and Preprocessing

The audio is converted into a **log-Mel spectrogram**, which represents the time-frequency characteristics of the audio signal. This spectrogram acts as the model's input, which is then passed to encoders.

Encoders: The encoder processes the spectrogram using multiple transformer encoder blocks. These blocks extract meaningful patterns and long-range dependencies from the audio data. The encoder produces a compressed, high-dimensional representation of the audio, which captures all relevant contextual information.

Decoders: The decoder generates transcriptions token by token, leveraging self-attention to focus on previously generated tokens, cross-attention to align with the encoder's output, and positional

encoding to preserve sequential order. It predicts the next token based on prior tokens and encoded features, ensuring coherent and grammatically correct transcriptions.

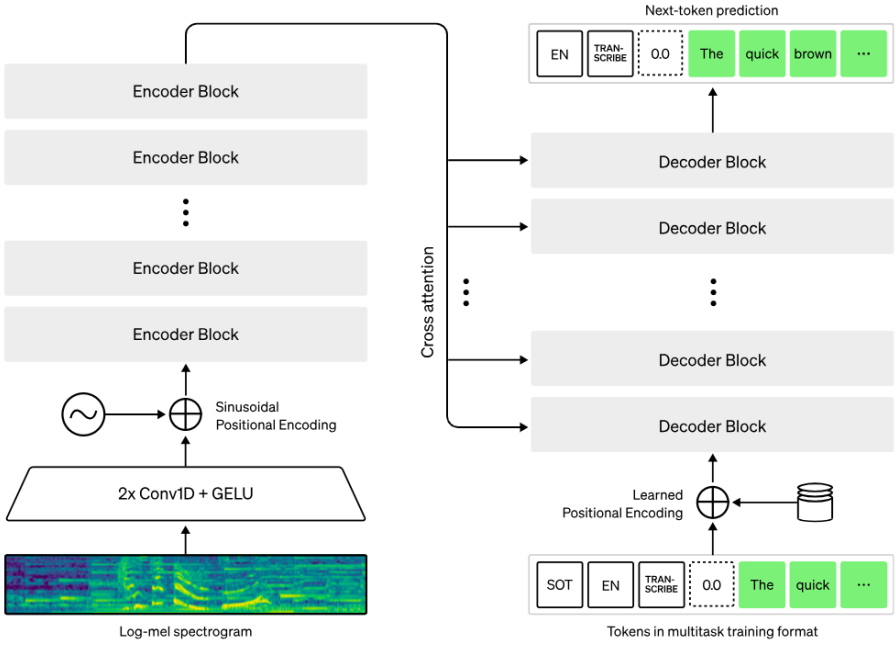

Figure 1: Whisper Architecture Overview.

Note: Whisper was trained on a large, diverse dataset containing both speech and text in multiple languages (VoxCeleb). This allows it to perform well in a **zero-shot setting**, handling unseen datasets with minimal errors.

1.4 Batch Transcription

For non-real-time transcription, audio files in formats such as MP3, WAV, and MP4 were processed from an input directory. Transcriptions were performed using `model.transcribe()` and saved as text files.

1.5 Real-Time Transcription

Real-time transcription was implemented using **PyAudio**, which captured live audio and processed it in 5-second chunks with a 0.1-second overlap. While the overlap ensured continuity, it introduced challenges in balancing latency and accuracy. Notably, if the entire audio is processed in one shot and fed to the model, it performs better compared to chunking with overlap, as chunking can disrupt context and reduce transcription quality. However, multi-threading and optimized pipelines were used to mitigate these challenges and maintain real-time performance.

1.6 Evaluating Accuracy

After transcription, the **Word Error Rate (WER)** was calculated to assess transcription quality. WER ranged from 0% to 10%, reflecting excellent transcription accuracy overall. Despite the challenges of chunking in real-time scenarios, WER remained consistently good, showcasing the robustness of the Whisper model.

2 Counter Speech

We implemented a real-time audio processing pipeline that introduces **adversarial perturbations** to the audio input and transcribes the perturbed audio. The primary goal is to manipulate the audio in such a way that it remains intelligible to human listeners but becomes challenging for the model to accurately transcribe.

2.1 Audio Processing and Perturbation

Each chunk of audio is processed as follows:

- The raw audio is converted to a NumPy array and normalized to the range $[-1, 1]$.
- The function `generate_adversarial_noise_frequency` applies **frequency-based perturbations**:
  - The audio is transformed into the frequency domain using Short-Time Fourier Transform (STFT).
  - Gaussian noise with a standard deviation of 0.22 is added to the magnitude of the frequency components, selectively distorting key frequencies.
  - The perturbed audio is reconstructed back into the time domain using inverse STFT.

This step ensures that the modified audio remains intelligible to human listeners while introducing subtle distortions that Whisper struggles to transcribe accurately.

The perturbed audio chunks are accumulated and concatenated. The complete adversarial audio is saved as a WAV file (`adversarial_output.wav`) for analysis and playback.

3 Speaker Recognition

The goal of speaker recognition is to recognize or confirm the identity of a speaker using distinctive aspects of their speech. This component's goal was to: recognize the speech, and then use adversarial perturbations for audio to compromise the model's capacity to precisely identify speakers while maintaining the audio's comprehensibility for human listeners.

3.1 Model Architecture

Feature Extraction

The model uses a **Thin-ResNet-34** architecture to extract frame-level features from spectrogram inputs. The input spectrograms are passed through convolutional and residual blocks with reduced

channels compared to standard ResNet-34 for efficiency. The output feature map has dimensions $T/32 \times 512$, capturing temporal and frequency information compactly.

Temporal Aggregation

The **NetVLAD** or **GhostVLAD** layer aggregates frame-level features into a fixed-length utterance-level descriptor. Each frame is softly assigned to clusters, and residuals between frames and cluster centers are computed. GhostVLAD introduces "ghost clusters" to down-weight noisy or irrelevant frames, enhancing robustness.

Classification Head

A fully connected layer reduces the dimensionality of the aggregated features to 512. The output is optimized using **Additive Margin Softmax (AM-Softmax)**, which minimizes intra-class variation and maximizes inter-class separation, ensuring robust speaker verification performance.

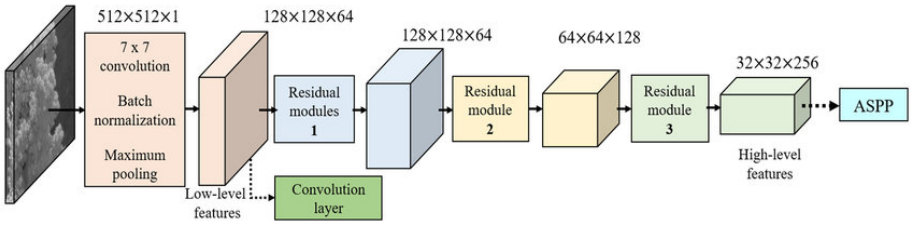

Figure 2: Thin-ResNet-34 Architecture

3.2 Fine-Tuning

Hyperparameters Summary

Parameter	**Value**
Input Dimensions	$(250, 40, 1)$
Batch Size	32
Optimizer	Adam
Learning Rate	1×10^{-5}
Loss Function	Categorical Cross-Entropy
VLAD Clusters	64
Bottleneck Dimension	512
Epochs	50
Early Stopping Patience	10
Trainable Layers	Last layers after freezing the first 100 layers

Table 1: Hyperparameters used for fine-tuning.

Challenges

- **Input Dimension Mismatches:** The pretrained model expected specific input dimensions for spectrograms. We adapted convolutional and pooling layers in the base model to handle MFCC features with dimensions $(250, 40)$.
- **Layer Compatibility:** Adjustments were needed in the *backbone code* to ensure compatibility at every layer while retaining the pretrained knowledge.
- **Training Stability:** Freezing the initial layers and using a low learning rate were essential for fine-tuning without overfitting or destabilizing the model.

3.3 Code Explanations

Two scripts were used for this task:

- **Without Countering (`speaker_recog_no_counter.py`):** Assesses the speaker recognition model's baseline performance using clear audio.
- **With Countering (`speaker_recog_counter.py`):** This script uses a custom function, `generate_adversarial_noi` to introduce targeted adversarial perturbations to the audio to interfere with speaker recognition.

Custom VladPooling Layer

The "most important" features from each frame are captured by a pooling approach used in audio processing to summarize feature maps. The layer highlights discriminative patterns by aggregating input features using 64 trainable cluster centers and weights.

MFCC Extraction and Preprocessing

Mel Frequency Cepstral Coefficients, or MFCCs, are a compact representation of the audio power spectrum that captures the spectral characteristics crucial for speaker differentiation. They isolate the frequencies that are most important for speech, simulating how the human ear perceives sound.

- The Fast Fourier Transform (FFT) is used to window and convert the audio signal into the frequency domain.
- Mel filters are used to draw attention to frequencies that are perceptually significant.
- The representation is compressed into a predetermined number of coefficients using a Discrete Cosine Transform (DCT).

Chunking Function

Chunking quickly processes lengthy recordings by breaking them up into smaller audio chunks.

- At 16 kHz samples per second, the audio is separated into fixed 5-second overlapping parts.
- The recognition model receives each chunk separately.
- Overlapping ensures that no important information at the boundaries is lost.

Prediction Function

This function outputs the predicted speaker and confidence level for each chunk. For each audio chunk, features are extracted and passed through the trained model. Then, the model assigns probabilities to each speaker class, with the highest probability corresponding to the predicted speaker. Confidence is computed as the softmax output for the predicted class, representing the model's certainty.

3.4 Baseline Results (Without Countering)

The speaker recognition model's performance on clean audio samples (no adversarial perturbations applied) is compiled in Figure 3. The majority of speakers were accurately predicted, frequently with high confidence (B: 0.92 and X: 1.00, for example). Misclassification occurred with an audio sample of M being predicted as A as well as a sample of X being predicted as Z, both with low confidence levels. Overall, the model is performant with an accuracy of $16/18 = 88.89\%$.

Speaker Analyzed	Speaker Predicted	Confidence Level
A	A	0.77
	A	0.88
B	B	0.92
	B	0.7
C	C	0.83
	C	0.9
D	D	0.92
	D	0.4
M	M	0.69
	A	0.5
X	X	1
	Z	0.4
Y	Y	1
	Y	0.4
Z	Z	0.85
	Z	1

Figure 3: Baseline Results (Without Countering).

4 Counter Speaker Recognition

The function `generate_adversarial_noise_frequency` introduces targeted perturbations to compromise speaker recognition.

4.1 Targeting Critical Bands

In order to capture speech characteristics, such as pitch, harmonics, and formants, speaker recognition models rely on specific frequency bands. This function optimizes the effect of disturbances by concentrating on these "critical bands." Critical bands in the bound range of 10–100 are the focus of this function. These are equivalent to 312.5 and 3125 Hz frequencies as calculated:

$$\text{Frequency per bin} = \frac{\text{Sampling rate (sr)}}{\text{FFT size (n\_fft)}} = \frac{16,000}{512} \approx 31.25\,\text{Hz}.$$

So, a range of $(10, 100)$ bins covers:

$$\text{Lower Bound} = 10 \times 31.25 = 312.5\,\text{Hz}, \quad \text{Upper Bound} = 100 \times 31.25 = 3125\,\text{Hz}.$$

Analyzed using the Nyquist Theorem and sampling rate, fmax = sr/2 = 8kHz. However, human speech predominantly lies below 4000 Hz. That's why most of the energy is concentrated between 300 Hz and 3000 Hz.

The following factors make these frequencies essential to human speech:

1. **Formants**: Formants are the vocal tract's resonance frequencies that give a speaker's voice its distinctive qualities. For the majority of human speech, the first three formants (F1, F2, and F3) usually lie between 300 and 3500 Hz (e.g., 300, 1500, and 2500 Hz).
2. **Harmonics**: Harmonics are integer multiples of the fundamental frequency. They correspond to the pitch. Since the fundamental frequency usually ranges from 90 to 300 Hz for speech, harmonics within the selected targeted range correspond to critical overtones for distinguishing speakers' timbres and pitches (Kacur et al., 2022).

Using Noise

Gaussian noise is the most significant compromise measure. In the function, it was generated with a mean of zero and standard deviation of 3.4. Gaussian noise is preferred because it is simple to generate and can effectively interfere with particular frequency-domain properties.

Using Temporal Variations

Oscillations are generated within the target bands by combining sine and cosine functions. They add time-dependent variations to the perturbations, making the noise dynamic and more difficult for the model to filter out.

Using Subharmonics

These add energy to lower-frequency bins that are half as high as the essential bands. Thirty percent of the magnitude of each bin in the critical band is added to the corresponding bin at half its frequency. This makes it harder for the model to detect patterns.

4.2 Results with Countering

Figure 4 shows how well the speaker recognition model performed following the application of adversarial perturbations. With a notably high degree of confidence, the model incorrectly identified the majority of speakers as Speaker X (e.g., B: 1.00, C: 1.00). However, Speaker M remained misclassified as Speaker A. So, the countering of speaker recognition is successful with an accuracy of $17/18 = 94.44\%$.

Speaker Analyzed	Speaker Predicted	Confidence Level
A	X	0.62
	X	0.62
B	X	1
	X	1
C	X	1
	X	0.9
D	X	0.5
	X	0.6
M	X	0.92
	A	0.5
X	X	1
	X	0.8
Y	X	0.67
	X	1
Z	X	1
	X	1

Figure 4: Results with Countering.

5 Conclusion

The codes demonstrate how adversarial perturbations degrade the performance of Whisper while keeping the audio intelligible to humans. This exposes the vulnerabilities of speech and speaker recognition systems to targeted noise and frequency modifications, showing that they can be "fooled" into producing incorrect outputs without compromising audio usability. Such techniques are valuable for testing model robustness and exploring defenses against adversarial attacks.

6 References

University of Manitoba. Formants in Phonetics. *University of Manitoba*. Available at: `https://home.cc.umanitoba.ca/~krussll/phonetics/acoustic/formants.html` (Accessed: December 2024).

Kacur, J., Puterka, B., Pavlovicova, J., and Oravec, M. (2022). Frequency, Time, Representation and Modeling Aspects for Major Speech and Audio Processing Applications. *Sensors*, **22**(16), p.

6304. Available at: https://doi.org/10.3390/s22166304.