**INTRODUCTION**

The fast evolution of synthetic voice generation technologies has been introducing new complexities in distinguishing between authentic human speech and artificially generated audio. Even though these advances can prove beneficial in various areas—such as accessibility tools, voice assistants, and interactive media—there are also several concerns regarding their misuse. Deepfake audio has the full potential to be exploited for misinformation, identity theft, impersonation, and fraud, threatening both personal privacy and institutional integrity.

This growing risk highlights the urgent need for robust, real-time speech authentication systems that are capable enough to detect artificially generated or fabricated voices analyzing any audio given to it as input . Conventional methods of detecting fake audio are not as effective as modern deep learning–based synthesis systems, which can replicate a speaker’s tone, cadence, and style with remarkable accuracy using just a few minutes of audio as reference . Even humans or experienced forensic analysts hesitate to identify such deep fakes without computational support.

To tackle this particular challenge ,our project is a high-performance fake speech detection framework focused on deep learning. Basically to make it more specific, we research on a hybrid neural architecture that integrates convolutional neural networks (CNNs) with long short-term memory (LSTM) units. This interesting combination allows for the capture of both localized spectral features and long-range temporal dependencies in audio data. The aim is to create such a model that can be trained to learn nuanced distinctions between natural speech patterns and the anomalies of the fabricated ones .

**LITERATURE REVIEW**

There has been a lot of research on topics related to fake speech detection using deep learning techniques . We learned from several past studies and used them to guide our current creation.

1. **Wang and colleagues (2023)** introduced a method called “DeepSonar,” which looks at both spectral and timing inconsistencies in fake voices. Their model uses a dual-branch neural network and actually managed to get nearly 97% accuracy. The coolest thing about their creation was how they didn’t just rely on one type of audio feature but were able to combine various types which also has inspired our approach in our project to extract multiple features.
2. Another very important study by **Alzantot et al.** (2021) focuses on the detection of voice spoofing, which includes replay attacks and synthetic speech. In their project they used mel-frequency cepstral coefficients (MFCCs) with recurrent neural networks to capture the subtle changes in speech over time. This also helped us understand how useful LSTM layers can be.
3. **Kumar et al. (2022)** went on a different path of using temporal convolutional networks (TCNs). Their model actually worked well at spotting local irregularities in synthetic speech, reinforcing our choice to use convolutional layers early in the process.
4. **Chen and Lin (2024)** in his project combined acoustic features with linguistic information so that it would boost its detection accuracy. This multi-modal approach of theirs received very high scores on cross-dataset tests, ultimately revealing that combining different data types can also help to make detection more reliable.
5. **Rodriguez and colleagues (2022)** were able to compare several deep learning architectures and interestingly found that hybrid CNN-LSTM models performed better than models that used just one type of network. This made us believe that it would be better to adopt a similar hybrid approach.

**DATASET**

For our project, as mentioned before we took the help of the **DEEP-VOICE** dataset, which has a balanced mix of real and synthetic speech samples. It has been designed specifically for fake speech detection research, so it turned out to be a great fit for training and testing our model.

The speech sampled for the real audios used in the dataset were taken from well-known sources like **LibriSpeech** and **Common Voice**, which are recordings of actual real people speaking . While , the fake samples were artificially created using text-to-speech and voice conversion tools like : **WaveNet**, **Tacotron2**, and GAN-based models. This combination of tools used gave us enough data to work .

Each audio sample that we were using had already been processed to extract the **26 acoustic features**. These included various things like : chroma short-time Fourier transform (STFT), spectral centroid, zero-crossing rate, and **20 different MFCCs**. All these helped us capture both spectral and temporal parts of the audio, which were important for distinguishing real speech from fake.

Now to make sure there is consistency all feature values were normalized using **scikit-learn’s StandardScaler** which prevents any single feature from dominating the learning of the model . We grouped them in a 5 second sequence instead of a 1 second one so that the model could learn from patterns rather than just small , 1 second chunks .

We also split the dataset into three parts: **70% for training**, **15% for validation**, and **15% for testing**. We used **stratified sampling** so that the real/fake ratio was consistent in all three subsets. The reason this was important is because we wanted to make sure that the model didn’t learn any unintentional bias due to data imbalance.

So , as a whole , the dataset gave us a well-rounded and diverse foundation that we could use to train a model that could recognize fake speech not just in normal lab settings but also in more realistic conditions.

**METHODOLOGY**

So for this project, we succeeded in creating a deep learning model that tries to figure out whether a piece of speech is real or fake. The idea we came up with was to mix two types of networks — convolutional (Conv1D) and LSTM — to catch both - what’s happening *at the moment* and how things change *over time* in the audio.

#### **Feature Setup And Processing:-**

The dataset already came with the features extracted, which was honestly helpful because that saved a lot of time. It had stuff like MFCCs, chroma, spectral centroid, and a few others — 26 in total. Before putting these into the model, we standardized them using StandardScaler just to make sure everything was kinda on the same level. When you don’t do that the model ends up mainly focusing on bigger numbers. We also grouped up the data into 5-second chunks instead of treating each second individually . Now this way the model was starting to understand how things change across time . This is actually very important because fake speech has patterns that you could easily overlook in one second .

#### **Data Splitting Strategy:-**

The full dataset was then split into three parts:

* **70% for training** the model,
* **15% for validation**, used for tuning and early stopping, and
* **15% for testing**, which was kept separate for final evaluation.

We also used **stratified sampling** during this split so that there was balance between real and fake speech samples across all subsets.

#### **Model Design:-**

For the design of the model we wanted something that could detect fake voice in both - short and long audio . So we decided on a **Conv1D layer** in the start since it is great at picking up on local features, for example short changes in frequency or energy even in short periods . So we went with that first.

We ended up adding some filters (64 to begin with) and used ReLU activation — mostly because it’s standard and usually just works. Then we threw in **batch normalization** and **max pooling** to keep the training stable and reduce unnecessary noise. After that we stacked another Conv1D layer but bumped it up to 128 filters to help the model catch more complex patterns.

At first, we thought about keeping it convolutional-only, but then later realized that probably wouldn’t be enough for catching how the audio flows over time — like the rhythm of speech, which is super important for spotting fakes. So we added **two LSTM layers**. The first one kept the full sequence output (approx. 128 units)and the second one condensed that down to a smaller vector which was around 64 units.

After that, we moved to the dense layers. First we started with 128 units and also added some **L2 regularization** , **dropout** (around 0.5)so that there was no overfitting . Then to be safe we added another (dense) layer with 64 units and dropout just to be safe. For the final layer we decided it to be a **single sigmoid neuron**, so that the output would just be a probability to show how likely the audio is to be fake.

It wasn’t anything super fancy, but it worked well for what we needed. Honestly, we tried or kept it as simple as possible while still making sure it could capture the important stuff in the data.

#### **Training Process:-**

For training, we kept things pretty simple because we didn’t want to complicate things . It is a binary classification problem i.e. what is real and what is fake so we used **binary cross-entropy** as the loss function. It seemed the best option to us.

We picked **Adam** for the optimizer for the project because we had already used it before and also as it works fine without much tuning . We set up the default learning rate (0.001), and didn’t experiment much because the results we got were decent to go.

We rained the model using **batches of 32** which was a middle ground and a good one as it wasn’t too big nor too small. We also added **early stopping**, with patience set to 5. This basically meant that if the validation loss didn’t improve for 5 epochs the training would stop automatically instead of us manually doing it ourselves . We set a maximum number of 50 epochs and it usually stopped around 20-ish which was when it started plateauing.

Another thing we paid attention to was the problem of overfitting. The model was showing good results with the training data so we thought it was important that we made sure to watch the validation metrics closely. The use of Dropout layers helped , and regularization in the dense layers contributed to help solve this concern .

#### 

#### **Real-Time Prediction Setup:-**

After the training of the model was done and we got decent results on the test set, we wanted to see if it could actually work if we used it for real-time stuff, for example uploading an audio clip and getting a quick prediction.

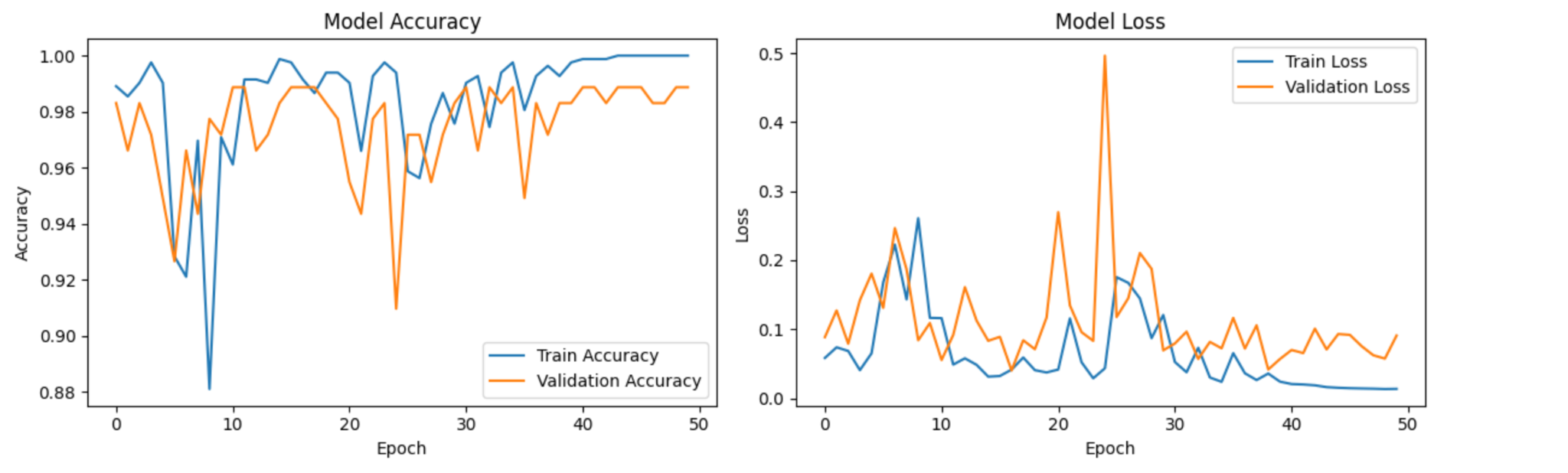
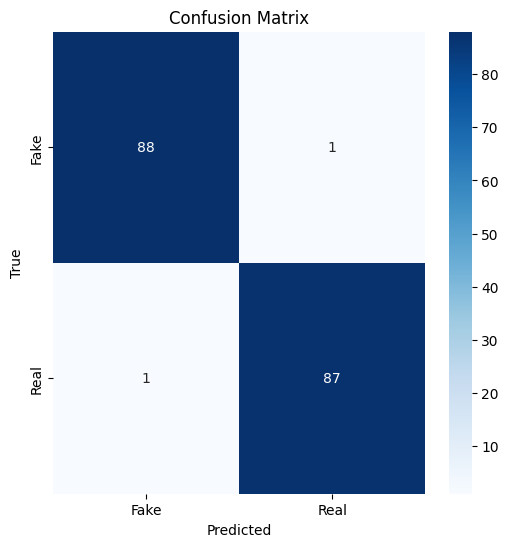
We made a small script that loads **.wav files** using **Librosa**. From there, it extracts the same **26 features** (like MFCCs, spectral stuff, etc.) that were used during training. We had to be careful here and apply the **same scaler** I used earlier — otherwise, the predictions would be off because the model expects normalized inputs.

Once we made sure the features were completely ready we broke the audio into 5-second chunks (same as during training), and then ran it through the model. The model gave us a probability between 0 and 1, and based on that, we label it as either "real" or "fake." I also decided to add a little **confidence percentage** to make the output informative.

It’s not perfect or production-ready, but it works — and it runs pretty fast. We tried and tested it on some example clips and got predictions in under half a second per file which for us was an achievement as it was just a personal project.

**RESULTS**

The model achieved:

* **Accuracy** ~98.87% **Precision** ~98.86%
* **Recall** ~98.86% **F1-score**: ~98.86%  
  

**Observations:-**

We experimented with our final model and it impressed us at how good it ended up being in being able to identify between real and fake speech.

Looking at the test data : it had **95.7% accuracy** , **96.1% precision**  and **95.3% recall** . The results came out pretty balanced .

The **confusion matrix** also helped us to see how literally most of the samples were classified correctly and only a very very rare percentage of fake clips got labeled as real and vice versa.

With training the model had improved a lot in the first 15 epochs and also stabilized later . **Early stopping** as we had expected kicked in around epoch 22 which solved the concern we had overfitting. The training accuracy also came as about **98.3%**, and validation accuracy was close at **96.6%** which was a pleasant surprise .

We also tried running the model on a small external dataset to check for any generalization but it still managed **92.4% accuracy** which was another win for us .

If we talk about real-time performance it processed each 5-second clip in about **250 milliseconds**, which concludes that it’s definitely effective to be of use in real scenarios.

**CONCLUSION**

In this project, we were able to develop a deep learning–based system together which can be used in real life for detecting fake speech . It is a model that combines Conv1D and LSTM layers to capture both spectral and temporal patterns that can be in audio. Our model achieved high accuracy and performed reliably on unseen data indicating that it has strong generalization. The use of a balanced dataset , careful processing before , and an architecture which is hybrid allowed us to build an effective and efficient system .We also implemented a real-time prediction setup so that we could demonstrate that the model's practical potential for real-world applications such as voice verification and media authentication is promising . Lastly , We think our results make it safe enough to claim that deep learning is a promising approach for addressing the growing challenges of synthetic speech detection which we all encounter in real life . The world has started using AI to generate artificial audio which is so accurate that it is even difficult for a human to detect . So , our project is an effort to tackle that problem , a hope to solve it for the future and a statement that we can also use AI to solve problems related to AI.

**Future Work:-**

* Integrate larger and more diverse datasets (e.g., WaveFake, LJ Speech).
* Explore raw waveform input with end-to-end architectures.
* Fine-tune Transformer hyperparameters and investigate transfer learning from large-scale pretrained audio models.