Identity Impersonation Detection

--------Using Voice Analysis----------

Team Members:

1.Anushka Anand

2.Dhiraj Inti

3.Manvi Aggarwal

4.Shria Sannuthi

5.Samridh Anand Paatni

6.Swastika Pandey

1.Problem Statement:

The advancement of AI voice impersonation technology poses a significant challenge in combating fraudulent activities, especially in sectors like banking and real estate. Cybercriminals now have access to sophisticated tools that can replicate voices with remarkable accuracy, making security vulnerabilities more prevalent. Companies specializing in voice recognition rely on deep learning algorithms, speaker verification, and voice biometrics to distinguish between genuine human voices and artificially generated ones. These technologies analyze intricate audio signal characteristics to detect and prevent voice impersonation fraud effectively. Besides fraudulent activities, AI voice impersonation has implications in entertainment, text-to-speech systems, and social engineering tactics targeting sensitive information. With the continuous evolution of voice cloning technology, there is an urgent need for robust security measures and heightened awareness to address the escalating risks associated with AI voice impersonation scams.

2.Dataset Used:

The Fake-or-Real (FoR) dataset is a collection of more than 195,000 utterances from real humans and computer generated speech. The dataset can be used to train classifiers to detect synthetic speech.

The "for-norm" version of the dataset contains the same audio files as the original version but with several enhancements to improve its quality and balance.

The dataset contains examples of real human speech, and DeepFake versions of those speeches by using Retrieval-based Voice Conversion.

Features:

Balanced Gender and Class: In the "for-norm" version, the distribution of audio files is adjusted to ensure a balanced representation of gender and class. This means an equal number of audio samples are included for each gender and class, helping to mitigate biases and ensure fairness in the dataset.

Normalization of Sample Rate: The sample rate of the audio files is normalized across the entire dataset. This means that all audio files are resampled to a consistent sample rate, ensuring uniformity and compatibility across different audio processing systems.

Volume Normalization: The volume levels of the audio files are normalized. This process involves adjusting the amplitude of each audio sample to a consistent level, reducing variations in loudness and ensuring consistent audio quality across the dataset.

Normalization of Number of Channels: The number of audio channels in each file is normalized. This involves ensuring that all audio files have the same number of channels, typically mono (1 channel) or stereo (2 channels). Normalizing the number of channels helps maintain consistency and simplifies processing during analysis or playback.

3.Architecture:

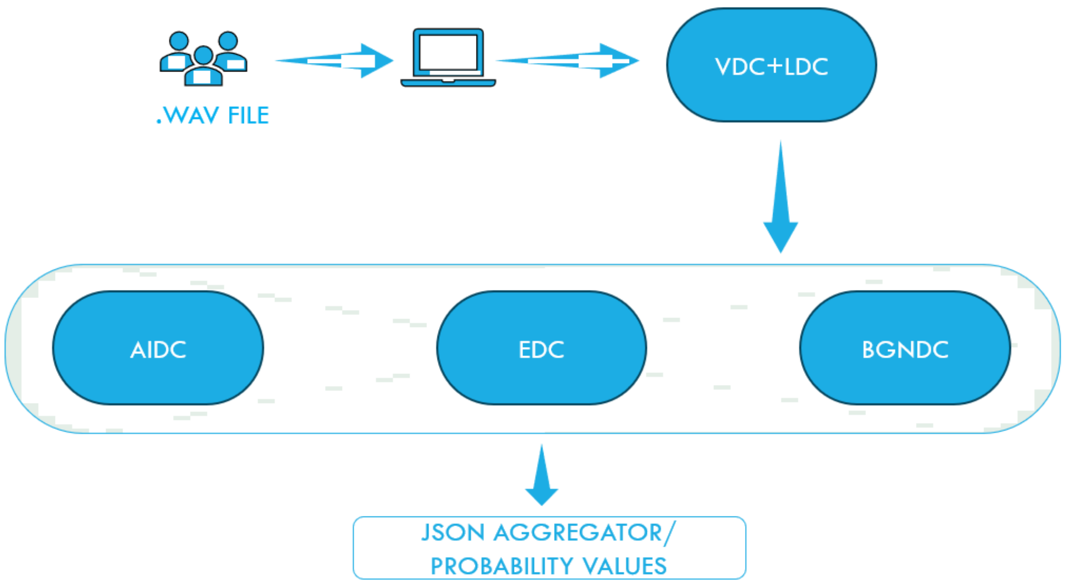


Fig1: High level design

4.Component wise Description:

4.1 Voice Detection Component:

Overview:

This script transcribes audio files in WAV format to text and detects the language of the transcribed text. It iterates through all WAV files in a specified directory, transcribes each file using the Google Speech Recognition API, and then detects the language of the transcribed text using the langdetect library.

For running the script, the following libraries are used:

- **`os`:** Provides functions for interacting with the operating system, such as reading file paths.
- **`speech_recognition` (imported as `sr`):** Allows recognition of speech from audio files.
- **One-liner:** An easy-to-use Python library for recognizing speech from audio files using various APIs.
- **`langdetect`:** A language detection library based on the `Google language detection library` and `N-Gram Language Identification Algorithm`.
- **`pydub`:** Manipulate audio with a simple and easy high-level interface.

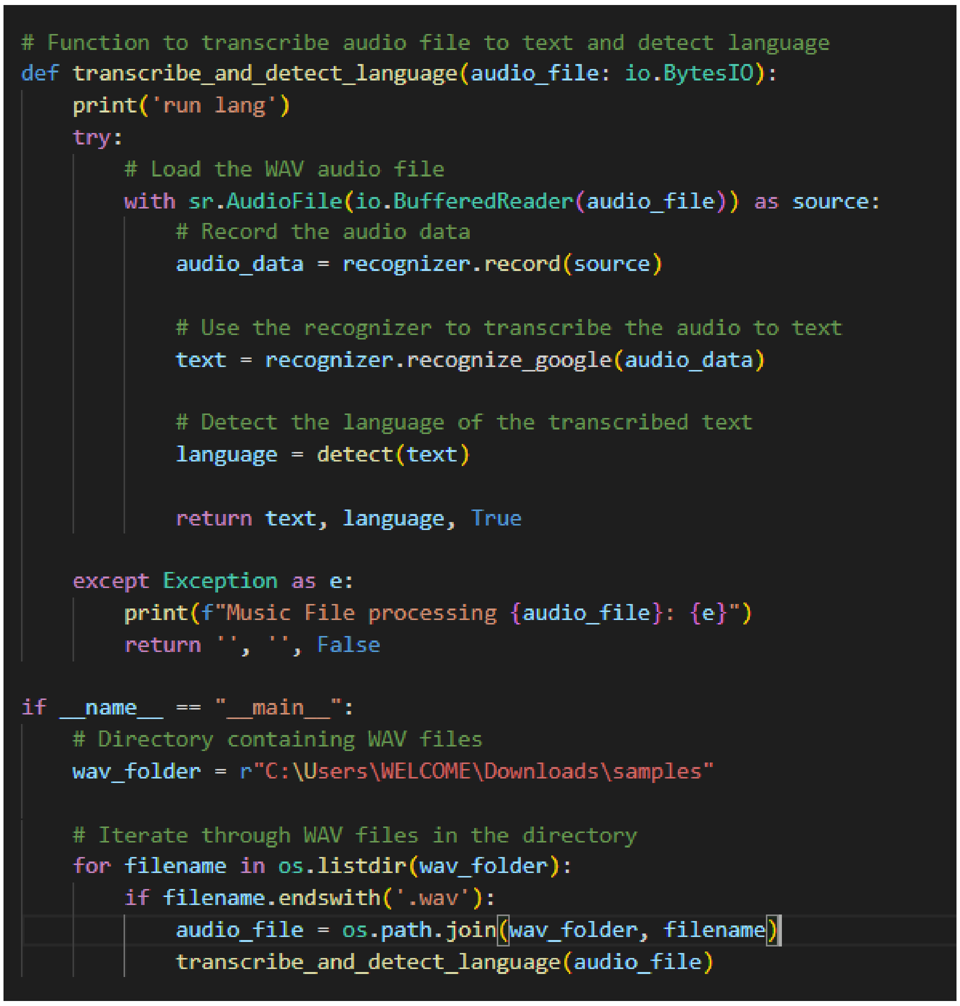
Advantage of using this method:

1. Simplicity and Ease of Use: The chosen libraries provide a straightforward way to transcribe audio and detect languages without the need for complex configurations.
2. Open Source and Free: All the libraries used in this script are open source and free to use, making them accessible to a wide range of users.
3. Integration: The provided script demonstrates an easy integration of multiple libraries to achieve a specific task, making it a good starting point for similar projects.

Alternative Methods:

1. Using Google Cloud Speech-to-Text API: Instead of using the `speech_recognition` library, one could directly use the Google Cloud Speech-to-Text API for audio transcription, which might offer more advanced features and accuracy.
2. Different Language Detection Libraries: While the langdetect library is simple and effective, alternatives such as `TextBlob` or `fastText` could be explored for language detection, each with its own strengths and weaknesses.
3. OpenAi WHisper

Example:



```python
# Function to transcribe audio file to text and detect language
def transcribe_and_detect_language(audio_file: io.BytesIO):
    print('run lang')
    try:
        # Load the WAV audio file
        with sr.AudioFile(io.BufferedReader(audio_file)) as source:
            # Record the audio data
            audio_data = recognizer.record(source)

            # Use the recognizer to transcribe the audio to text
            text = recognizer.recognize_google(audio_data)

            # Detect the language of the transcribed text
            language = detect(text)

            return text, language, True

    except Exception as e:
        print(f"Music File processing {audio_file}: {e}")
        return '', '', False

if __name__ == "__main__":
    # Directory containing WAV files
    wav_folder = r"C:\Users\WELCOME\Downloads\samples"

    # Iterate through WAV files in the directory
    for filename in os.listdir(wav_folder):
        if filename.endswith('.wav'):
            audio_file = os.path.join(wav_folder, filename)
            transcribe_and_detect_language(audio_file)
```

Fig2:VDC Example

4.2 Background Noise Detection Component (BGNDC):

Overview:

The BGNDC (Background Noise Detection Component) model utilizes a combination of NumPy, Librosa, Pandas, scikit-learn, and TensorFlow/Keras libraries for audio classification, specifically aimed at detecting background noise levels in audio files. It employs the cnn_extract_features function to extract Mel-frequency cepstral coefficients (MFCCs) from audio files, providing an effective representation of sound spectra. The model architecture, built with convolutional layers, max-pooling layers, and dense layers with dropout regularization, is compiled using the Adam optimizer and sparse categorical cross-entropy loss for multi-class classification. Training is conducted using the train_model function with training and validation data loaded through the load_data function. This approach capitalizes on the strengths of CNNs and MFCC features for audio classification tasks while benefiting from the ease of use and scalability offered by TensorFlow/Keras, with potential alternatives including different feature extraction techniques, model architectures, and optimization strategies to further enhance performance.

Libraries Used:

- **NumPy:** For numerical computations and array manipulation.
- **Librosa:** Used for audio loading and feature extraction.
- **Pandas:** For data manipulation.
- **scikit-learn:** Specifically, the train_test_split function for dataset splitting and LabelEncoder for label encoding.
- **TensorFlow/Keras:** For building and training the CNN model.

Feature Extraction

- **The cnn_extract_features function** extracts Mel-frequency cepstral coefficients (MFCCs) from audio files using Librosa. MFCCs are a representation of the short-term power spectrum of a sound, commonly used in audio processing tasks.

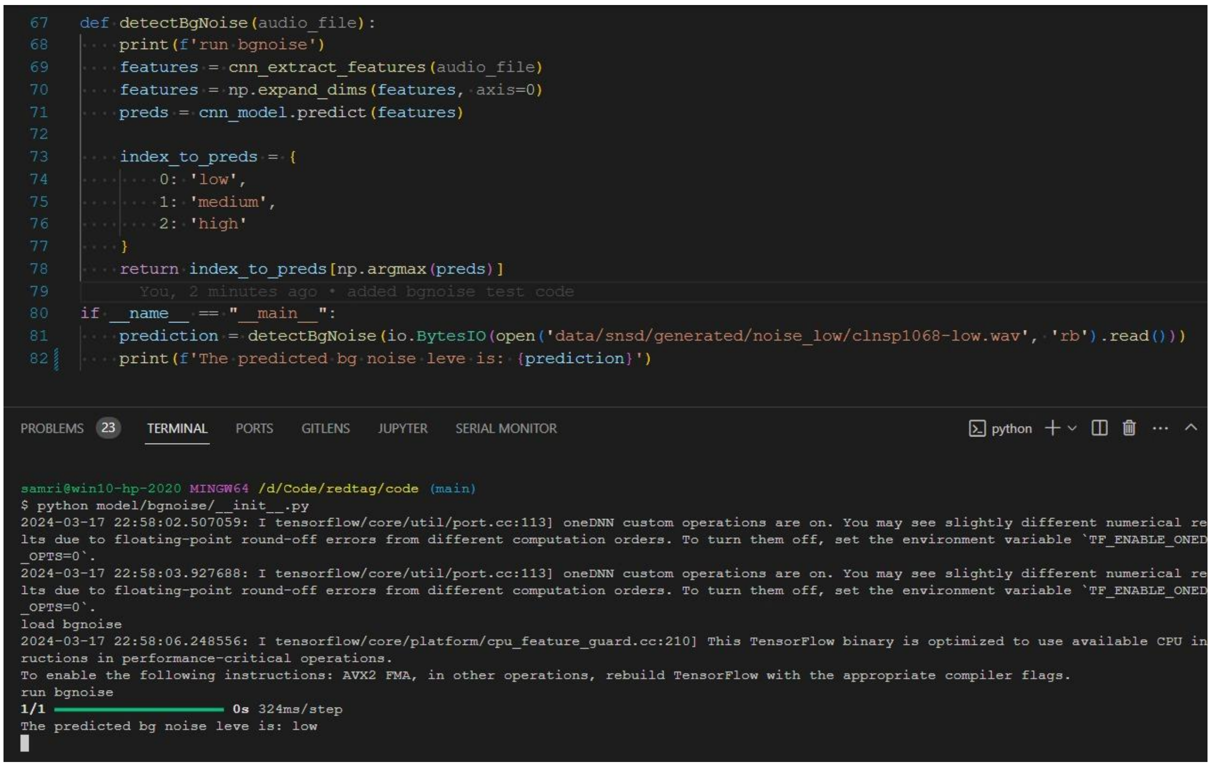
Advantages:

- Effective Feature Representation: MFCCs are widely used and effective for audio classification.
- CNNs for Audio Classification: CNNs excel at learning spatial and temporal patterns in audio data.
- TensorFlow/Keras Integration: TensorFlow/Keras provides a user-friendly interface for building and training neural networks, making experimentation and prototyping easy.
- Scalability: The code can be easily scaled for larger datasets or adapted for different audio classification tasks.

Alternative Methods:

- Alternative feature extraction techniques could include spectrogram-based features computed using Librosa or other audio processing libraries.
- Different model architectures like recurrent neural networks (RNNs) or hybrid architectures combining CNNs and RNNs could be explored.
- Hyperparameter tuning, regularization techniques, and different optimizers could improve model performance.
- In summary, the chosen approach leverages the strengths of CNNs and MFCC features for audio classification tasks while benefiting from the convenience and scalability provided by TensorFlow/Keras and other supporting libraries.

Example:



```python
def detectBgNoise(audio_file):
    print(f'run bgnoise')
    features = cnn_extract_features(audio_file)
    features = np.expand_dims(features, axis=0)
    preds = cnn_model.predict(features)

    index_to_preds = {
        0: 'low',
        1: 'medium',
        2: 'high'
    }
    return index_to_preds[np.argmax(preds)]
      You, 2 minutes ago • added bgnoise test code
if __name__ == "__main__":
    prediction = detectBgNoise(io.BytesIO(open('data/snsd/generated/noise_low/clnsp1068-low.wav', 'rb').read()))
    print(f'The predicted bg noise leve is: {prediction}')
```

```
samri@win10-hp-2020 MINGW64 /d/Code/redtag/code (main)
$ python model/bgnoise/__init__.py
2024-03-17 22:58:02.507059: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical re
lts due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONED
_OPTS=0`.
2024-03-17 22:58:03.927688: I tensorflow/core/util/port.cc:113] oneDNN custom operations are on. You may see slightly different numerical re
lts due to floating-point round-off errors from different computation orders. To turn them off, set the environment variable `TF_ENABLE_ONED
_OPTS=0`.
load bgnoise
2024-03-17 22:58:06.248556: I tensorflow/core/platform/cpu_feature_guard.cc:210] This TensorFlow binary is optimized to use available CPU in
ructions in performance-critical operations.
To enable the following instructions: AVX2 FMA, in other operations, rebuild TensorFlow with the appropriate compiler flags.
run bgnoise
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 324ms/step
The predicted bg noise leve is: low
```

Fig3:BGNDC Example

4.3 AI Detection Component (AIDC):

We have Used various methods in order to build AIDC. These are the following models we trained:

4.3.1 RawNet

Overview:

The RawNet model takes audio data as input and processes it through multiple layers, including SincConv layers for initial feature extraction, residual blocks for deeper processing, and fully connected layers for classification. RawNet offers a specialized and effective solution for processing audio signals, particularly in tasks related to voice recognition and synthesis. By leveraging its unique architecture and attention mechanisms, the model can accurately identify and distinguish between human and AI-generated voices, making it a valuable tool in voice-related applications.

Modules:

SincConv Module: Defines a custom convolutional layer called SincConv, which applies a set of band-pass filters to input audio signals. These filters are initialized based on Mel scale frequency bands.
Residual_block Module: Implements a residual block, a building block commonly used in deep neural networks to aid in training deeper models.
RawNet Module: Defines the main neural network architecture, RawNet, comprising multiple layers including SincConv layers, residual blocks, and fully connected layers.

Artifacts Collection:

- The SincConv module initializes band-pass filters based on the Mel scale, a perceptual scale of pitches of sounds.
- The network architecture captures various artifacts present in audio signals, such as frequency components and temporal patterns, through its convolutional and recurrent layers.
- By analyzing these artifacts, the model can learn to distinguish between different speakers, emotions, or even detect synthetic voices generated by AI algorithms.

Processing Based on Source of AI Generation:

- The RawNet architecture processes audio signals to identify features specific to AI-generated voices.
- Through its convolutional and recurrent layers, the model learns to differentiate between characteristics typical of human speech and those produced by AI algorithms.
- By training on a diverse dataset containing both human and AI-generated voices, the model can learn to detect subtle differences in speech patterns, intonations, and other features.

Working of RawNet:

- RawNet utilizes SincConv layers to capture frequency components of audio signals, followed by residual blocks to extract high-level features.
- Attention mechanisms are incorporated to focus on relevant parts of the audio sequence, enhancing the model's ability to identify distinctive features.
- The final layers include fully connected layers for classification into binary or multi-class categories, depending on the application.

Libraries Used:

- **PyTorch:** A deep learning framework for building and training neural networks, offering GPU acceleration and automatic differentiation.
- **NumPy**: For numerical computations and array manipulation.
- **os:** For interacting with the operating system, specifically for file operations.
- **io:** A module in Python's standard library providing tools for working with streams of data, including input/output operations like reading and writing to files, strings, and other types of data streams.

- **speech_recognition:** A Python library that provides easy access to several speech recognition APIs, allowing the conversion of speech to text
- **langdetect:** A Python library for language detection, capable of identifying the language of text data.
- **pydub:** A Python library for audio manipulation, providing functionalities like reading, writing, and modifying audio files in various formats.

Advantages:

- **Specialized Architecture:** RawNet is specifically designed for audio processing tasks, making it well-suited for voice recognition and synthesis applications.
- **Feature Extraction:** The model's architecture effectively captures relevant features from raw audio signals, allowing it to discriminate between natural and AI-generated voices.
- **Scalability:** With its modular design, RawNet can be adapted and extended to handle various audio processing tasks, including speaker identification, emotion recognition, and speech synthesis.

4.3.2 CNN

Overview:

Data Loading and Preprocessing
The load_and_preprocess_data function loads audio files from the specified directory, preprocesses them by converting to Mel spectrograms, and resizes them to a target shape.
- It organizes the data into input features (data) and corresponding labels (labels).

Model Construction and Training
- The CNN model is built using TensorFlow's Keras API, comprising convolutional and pooling layers followed by fully connected layers.
- The model is compiled with the Adam optimizer and categorical cross-entropy loss for multi-class classification.
- Training is performed using the fit method, specifying the training data, validation data, epochs, and batch size.

Libraries Used:

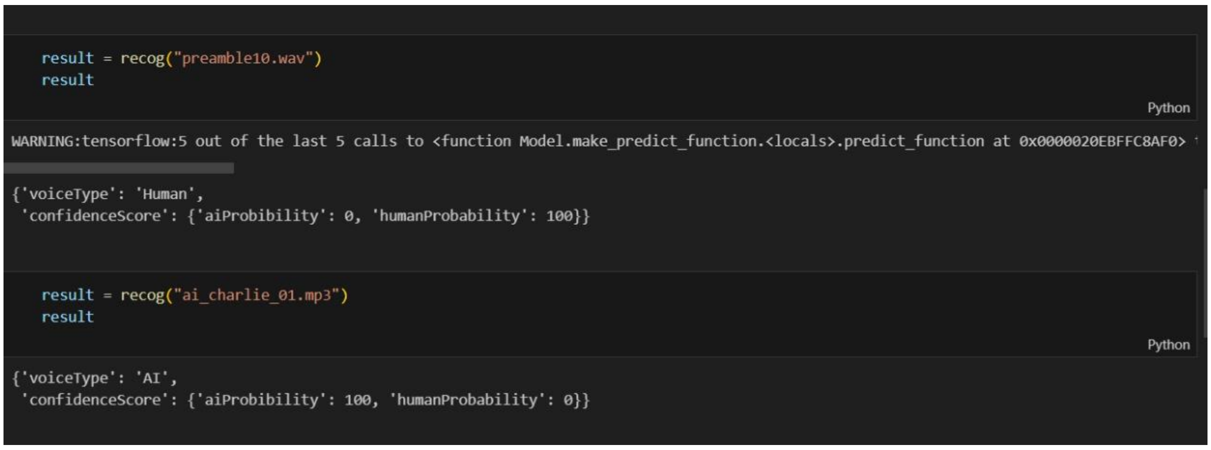
- **os**: For interacting with the operating system, specifically for file operations.
- **librosa:** Used for audio loading and preprocessing, including extracting Mel spectrograms.
- **NumPy**: For numerical computations and array manipulation.
- **TensorFlow**: For building and training the CNN model.
- **sklearn:** Specifically, the train_test_split function for splitting the dataset into training and testing sets.

- **TensorFlow Image (tf.image**): Specifically, the resize function for resizing Mel spectrograms.
- **tensorflow.keras:** For defining and training the CNN model.

Advantages:

- **Effective Feature Representation:** Mel spectrograms capture relevant audio features for classification tasks.
- **CNNs for Audio Classification:** CNNs excel at learning spatial and temporal patterns in data, making them suitable for audio classification.
- **Ease of Use with TensorFlow/Keras:** TensorFlow's Keras API provides a high-level interface for building and training neural networks, simplifying the development process.
- Integration with Other Libraries: The code seamlessly integrates with NumPy, scikit-learn, and librosa, enhancing flexibility and functionality.
- **Preprocessing Capabilities:** The preprocessing pipeline efficiently converts audio data into a format suitable for CNNs, enhancing model performance.

Example:



```
   result = recog("preamble10.wav")
   result
```

Python

```
WARNING:tensorflow:5 out of the last 5 calls to <function Model.make_predict_function.<locals>.predict_function at 0x0000020EBFFC8AF0>

{'voiceType': 'Human',
 'confidenceScore': {'aiProbibility': 0, 'humanProbability': 100}}
```

```
   result = recog("ai_charlie_01.mp3")
   result
```

Python

```
{'voiceType': 'AI',
 'confidenceScore': {'aiProbibility': 100, 'humanProbability': 0}}
```

Fig 4:AIDC Example

4.4 Emotional Detection Component (EDC):

Overview:

This process involves using the Whisper library to convert audio into text for emotional analysis. The transcribed text is prepared as a JSON request for the Google Cloud Natural Language API. After updating the JSON object with the text content, it's sent to the API. The sentiment score from the API response, reflecting the emotional intensity of the text, is then extracted and returned. This process enables automated speech-to-text conversion and subsequent emotional analysis of the transcribed content.

Libraries Used:

- **json:** This is a standard library in Python used for encoding and decoding JSON data. It provides functions to read JSON data from files and convert Python objects into JSON format.
- **subprocess:** Also a standard library in Python, the subprocess module allows you to spawn new processes, connect to their input/output/error pipes, and obtain their return codes. it's used to execute shell commands, specifically to run the curl command to make an HTTP request to the Google Cloud Natural Language API.

Working:

Audio to Text Conversion:
The process begins with converting audio data into text. This step is crucial as it allows us to analyze the content of the audio for emotional cues. Whisper library is being used to transcribe speech from an audio file, enabling applications such as automated speech-to-text conversion

Preparation of JSON Request:
The text data is then prepared to be sent as input to the Google Cloud Natural Language API. This involves creating a JSON object with the text content.

Updating JSON Object:
The object is updated within this JSON object with the transcribed text obtained from the audio. Then, the updated JSON data is written back to the file.

Sending Request to Google Cloud Natural Language API:
The updated JSON data, containing the text content, is sent to the Google Cloud Natural Language API. This command is executed using subprocess.getoutput().

Receiving and Parsing API Response:

The response from the API is received in JSON format. The score of the document sentiment is extracted from this response. The score of the document sentiment, indicating the overall emotional intensity of the text, is returned.

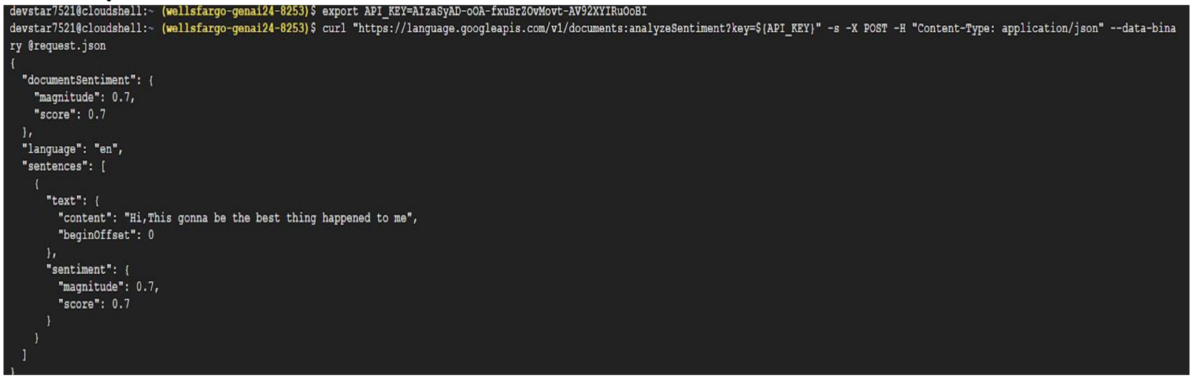
Advantages:

- **Ease of Use**: Whisper provides a straightforward interface for transcribing audio to text, making it accessible for developers.
- **Accuracy:** Depending on the quality of the audio and the model used, Whisper can offer good accuracy in speech recognition.
- **Customization**: Whisper may allow for model customization or fine-tuning, enabling adaptation to specific use cases or languages.

Alternative Methods:

- **Mozilla DeepSpeech:** An open-source speech-to-text engine developed by Mozilla, DeepSpeech provides accurate speech recognition and allows customization of models. It can be used offline and integrates well with Python.
- **CMU Sphinx:** Another open-source speech recognition system, CMU Sphinx, offers good accuracy and supports multiple programming languages. It can be run offline and is suitable for resource-constrained environments.
- **Wit.ai:** Wit.ai, a Facebook-owned platform, offers an API for natural language understanding, including speech recognition. It provides easy integration and supports multiple languages and platforms.

Example:



```
devstar7521@cloudshell:~ (wellsfargo-genai24-8253)$ export API_KEY=AIzaSyAD-oOA-fxuBrZOvMovt-AV92XYIRuOoBI
devstar7521@cloudshell:~ (wellsfargo-genai24-8253)$ curl "https://language.googleapis.com/v1/documents:analyzeSentiment?key=${API_KEY}" -s -X POST -H "Content-Type: application/json" --data-bina
ry @request.json
{
  "documentSentiment": {
    "magnitude": 0.7,
    "score": 0.7
  },
  "language": "en",
  "sentences": [
    {
      "text": {
        "content": "Hi,This gonna be the best thing happened to me",
        "beginOffset": 0
      },
      "sentiment": {
        "magnitude": 0.7,
        "score": 0.7
      }
    }
  ]
}
```

Fig 5:EDC Example