Truth or Trap: Fake Speech Detection Using Deep Learning

AITech Hackathon 2025 – HCLTech x IIT Mandi

Group Number: 31

Team Members:

Ankit(B23118)
Bhavik Ostwal(B23395)
Lavish Singal(B23400)
Nishant Nehra(B23163)
Piyush Roy(B23355)

May 4, 2025

Contents

Truth or Trap: Fake Speech Detection Using Deep Learning

AITech Hackathon 2025 – HCLTech x IIT Mandi

Team Members: Ankit, Bhavik Ostwal, Lavish Singal, Nishant Nehra, Piyush Roy

May 4, 2025

1 Introduction

With the advancement of generative AI technologies, deepfake audio has emerged as a major concern in digital communication. These artificially generated voices mimic real individuals with alarming accuracy, posing threats to security, misinformation, and privacy. Detecting such fake audio in real-time is crucial for maintaining trust in digital systems and preventing malicious use of AI. This project was developed as part of the CS671 - Deep Learning and Application course in collaboration with HCLTech.

2 Problem Statement

- Develop an AI-based solution to detect fake voice recordings generated using deepfake or voice cloning techniques.
- The model should classify an input audio clip as **real** or **fake** with high accuracy.
- The system must be capable of working on real-time inputs.
- The approach should utilize efficient data preprocessing and deep learning-based audio classification.
- Ensure proper validation of the model and demonstrate its performance on various voice samples.

3 Tech Stack

This section outlines the programming environment, key libraries, and data-processing pipeline used in the development and training of our model.

- **Programming Language:** Python
- **Deep Learning Framework:**
  - `PyTorch`: Used for building and training neural networks including model definition, loss computation, and backpropagation.

`torchaudio` library is primarily used for audio preprocessing tasks within the project. One of its main roles is to ensure that the input audio waveform is resampled to a consistent sampling rate of 16 kHz, which is the required input format for the deepfake detection model

- **Pretrained Model and Processor:**
  - `transformers` (by HuggingFace): The `Wav2Vec2Model` and `Wav2Vec2Processor` from the `facebook/wav2vec2-base-960h` checkpoint were used for extracting contextual embeddings from raw audio waveforms.
- **Dataset Management:**
  - `AudioDataset` (custom class): Defined to load audio waveforms and corresponding labels from the HDF5 files.
  - `h5py`: Used to store the preprocessed waveforms and labels in a structured, efficient format for training.
- **Audio Preprocessing:**
  - `librosa`: For loading and resampling audio to a common format (16kHz).
  - `pydub` and `pydub.silence`: For silence removal based on energy thresholds.
  - `pyroomacoustics`: For simulating reverberation using Room Impulse Response (RIR) generation.
  - `scipy`: Applied RIR to simulate realistic reverberant conditions.
  - `numpy`: Used throughout for array processing, normalization, masking, and padding.
- **Utilities:**
  - `tqdm`: Used for progress visualization during model training epochs.

4 Data Set

For this project, we have used the **Fake-or-Real (FoR) Dataset** available on Kaggle. This dataset contains a large collection of audio samples labeled as either *real* or *fake* (deepfake). The fake samples are generated using advanced voice cloning and speech synthesis techniques, making them suitable for training robust classification models.

- **Source:** Kaggle – The Fake-or-Real (FoR) Dataset
- **Labels:** Real, Fake
- **Format:** Audio files in WAV format
- **Purpose:** To train and evaluate deep learning models for binary classification of audio authenticity

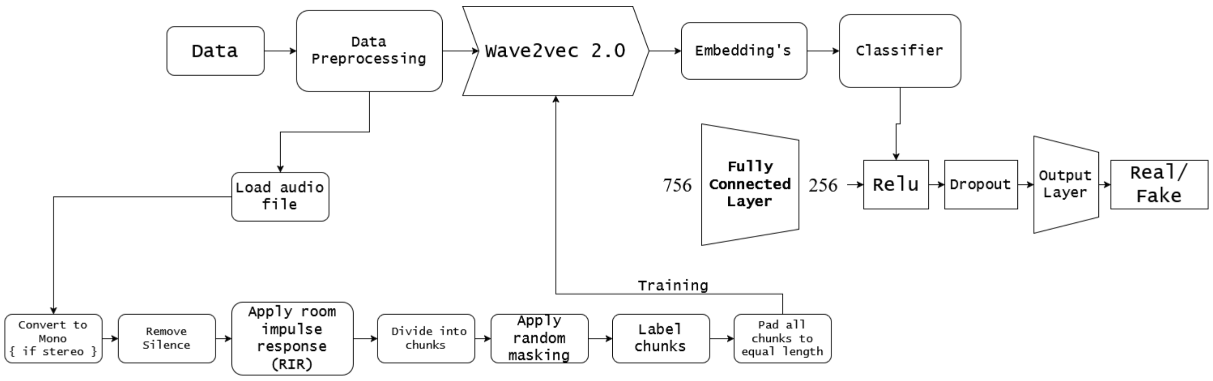


Figure 1: Overall Solution Architecture of the Classifier Model

5 Solution

The architecture of the classifier model is illustrated in Figure 1. It shows the flow of data through the layers of the neural network, from input to output.

The model consists of:

- An input layer that receives the embedding vector.
- A fully connected hidden layer with ReLU activation.
- A dropout layer (used during training to reduce overfitting).
- An output layer that produces a single scalar value, representing the likelihood of the audio being fake.

As a enhancement, we incorporate reverberation effects by convolving the audio signal with a **Room Impulse Response (RIR)**. This simulates real-world acoustics and increases the robustness of the model against variations in recording environments. Adding reverberation allows the classifier to generalize better, especially for audio recorded in echoic or reverberant settings, which are common in real-life use cases.

5.1 Data Preprocessing

To ensure the model is trained on high-quality and diverse audio features, we implemented a comprehensive preprocessing pipeline. The steps are detailed below:

1. **Loading Audio:** Each audio file is loaded using the `librosa` library with a sample rate of 16 kHz. The dataset is assumed to be single-channel, so stereo-to-mono conversion is not necessary.
2. **Silence Removal:** Silence segments are removed using `pydub`'s silence detection utility. Audio is first converted into 16-bit integer format and then segmented using a threshold-based method. Only non-silent chunks are retained for further processing, which helps in focusing the model on useful speech content.
3. **Reverberation Simulation:** To make the model robust to various acoustic environments, we simulate room reverberation using `pyroomacoustics`. A Room

Impulse Response (RIR) is generated for a shoebox-shaped room with defined dimensions, source, and microphone positions. The RIR is then convolved with the audio using FFT-based convolution to simulate reverberation.

4. **Chunking:** Each processed audio signal is split into overlapping chunks of 2 seconds with a hop length of 1 second. This ensures temporal consistency and augmenting the dataset with overlapping contextual frames. A random 10% portion of each chunk is zero-masked to encourage robustness against missing or corrupted data.

5. **Labeling:** Fake and real audio samples are labeled as 1 and 0 respectively. Each chunk inherits the label of its original file.

6. **Padding:** All chunks are padded to match the length of the longest chunk to ensure uniform input dimensions for model training.

7. **HDF5 Storage:** The final preprocessed dataset (waveforms and labels) is stored in an HDF5 file format. This format is efficient for large-scale training and supports fast I/O access.

The preprocessing script enables scalable and efficient handling of large audio datasets while introducing essential augmentations like silence trimming and artificial reverberation for better generalization.

5.2 Training

We trained a binary classifier using embeddings extracted from a pre-trained Wav2Vec2 model (`facebook/wav2vec2-base-960h`). The model was frozen and used as a feature extractor to generate mean-pooled representations from 16kHz mono audio waveforms. These embeddings were passed to a custom neural classifier for training.

- **Dataset:** The training data was loaded from the dataset file (HDF5 file) prepared in the earlier step (`trainingdataset_reverb.h5`) using a custom `AudioDataset` class.

- **Feature Extraction:** Embeddings were extracted in batches using the Wav2Vec2 model with mean pooling across the temporal dimension.

- **Model:** The classifier was a custom neural network defined in the `Classifier` class, with an input dimension equal to the Wav2Vec2 hidden size and a dropout rate of 0.2.

- **Training Parameters:**
  - Epochs: 50
  - Batch size: 512
  - Learning rate: 1×10^{-4}
  - Loss function: Binary Cross-Entropy with Logits
  - Optimizer: Adam

- **Checkpoints:** Model checkpoints were saved every 5 epochs. An early stopping condition was triggered if the average loss improvement over 4 epochs was less than 10^{-4}.

The training loop included loss tracking and checkpointing, ensuring the model could be resumed or evaluated at various stages of training.

5.3 Model

Our model architecture consists of two components: a frozen Wav2Vec2.0 encoder for extracting meaningful audio representations and a lightweight neural network classifier for binary classification.

- **Embedding Extraction:** We utilize the pre-trained `facebook/wav2vec2-base-960h` model to extract embeddings from raw 16kHz mono audio. The Wav2Vec2.0 model produces a sequence of hidden states, which are mean-pooled across the temporal dimension to form a fixed-size embedding vector for each audio sample. This model is frozen during training to reduce computational cost and overfitting.
- **Classifier Architecture:** The classifier is a simple feedforward neural network defined as follows:
  - Input Layer: Fully connected layer with input dimension equal to the hidden size of Wav2Vec2.0 embeddings.
  - Hidden Layer: A dense layer with 256 units followed by ReLU activation.
  - Dropout Layer: A dropout layer with a dropout rate of 0.2 was used during training to prevent overfitting. It was set to 0 during testing to ensure compatibility with the saved model weights.
  - Output Layer: A single-node output layer producing a raw score, which is interpreted as a logit for binary classification.

5.4 Real-Time Audio Classification Backend

The core of our real-time deepfake detection system is built using `Streamlit`, which serves a user-friendly web interface for uploading or recording audio inputs. The backend pipeline is as follows:

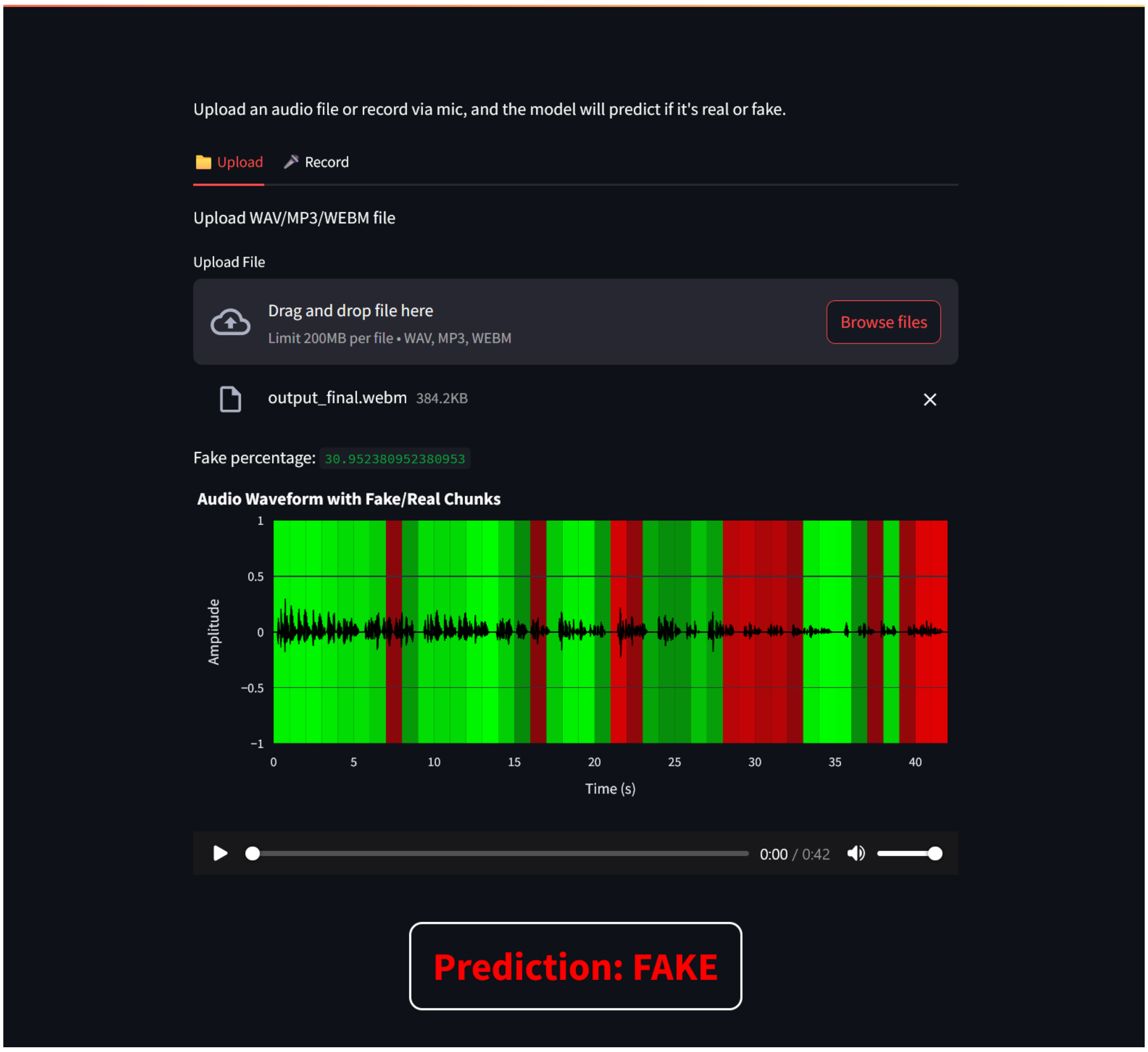
1. **Input Acquisition:** The user provides audio via microphone (using streamlit-mic-recorder) or file upload. WebM audio is converted to WAV format at 16kHz using FFmpeg.
2. **Preprocessing:** The input audio is downsampled to 16kHz and converted to mono. It is then split into overlapping chunks of 2 seconds (with 1-second hop) using a sliding window approach.
3. **Model Inference:** Each chunk is passed through the trained deepfake detection model. The model outputs a confidence score indicating the likelihood of the chunk being fake.

4. **Chunk-wise Aggregation:** For each second of audio, overlapping chunk scores are averaged. Chunks are then classified as fake (score ≥ 0.5) or real, and their confidence is visualized.

5. **Majority Vote Decision:** The final label (fake or real) is determined by the percentage of fake chunks. If more than a threshold (default 20%) of the audio is deemed fake, the entire sample is labeled fake.

6. **Visualization:** An interactive waveform plot is generated using `Plotly`, with red and green overlays showing fake and real segments respectively, weighted by confidence.

This framework allows users to not only get a binary decision but also a fine-grained view of suspicious regions in the audio stream.



Upload an audio file or record via mic, and the model will predict if it's real or fake.

Upload WAV/MP3/WEBM file

Upload File

Drag and drop file here
Limit 200MB per file • WAV, MP3, WEBM

output_final.webm 384.2KB

Fake percentage: `30.952380952380953`

Audio Waveform with Fake/Real Chunks

Prediction: FAKE

Figure 2: Real-time fake speech classification interface. Red represents fake and green represents real chunks.

6 Discussion

6.1 Model Accuracy

The final trained model achieved an accuracy of 0.8039, demonstrating effective classification performance between real and fake audio samples.

6.2 F1 Scores

The model achieved an F1 Score of 0.7703, indicating a good balance between precision and recall in distinguishing real and fake audio samples.

6.3 Confusion Matrix

To evaluate the classification performance in more detail, we plotted the confusion matrix of the model's predictions:

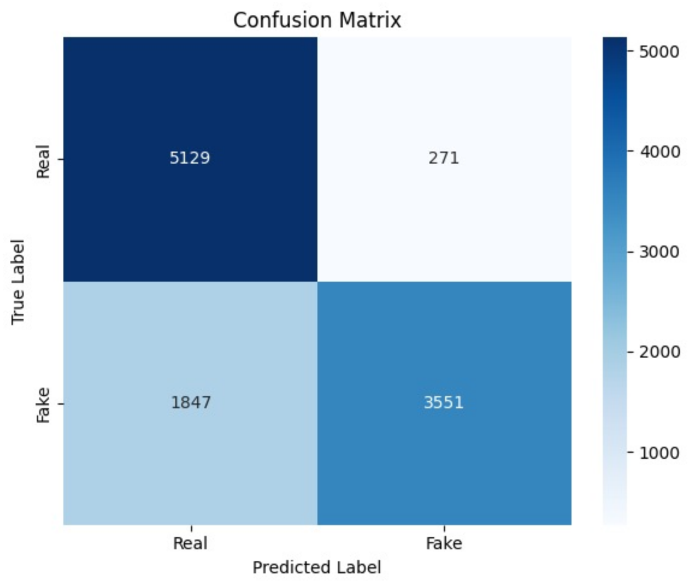


Figure 3: Confusion matrix illustrating true vs predicted labels for real and fake audio samples.

6.4 Condition for Model Failure

While our model performs well on many types of AI-generated audio, it struggles with the latest generation of TTS systems that produce highly realistic speech, including subtle breathing sounds and emotional tones. These advanced TTS models closely mimic human-like nuances, making detection increasingly difficult. However, with continued enhancements—such as incorporating more diverse and challenging training data and refining the augmentation strategies—there is strong potential to improve the model's ability to keep pace with evolving TTS technologies.

6.5 t-SNE Visualization

To understand the distribution of the learned feature representations, we use t-SNE (t-distributed Stochastic Neighbor Embedding) to project high-dimensional features into a 2D space. This helps in visualizing how well the model separates different classes in the latent space.

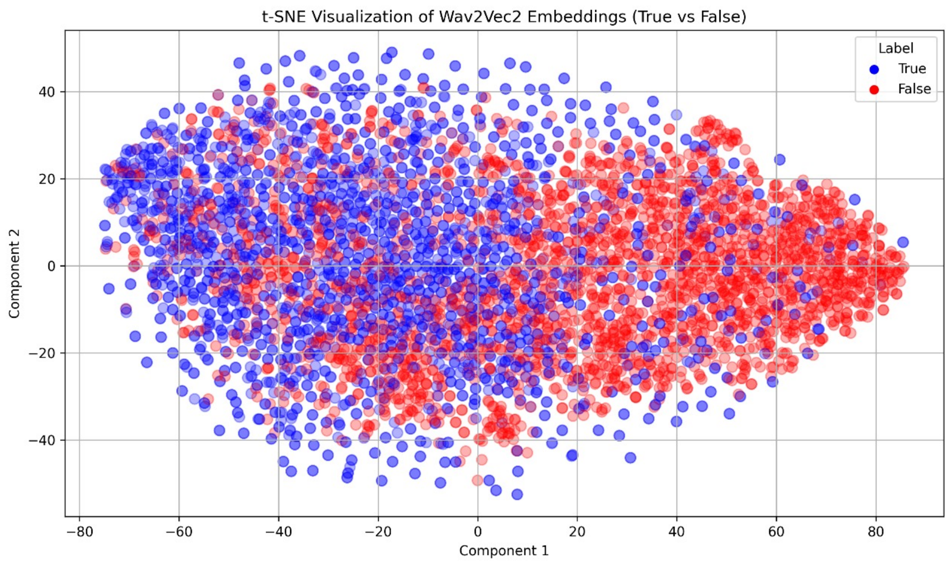


Figure 4: t-SNE visualization of feature embeddings. Each point represents a sample, colored by its class label.

6.6 Error Analysis on Confusion Between Similar-Sounding Samples

Adding reverb during training helps the model learn more complex acoustic patterns and subtle differences between human and AI-generated voices. This makes the model more robust against modern text-to-speech (TTS) systems, which often sound highly natural. By exposing the model to these audio variations, it becomes better equipped to handle diverse and challenging real-world inputs. As TTS technology continues to advance, further improvements to the augmentation pipeline can help the model stay effective and continue improving its discrimination capabilities.

6.7 Lightweight Deployable Model

Our model leverages Wav2Vec 2.0 to extract rich audio embeddings, which are then passed through a lightweight architecture consisting of a single hidden layer MLP with 256 units. The final output layer is a single neuron with a sigmoid activation, making it suitable for binary classification tasks. Thanks to the simplicity of the MLP—just one hidden layer and minimal parameters—the model remains computationally efficient and lightweight, allowing fast inference while still benefiting from the powerful feature extraction of Wav2Vec 2.0.

6.8 Support for Noisy or Low-Quality Audio

To make the model robust to noise and real-world audio imperfections, we augment the training data with reverb and apply masking, helping it adapt to noisy or lower-quality audio. This simple yet effective architecture ensures fast inference and low computational overhead while retaining strong performance through the rich representations provided by Wav2Vec 2.0.

7 Links

- Dataset: Fake-or-Real Dataset
- Youtube: Demonstration link for Real-Time Audio Classification

References

1. Speech DF Arena. (2025). *Speech DF Arena.* Hugging Face. `https://huggingface.co/spaces/Speech-Arena-2025/Speech-DF-Arena`

2. Hugging Face. (n.d.). *Wav2Vec2 model documentation.* Retrieved May 4, 2025, from `https://huggingface.co/docs/transformers/en/model_doc/wav2vec2`

3. Baevski, A., Zhou, H., Mohamed, A., Auli, M. (2020). *wav2vec 2.0: A framework for self-supervised learning of speech representations.* arXiv. `https://arxiv.org/abs/2006.11477`

4. BIL, E. E. E. C. S. (2021). *SPED 2021.* Retrieved from `https://bil.eecs.yorku.ca/wp-content/uploads/2021/10/sped2021.pdf`

5. BIL, E. E. E. C. S. (2020). *FoR-Dataset_RR_VT_final.pdf.* Retrieved from `https://bil.eecs.yorku.ca/wp-content/uploads/2020/01/FoR-Dataset_RR_VT_final.pdf#:~:text=in%20the%20beginning%20nor%20in,contains%20a%20total%20of%2069%2C400`