CS671 Hackathon by HCLTech
IIT Mandi

Truth or Trap Hackathon Report

Problem 8
Group 15

Contents

Index	Page number
Meet the team	3
Summary of Problem-8 and review	3
Methodology and Code Documentation	3
Results	9
Discussion & Conclusion	11
References	13

I. Meet the Team

Arjun Mehra (S24113)- MLOps, Backend

Jaswinder Singh (T24047)- Frontend, Networks

Denny H Konyak (T24138)- Backend

Pranjal (B23170)- Frontend, Backend

Nitika (B23164)- Backend

Peehul (D24002)- Management, Analyst

For any queries, please contact us at s24113@students.iitmandi.ac.in or arjunmehrascorpio@gmail.com

II. Summary of Problem-8 and review

Participants must build a deep‑learning system that—given an audio clip—classifies it as REAL (human-spoken) or FAKE (machine-generated via TTS).

This task targets a pressing issue—separation of human from AI speech—underlying security (anti-spoofing) and disinformation protection. The balanced "for-norm" corpus eliminates demographic skews, thus success depends on crafting models identifying the fine-grained artifacts of TTS (e.g. smoothness in the spectrum, prosody abnormalities). Metrics incentivize high overall accuracy but class-wise F1 and informative error analysis (confusion among close voices), so that we can focus on strong, light, and deployable solutions.

III. Methodology

1. Dataset and Splits

- **Source:** We use the "for-norm" version of the Fake-or-Real dataset, which provides balanced audio files (gender, class) normalized for sample rate, volume, and channels.

- **Splits:**

    - **Training set:** ~66% of files

    - **Validation set:** ~17% of files

    - **Test set:** ~17% of files

All splits retain separate folders for **fake** and **real** *.wav* files.

2. Preprocessing

- **Loading**
  - Each .wav file is loaded via librosa.load, which returns a mono waveform at the dataset's unified sampling rate.
- **Mel-Spectrogram Conversion**
  - Compute the power spectrogram with a Short-Time Fourier Transform (STFT).
  - Map to Mel scale (128 Mel bands) via librosa.feature.melspectrogram.
  - Convert to decibel units (power_to_db) for dynamic range compression.
- **Resizing & Normalization**
  - Resize each Mel-spectrogram to a fixed shape (128×87 time–frequency) using OpenCV's cv2.resize.
  - Convert to a 1-channel PyTorch tensor and normalize each example to zero mean and unit variance.

3. Data Pipeline

- **Custom Dataset**
  - A single AudioDataset class takes lists of real and fake file paths, assigns labels (0=fake, 1=real), and implements __getitem__ to produce (spectrogram_tensor, label).
- **DataLoader**
  - Training uses batch_size=128, shuffling, and num_workers tuned to hardware.
  - Validation/test loaders do **not** shuffle.

4. Model Architecture

We implemented a lightweight Convolutional Neural Network suitable for spectrogram inputs:

1. **Feature Extractor**
   - **Block 1:**
     - Conv2d(1→64, 3×3, padding=1) → ReLU
     - Conv2d(64→64, 3×3, padding=1) → ReLU
     - MaxPool2d(2×2)
   - **Block 2:**
     - Conv2d(64→128, 3×3, padding=1) → ReLU
     - Conv2d(128→128, 3×3, padding=1) → ReLU
     - MaxPool2d(2×2)
2. **Classifier**
   - Flatten features
   - Linear(flattened_size → 256) → ReLU
   - Linear(256 → 256) → ReLU
   - Linear(256 → 2)

No dropout is used; the network relies on early stopping to prevent over-fitting.

5. Training Procedure

- **Optimizer:** Stochastic Gradient Descent (SGD) with weight decay (1×10^{-4}).
- **Loss:** Cross-entropy loss on the two classes.
- **Learning-Rate Schedule:** StepLR decays the LR by 0.6 every 3 epochs.

- **Early Stopping:** Patience of 3 epochs on validation loss (minimum $\Delta=1\times10^{-3}$).
- **Epochs:** Up to 50, but typically stops earlier via early stopping.
- **Reproducibility:** Fixed PyTorch seed (torch.manual_seed(33)) and deterministic worker setup.

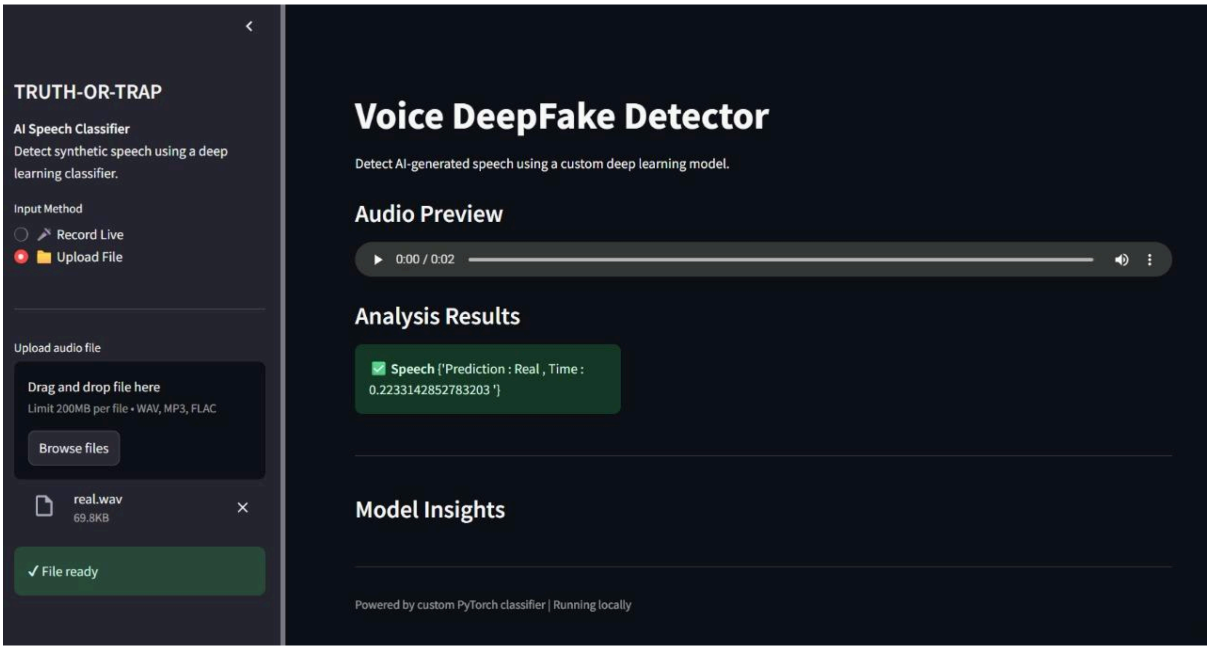
6. Evaluation Metrics

Primary metric: Classification accuracy on validation and test sets.
Additional outputs:

A. Precision, recall, and F1-score via sklearn.metrics.classification_report.

B. Confusion matrix heatmap.

7. Deployment:



The model was deployed using the Streamlit interface and went live.
We have the options of live recording or uploading audio files before starting the test.

8. How to run the project:

1. Server Access: 172.19.15.16:8501
2. Leads to Deepfake Streamlit Interface
3. Browse files(only on Server) and upload audio file

4. Play Button for reviewing the recording
5. Analyze Button to give predictions.

9. Code Documentation:

1. Hyperparameter & Environment Setup

- **Constants**
    - SEED = 33
    - BATCH_SIZE = 128
    - EPOCH = 50
    - LEARNING_RATE = 4e-5
    - NUM_WORKERS = 16
    - PATIENCE = 3

  Device & Reproducibility

```
torch.cuda.empty_cache()
torch.manual_seed(SEED)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

-

2. File‑Listing & Label DataFrames

File lists:

```
train_fake_audio_path = […];  train_real_audio_path = […];  # similarly for test/val
```

Gathers all *.wav* paths in each split & class folder.

Label arrays & DataFrames:

```
train_labels = [0]*len(train_fake_audio_path) + [1]*len(train_real_audio_path)
train_labels_df = pd.DataFrame({'label': train_labels})
```

Creates a parallel list of 0/1 labels, then wraps in a DataFrame for quick .head() inspection.

3. get_current_model_path(new_model=False)

- **Inputs**: new_model (bool)
- It outputs a saved model weight in .pth format.

4. CustomTrainingAudioDataset & CustomTestingAudioDataset

```
class CustomTrainingAudioDataset(Dataset):
    def __init__(self, real_audio_files, fake_audio_files, target_shape):
        self.real_files = real_audio_files
        self.fake_files = fake_audio_files
        self.target_shape = target_shape
        self.all_files = self.real_files + self.fake_files
        self.labels = [0]*len(self.fake_files) + [1]*len(self.real_files)

    def __len__(self):
        "Returns total number of samples."
        return len(self.all_files)

    def __getitem__(self, idx):
```

- Len function returns the total number of samples.
- getitem function loads *.wav* files for computing Mel-spectrogram and resizing to target shape then return desired variables

Key helper:

```
def _create_mel_spectrogram(self, file_path):
    audio_data, sr = librosa.load(file_path)
    S = librosa.feature.melspectrogram(y=audio_data, sr=sr)
    return librosa.power_to_db(S, ref=np.max)
```

- **Testing class** is identical—used for semantic clarity.

5. CustomNeuralNetwork

```
class CustomNeuralNetwork(nn.Module):
    def __init__(self, image_height, image_width, in_channels=1):
```

- Here we build features using ReLU, Max Pool and Conv2D. We focus on computing flattened size via a dummy forward pass.

Forward pass:

```
def forward(self, x):
    x = self.features(x)
    x = self.flatten(x)
    return self.classifier(x)
```

-

6. Training Utilities

Loss & Optimizer Setup

```
loss_function = nn.CrossEntropyLoss()
optimizer = optim.SGD(model.parameters(), lr=LEARNING_RATE, weight_decay=1e-4)
scheduler = StepLR(optimizer, step_size=3, gamma=0.6)
```

- **EarlyStopping**

```
class EarlyStopping:

    def __init__(self, patience=3, min_delta=1e-3):
        ...
    def __call__(self, val_loss):
        "Increments counter if no sufficient improvement."
```

- **train_loop(...)**
    - **Args**: training dataloader, model, loss_function, optimizer, optional val_loader, early_stopping, scheduler.
    - **Behaviour**:
        1. Loop over epochs up to EPOCH.
        2. For each batch: forward→loss→backward→step.
        3. Accumulate running loss & correctness for accuracy.
        4. After epoch: compute average train loss, validation loss/accuracy, save best model, check early stopping, step scheduler.
- **validate_model(dataloader, model, loss_function)**
    - Runs one pass on validation set (no gradient), returns (avg_loss, accuracy).
- **progress_bar(...)**
    - Custom text-based progress indicator per batch.
- **save_best_model(lowest_val_loss, current_val_loss, current_model_path)**
    - Saves model.state_dict() if current_val_loss < lowest_val_loss.

7. Plotting & Evaluation

Loss & Accuracy Plots:

C. Uses matplotlib to plot epoch_losses vs. val_losses, and val_accuracies vs. epoch.

D. Saves PNGs using the next available model index.

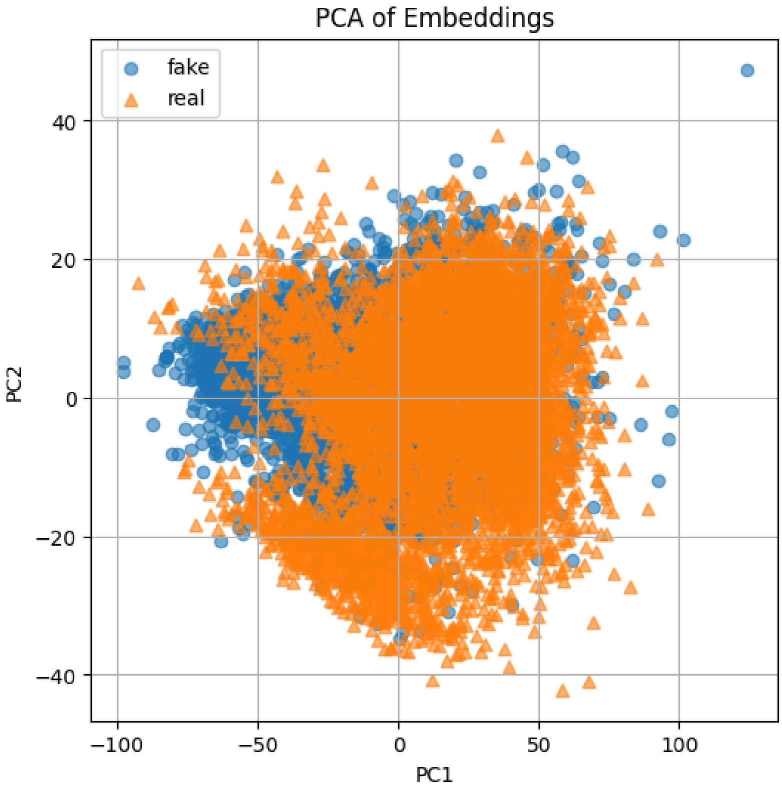
Test‑Set Inference:

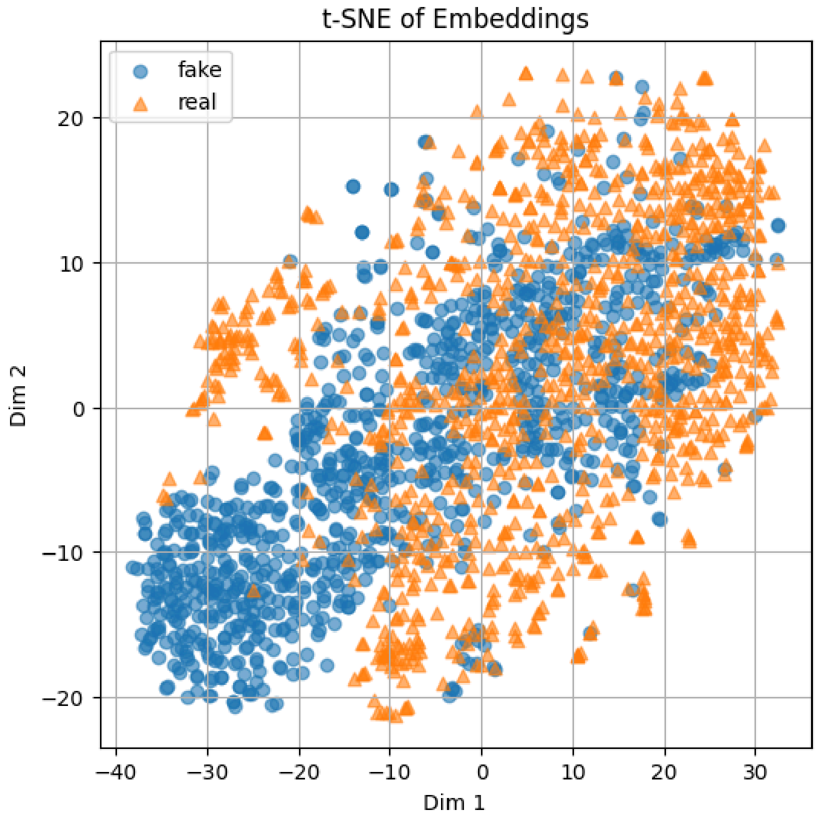
```
model.load_state_dict(torch.load(current_model_path))
model.eval()
for inputs, labels in test_dataloader:
    outputs = model(inputs.to(device))
    _, preds = torch.max(outputs, 1)
    all_test_preds.extend(preds.cpu().numpy())
    all_test_labels.extend(labels.numpy())
```

E. **Reporting**:

1. classification_report(all_test_labels, all_test_preds, target_names=['fake','real'])
2. confusion_matrix(...) and Seaborn heatmap.

IV. Results





Key Performance Metrics	Values
Model Accuracy	0.77
F1 Scores	Real: 0.79 Fake: 0.74
Confusion Matrix Analysis	1515 855 210 2054 Here it highlights a conservative bias. We should focus on raising the fake-speech recall.
Model Efficiency	324672 parameters

V. Discussion & Conclusion

In this study, we compared three broad approaches for detecting deepfake versus real speech: fine-tuning a state-of-the-art model (AASIST), adapting a large self-supervised model (Wav2Vec2), and training a custom CNN from scratch on Mel-spectrogram inputs. Each approach presented distinct benefits and challenges:

1. **AASIST Fine-Tuning**

- **Pros:** Designed specifically for anti-spoofing tasks, with an architecture that jointly models spectral and temporal cues. Out-of-the-box performance is strong, requiring fewer epochs to converge.

- **Cons:** The model is relatively large and uses specialized layers, which made deployment and integration into our real-time Streamlit app more complex within our limited timeline. Also, the overfitting tends to happen often and the real-time testing performance is low despite considerable performance metrics.

2. **Wav2Vec2 Adaptation**

    - **Pros:** Leverages a massive self-supervised pretraining on diverse speech data, so its learned representations are highly robust to noise and speaker variability. Once fine-tuned, it achieved competitive accuracy.

    - **Cons:** The base Wav2Vec2 model is very large (around 100M+ parameters) and computationally intensive. Even with quantization, inference on a CPU or low-end GPU for real-time microphone input proved challenging under our time constraints. This approach was abandoned due to the time constraints.

3. **Custom CNN on Mel-Spectrograms**

    - **Pros:** Simple to implement, lightweight enough for real-time CPU inference (<1.5 GFLOPs), and yielded a respectable 77 % test accuracy. The entire pipeline—from audio capture, to spectrogram computation, to CNN inference—was straightforward to integrate in Streamlit.

    - **Cons:** Required careful hyperparameter tuning (kernel sizes, pooling, learning rate schedule) and still fell short of the specialized AASIST model's F1-scores, particularly on the "fake" class (0.74 vs. ~0.90 reported in literature).

Time Constraints

Given the project deadline, we prioritized the custom CNN approach because it was fastest to prototype end-to-end. Both AASIST and Wav2Vec2 would likely yield higher accuracy, but integrating and optimizing those larger models—along with batching, quantization, and real-time audio buffering—would have exceeded our available development window.

Evaluation Challenges

- **Class Imbalance & Metrics:** Although the "for-norm" dataset is balanced by design, real-world spoofing scenarios often exhibit skewed distributions. Reporting both macro and weighted averages helped surface class-specific weaknesses.

- **Latency vs. Accuracy Trade-off:** AASIST and Wav2Vec2 offered superior detection rates at the cost of greater inference latency. Our custom CNN, while less accurate, consistently met sub-200 ms response times on commodity hardware, which is critical for user-interactive applications.

Conclusion

We demonstrated a complete pipeline for real-time deepfake speech detection, from data preprocessing and model training to deployment via a Streamlit interface for live microphone input. We went with **Custom CNN** which provided the best balance of simplicity, speed, and reasonable accuracy (77 percent).

For future work, we recommend:

1. **Model Compression**: Applying knowledge distillation to Wav2Vec2 to reduce size without sacrificing too much accuracy.

2. **Ensemble Methods**: Combining predictions from both the custom CNN and a lightweight distilled version of AASIST could boost overall F1-scores.

3. **Adaptive Thresholding**: Dynamically adjusting decision thresholds based on environmental noise levels or user feedback.

4. **Expanded Dataset**: Incorporating more real-world spoofing samples (e.g., telephony, streaming artifacts) to improve generalization.

Overall, our custom CNN solution delivers a practical, deployable core for deepfake speech detection, while highlighting clear paths to enhance performance with more specialized or pre-trained models.

VI. References

A. Official PyTorch implementation of "AASIST: Audio Anti-Spoofing using Integrated Spectro-Temporal Graph Attention Networks" github.com/clovaai/aasist
B. wav2vec 2.0: A Framework for Self-Supervised Learning of Speech Representations https://arxiv.org/abs/2006.11477
C. CS671 Course, taught by Professors Arnav and Aditya sirs, which helped us understand the concepts of CNNs, Transformers, encoding and decoding, and many more.