DATA MINING - CIA 2

23011101119 - S K KARISMA
23011101128 - SAMYUKTHA R G

1. ANALYSIS OF CHALLENGES IN SPEECH DATA:

- Background noise:
  - Environmental sounds like traffic, music, or other people talking can significantly degrade speech quality, making it difficult for a system to accurately transcribe speech.
- Homophones:
  - Words that sound the same but have different meanings (e.g., "to" and "too") can lead to recognition errors.
- Speaker variations:
  - Different accents, dialects, and speaking styles from various individuals can confuse speech recognition systems.
- Speech impairments:
  - People with speech disorders like stuttering or slurred speech can pose challenges for speech processing systems.

2. APPLICATION CHOSEN:

Keyword Spotting :

Keyword spotting/keyword recognition detects a word or short phrase within a stream of audio.

Usecase:

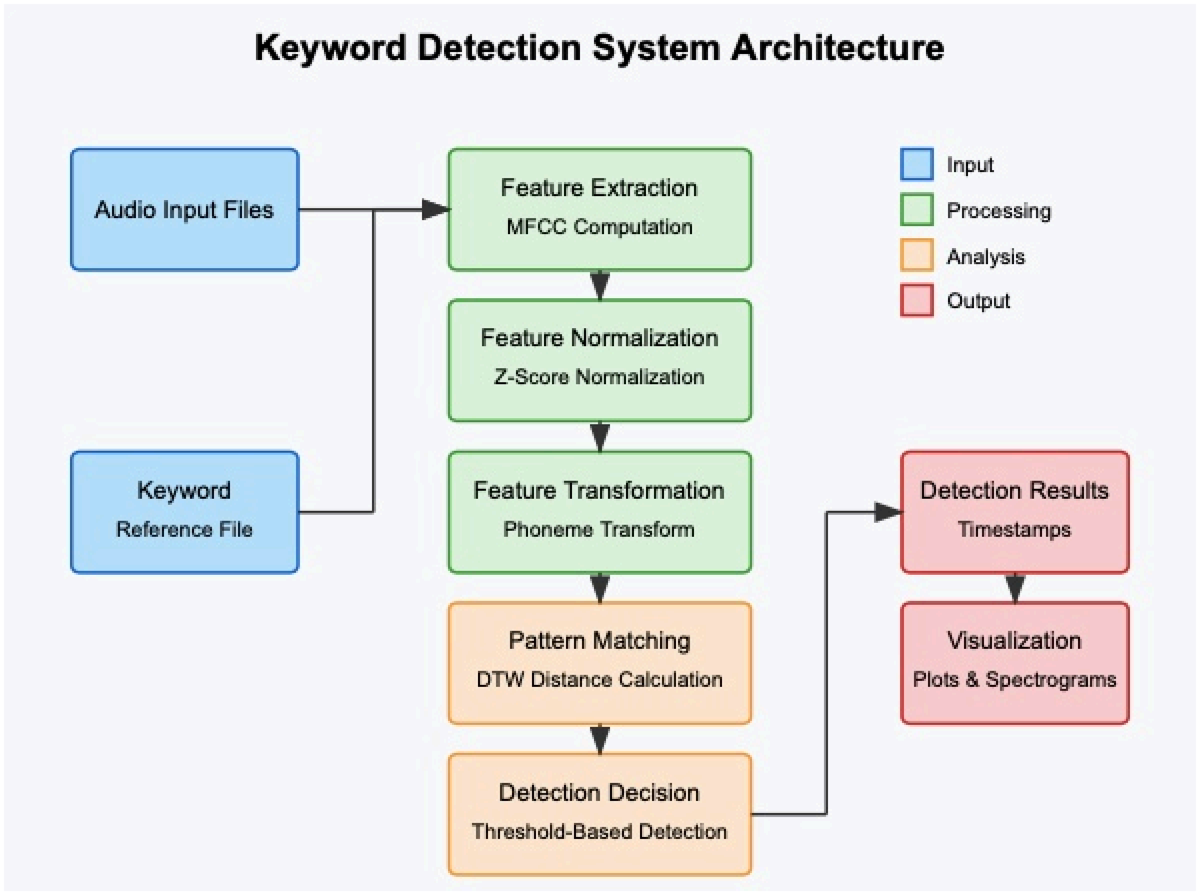
Crime & Threat Detection from Surveillance Audio

- Detect words like **"kill", "shoot", "bomb", "stab", "attack", "drug deal"** in phone calls, security recordings, or online communications.
- Used in **counter-terrorism and criminal investigations** to identify suspects or ongoing illegal activities.

Challenge chosen:

- **Accents** can present challenges to keyword spotting.
- These challenges stem from variations in pronunciation, rhythm, and intonation that can make it difficult for speakers to be heard clearly and for listeners to understand the message accurately.

3.ARCHITECTURE DIAGRAM:



4.MODULE DESCRIPTION:

Feature Extraction Module

- **Purpose**: Converts raw audio signals into meaningful feature representations
- **Techniques**:
    - **MFCC (Mel-frequency Cepstral Coefficients)**: The primary feature extraction method used, which models the human auditory perception system
    - **Librosa Library**: Used for audio processing and feature extraction with a 16kHz sampling rate
    - **Parameters**: 20 MFCC coefficients are extracted, representing spectral characteristics of the audio

Feature Normalization Module

- **Purpose**: Standardizes features to improve comparison robustness
- **Techniques**:
  - **Z-score normalization**: Calculates mean and standard deviation for each feature dimension and normalizes them to have zero mean and unit variance
  - **Mathematical Implementation**: (features - mean) / std
  - **Numerical stability**: Adds small epsilon (1e-10) to standard deviation to prevent division by zero

Feature Transformation Module

- **Purpose**: Applies domain-specific transformations to improve detection performance
- **Techniques**:
  - **Phoneme transformation**: A simple transformation that modifies the first MFCC coefficient (sets it to the mean value)
  - **Implementation**: This is a basic transformation that could be expanded to include more sophisticated phoneme-level processing

Pattern Matching Module

- **Purpose**: Compares keyword features with audio segment features to find matches
- **Techniques**:
  - **Dynamic Time Warping (DTW)**: Algorithm that measures similarity between temporal sequences that may vary in speed
  - **Cost Matrix**: Builds a matrix that represents alignment costs between sequences
  - **Distance Calculation**: Euclidean distance between feature vectors
  - **Normalization**: Returns distance normalized by combined sequence lengths for consistent thresholding

Detection Decision Module

- **Purpose**: Determines presence of keywords based on distance metrics
- **Techniques**:
  - **Threshold-based detection**: Compares DTW distances against a user-defined threshold value
  - **Temporal filtering**: Ensures detected keywords are at least 0.5 seconds apart to avoid duplicates

    - **Sliding window approach**: Steps through audio in configurable increments to balance processing speed and detection accuracy

Visualization Module

- **Purpose**: Provides visual analysis of detection results
- **Techniques**:
    - **Waveform plots**: Displays audio amplitude over time with detection markers
    - **Spectrogram visualization**: Shows frequency content over time with detection markers
    - **Distance plot**: Shows DTW distance metric throughout the audio file
    - **Threshold visualization**: Displays the detection threshold line for comparison

5. DATA SELECTION AND PREPROCESSING:

Data Selection

We generated our own data using Narakeet. First, we generated an audio file of our keyword we want to detect “kill” in American accent. Then we generated the sentence “They were planning to kill the witness before the trial” in American and British accent. We also generated the sentence “They started to shoot without warning”.

For keyword spotting, two types of audio data are needed:

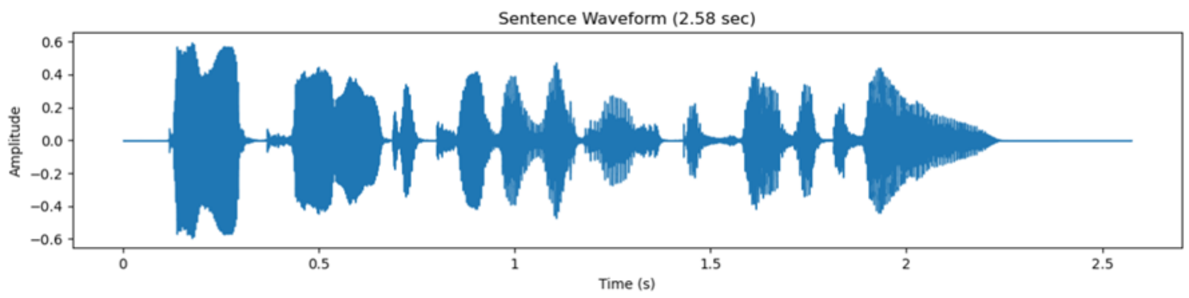
1. **Keyword Reference File**:
    - Single utterance of the target keyword (in this case, "kill.wav")
    - Should be clean and clearly pronounced
    - In this implementation, a pre-recorded file is used
2. **Audio Content for Detection**:
    - Longer audio files where the keyword may appear
    - Can contain various speech, noise, and potentially multiple instances of the keyword
    - Example file: "They were planning t (2).wav"

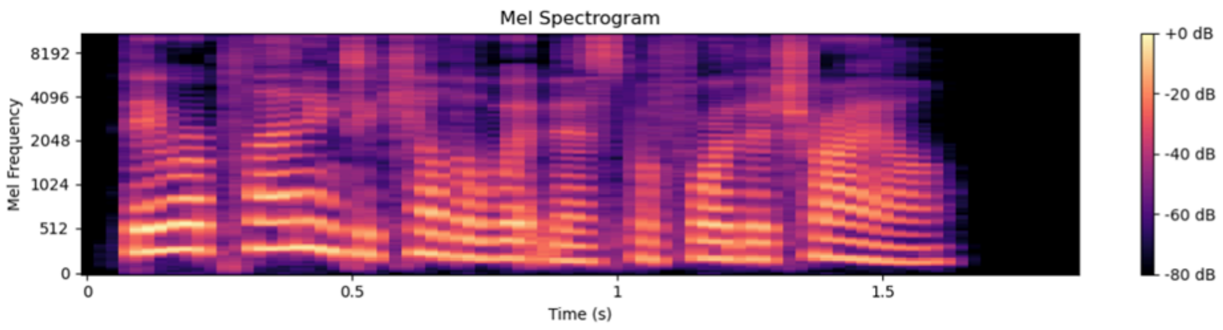
Preprocessing Steps

1. **Audio Loading**:
   - Audio files are loaded using librosa with a consistent 16kHz sampling rate
   - This resampling ensures consistent processing regardless of the original recording rate
2. **Feature Extraction**:
   - For both keyword and search audio:
     - Convert to mono if stereo
     - Extract 20 MFCC coefficients using librosa
     - Transpose the feature matrix for ease of frame-by-frame processing
3. **Feature Normalization**:
   - Z-score normalization is applied to standardize features
   - Each feature dimension is normalized separately
   - This accounts for variations in recording conditions
4. **Feature Transformation**:
   - A simple phoneme transform is applied (modifying the first MFCC coefficient)
   - This can be extended to more sophisticated transformations
5. **Sliding Windowing Strategy**:
   - The detection algorithm uses a sliding window approach
   - Window size is determined by a factor of the keyword length (window_size_factor=2)
   - Step size controls the stride between consecutive windows (step_size=5)
   - Smaller step sizes increase computational cost but improve detection accuracy

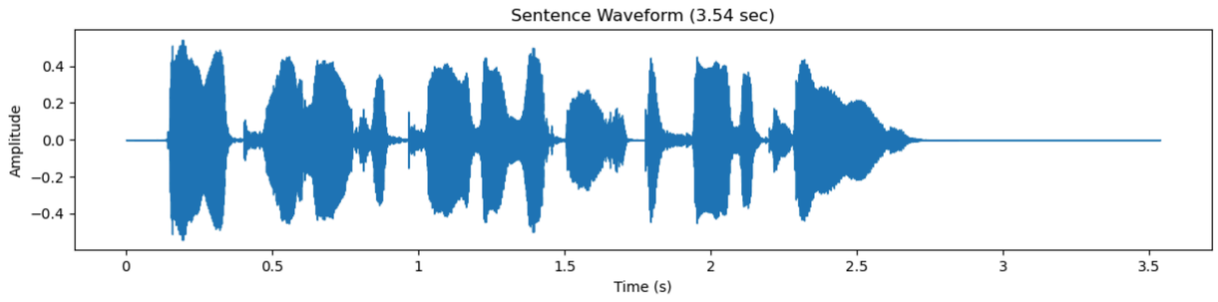
Visual Representation of data

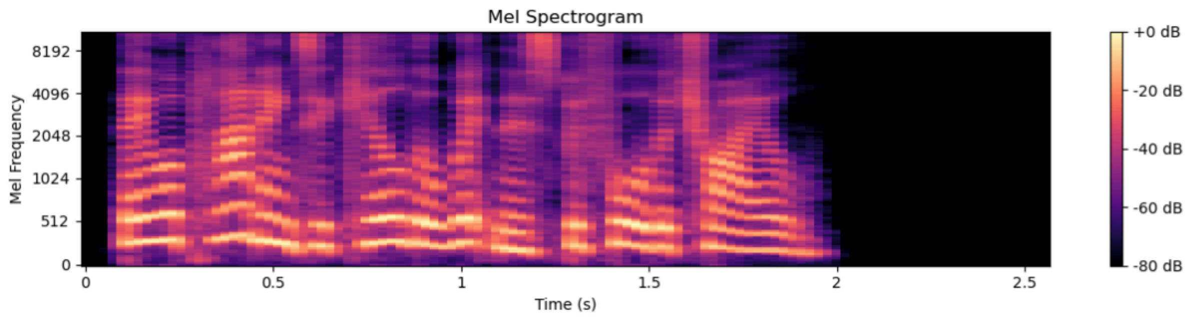
They were planning to kill the witness before the trial (American accent)



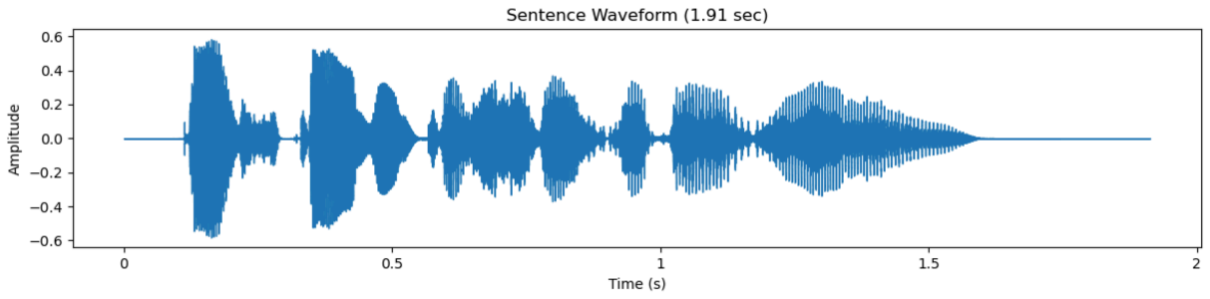


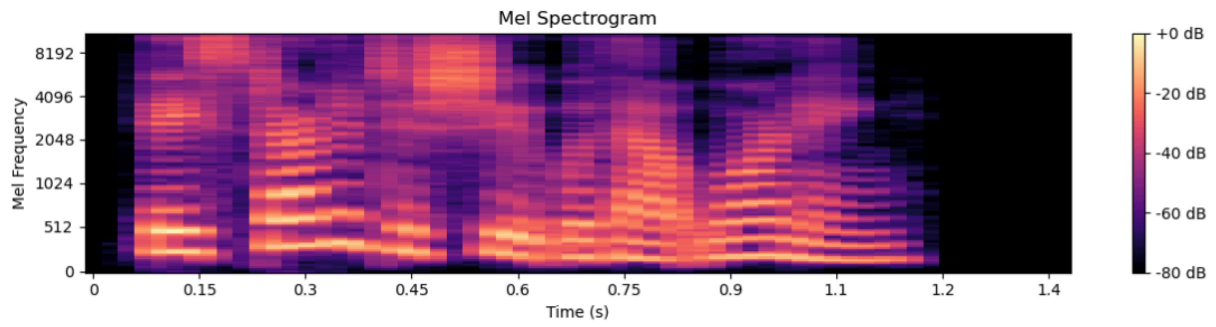
They were planning to kill the witness before the trial (British accent)





They started to shoot without warning (American accent)





Algorithm

The keyword detection system uses audio feature extraction and Dynamic Time Warping (DTW) to identify occurrences of a predefined keyword within longer audio recordings. The process involves comparing a reference keyword's acoustic features with sliding windows of features from the target audio.

Step 1: Keyword Reference Processing

1. Load the keyword reference audio file (e.g., "kill.wav") at a standardized 16kHz sampling rate.
2. Extract Mel-Frequency Cepstral Coefficients (MFCCs) from the keyword audio to represent its acoustic characteristics.
3. Apply Z-score normalization to standardize the feature values, making them more comparable across different recording conditions.
4. Apply a simple phoneme transformation that modifies the first MFCC coefficient to its mean value.
5. Store these processed features as the reference template for comparison.
6. Display the waveform and spectrogram of the keyword for visual reference.

Step 2: Target Audio Processing

1. Load the target audio file (where keyword instances are to be detected) at the same 16kHz sampling rate.
2. Extract the same type of MFCC features from the target audio using identical parameters (20 coefficients).
3. Apply the same normalization and phoneme transformation to ensure consistency with the keyword features.

4. Confirm that the audio file is longer than the keyword to allow for meaningful comparison.

Step 3: Sliding Window Feature Comparison

1. Determine an appropriate window size for comparison (default is twice the keyword length).
2. Establish a step size (default is 5) to control how the window slides through the target audio.
3. For each window position:
    - Extract a segment of features from the target audio of the same length as the keyword features.
    - Calculate the DTW distance between this segment and the keyword features.
    - Store this distance for later visualization and analysis.
    - Display a progress bar showing the search advancement through the audio.

Step 4: DTW Distance Calculation

1. For a given pair of feature sequences (keyword template and audio segment):
    - Normalize both sequences to make comparison more robust.
    - Create a cost matrix to track alignment costs.
    - For each cell in the matrix, calculate:
        - The Euclidean distance between corresponding feature vectors
        - The minimum cumulative cost from three possible previous steps (diagonal, horizontal, vertical)
        - The new cumulative cost by adding the current distance to the minimum previous cost
    - Return the final cumulative cost in the bottom-right cell, normalized by the sum of sequence lengths.

Step 5: Threshold-Based Detection

1. Compare each calculated DTW distance against a predefined threshold (e.g., 2.1).
2. When a distance falls below the threshold, mark that time point as a potential keyword detection.
3. Apply temporal filtering to avoid duplicate detections:

    - Only accept new detections if they are at least 0.5 seconds apart from previously detected instances.
4. Convert the frame indices of detections to timestamps in seconds based on the audio duration.

Step 6: Result Visualization and Analysis

1. Plot a graph of all DTW distances throughout the audio with the threshold line marked.
2. If no detections are found, suggest adjusting the threshold based on the minimum distance observed.
3. Generate a visual representation of detection results:
    - Display the audio waveform with vertical lines marking detected keyword instances.
    - Display the audio spectrogram with the same detection markers.
4. Return the list of timestamps where keyword instances were detected.

Computational Considerations

- The algorithm balances detection accuracy with computational efficiency through the step size parameter.
- Smaller step sizes provide more detection points but increase processing time.
- The window size factor controls how much context is considered around each potential match.
- The threshold parameter directly controls sensitivity—lower values increase detections but may introduce false positives.
- DTW inherently accommodates variations in speaking rate, making it robust for keyword spotting across different speakers.

This algorithm combines signal processing techniques, feature engineering, and dynamic programming to create a template-matching approach for keyword detection that is adaptable to various audio conditions and speaking styles.

CODE:

```python
import numpy as np
import matplotlib.pyplot as plt
from scipy.io import wavfile
import librosa
import librosa.display
import sys
import time
```

```python
def normalize_features(features):
    mean = np.mean(features, axis=0)
    std = np.std(features, axis=0) + 1e-10
    return (features - mean) / std
```

```python
def phoneme_transform(features):
    transformed = np.copy(features)
    transformed[:, 0] = np.mean(transformed[:, 0])  # Example transformation
    return transformed
```

```python
def extract_features(file_path):
    y, sr = librosa.load(file_path, sr=16000)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20).T
    mfccs = normalize_features(mfccs)
    mfccs = phoneme_transform(mfccs)
    return mfccs
```

```python
def dtw_distance(template, sample):
    if len(template) == 0 or len(sample) == 0:
        return float('inf')

    template = normalize_features(template)
    sample = normalize_features(sample)

    n, m = len(template), len(sample)
    cost_matrix = np.zeros((n + 1, m + 1))
    cost_matrix[0, 1:] = np.inf
    cost_matrix[1:, 0] = np.inf
    cost_matrix[0, 0] = 0

    for i in range(1, n + 1):
        for j in range(1, m + 1):
            cost = np.linalg.norm(template[i-1] - sample[j-1])
            prev_cost = min(cost_matrix[i-1, j], cost_matrix[i-1, j-1], cost_matrix[i, j-1])
            cost_matrix[i, j] = cost + prev_cost

    return cost_matrix[n, m] / (n + m)
```

```python
def detect_keyword_in_file(audio_file_path, keyword_file_path, threshold=0.4, window_size_factor=2,
    keyword_features = extract_features(keyword_file_path)
    keyword_len = len(keyword_features)

    y, sr = librosa.load(audio_file_path, sr=16000)
    audio_duration = librosa.get_duration(y=y, sr=sr)
    mfccs = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=20).T
    features = normalize_features(mfccs)
    features = phoneme_transform(features)

    print(f"Audio duration: {audio_duration:.2f}s | Feature shape: {features.shape}")
    window_size = min(keyword_len * window_size_factor, len(features))

    detections = []
    all_distances = []
    total_steps = range(0, len(features) - keyword_len + 1, step_size)
```

```python
    for idx, i in enumerate(total_steps):
        sequence = features[i:i+keyword_len]
        distance = dtw_distance(keyword_features, sequence)
        all_distances.append(distance)

        progress = (idx + 1) / len(total_steps)
        bar = '█' * int(progress * 40) + '-' * (40 - int(progress * 40))
        sys.stdout.write(f'\rScanning |{bar}| {int(progress * 100)}%')
        sys.stdout.flush()

        if distance < threshold:
            time_sec = i * audio_duration / len(features)
            if not detections or time_sec - detections[-1] > 0.5:
                detections.append(time_sec)
                print(f"\nKeyword detected at {time_sec:.2f}s (distance: {distance:.4f})")

    print()
```

```python
    # Plot distances
    plt.figure(figsize=(15, 5))
    plt.plot(np.linspace(0, audio_duration, len(all_distances)), all_distances)
    plt.axhline(y=threshold, color='r', linestyle='--', label=f'Threshold ({threshold})')
    plt.title('DTW Distances')
    plt.xlabel('Time (s)')
    plt.ylabel('Distance')
    plt.legend()
    plt.show()

    return detections
```

```python
def visualize_sentence_waveform(audio_file_path):
    y, sr = librosa.load(audio_file_path, sr=16000)
    duration = librosa.get_duration(y=y, sr=sr)
    plt.figure(figsize=(12, 6))

    # Waveform
    plt.subplot(2, 1, 1)
    librosa.display.waveshow(y, sr=sr)
    plt.title(f'Sentence Waveform ({duration:.2f} sec)')
    plt.xlabel('Time (s)')
    plt.ylabel('Amplitude')

    # Spectrogram
    plt.subplot(2, 1, 2)
    S = librosa.feature.melspectrogram(y=y, sr=sr)
    librosa.display.specshow(librosa.power_to_db(S, ref=np.max), y_axis='mel', x_axis='time')
    plt.colorbar(format='%+2.0f dB')
    plt.title('Mel Spectrogram')
    plt.xlabel('Time (s)')
    plt.ylabel('Mel Frequency')
    plt.tight_layout()
    plt.show()
```

```python
def visualize_detections(audio_file_path, detections):
    y, sr = librosa.load(audio_file_path, sr=16000)

    plt.figure(figsize=(15, 10))
    plt.subplot(2, 1, 1)
    librosa.display.waveshow(y, sr=sr)
    plt.title('Waveform with Keyword Detections')
    for detection in detections:
        plt.axvline(x=detection, color='r', linestyle='--')

    plt.subplot(2, 1, 2)
    S = librosa.feature.melspectrogram(y=y, sr=sr)
    librosa.display.specshow(librosa.power_to_db(S, ref=np.max), y_axis='mel', x_axis='time')
    plt.title('Spectrogram with Keyword Detections')
    for detection in detections:
        plt.axvline(x=detection, color='r', linestyle='--')

    plt.tight_layout()
    plt.show()
```

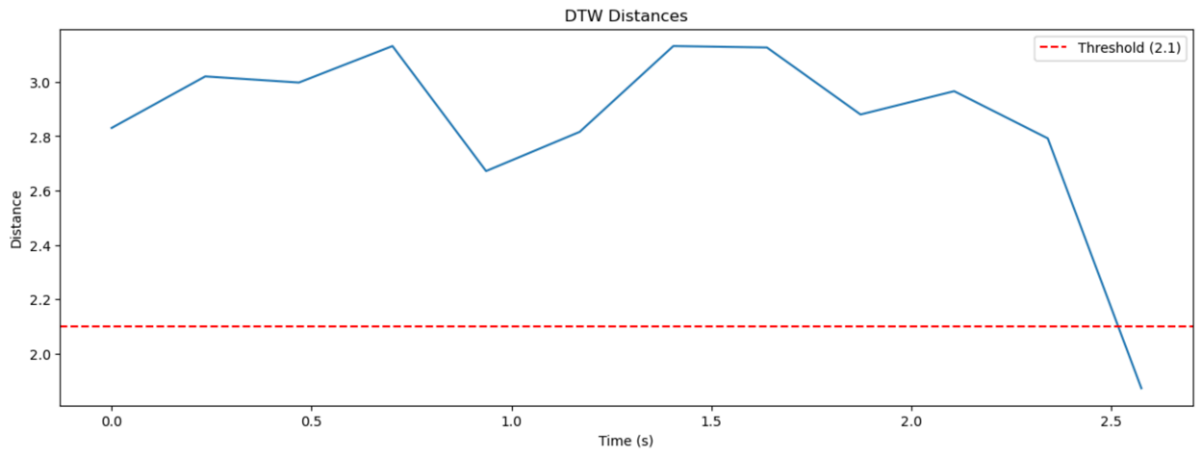
```python
sentence_path = r"C:\Users\hp\Downloads\They were planning t.wav"
visualize_sentence_waveform(sentence_path)

detections = detect_keyword_in_file(sentence_path,r"C:\Users\hp\Downloads\kill.wav", threshold=2.1)
print(detections)
visualize_detections(sentence_path, detections)
```

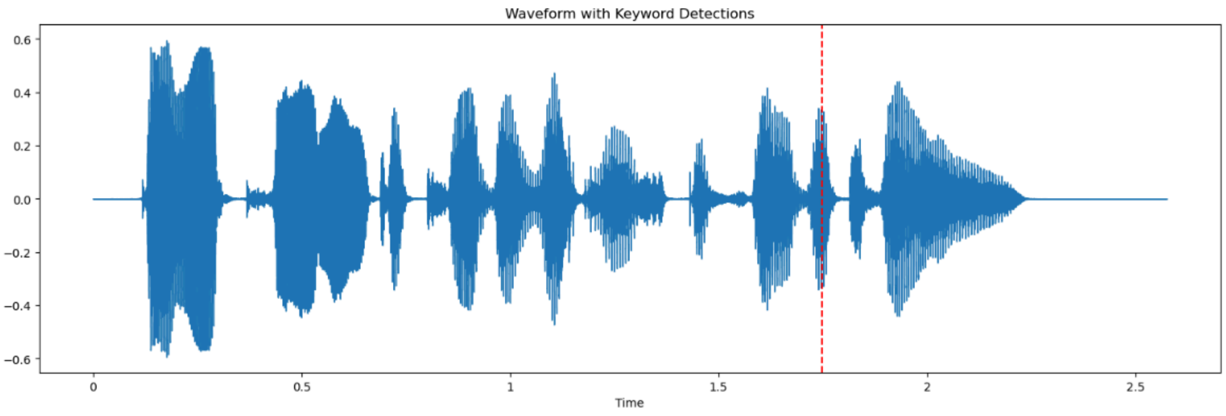
We set threshold for keyword detection using trial and error as 2.1.

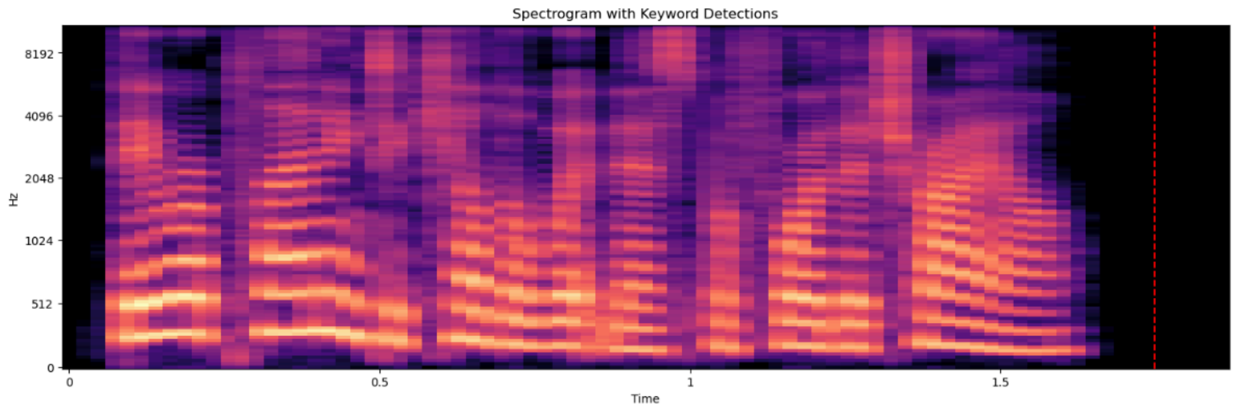
They were planning to kill the witness before trial (American accent)

```
Audio duration: 2.58s | Feature shape: (81, 20)
Scanning |██████████████████████████████████████████| 100%
Keyword detected at 1.75s (distance: 1.8725)
```



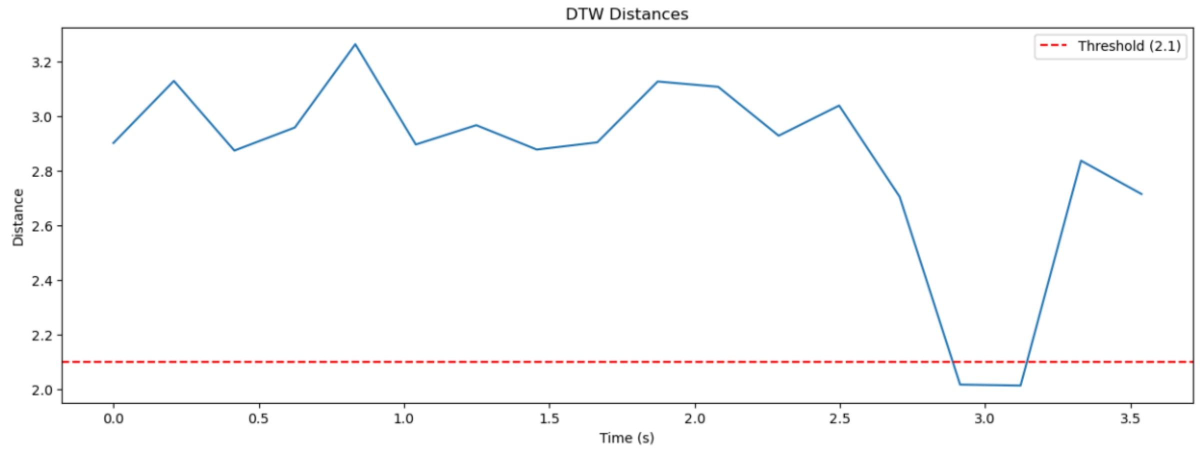
```
[1.748456790123457]
```



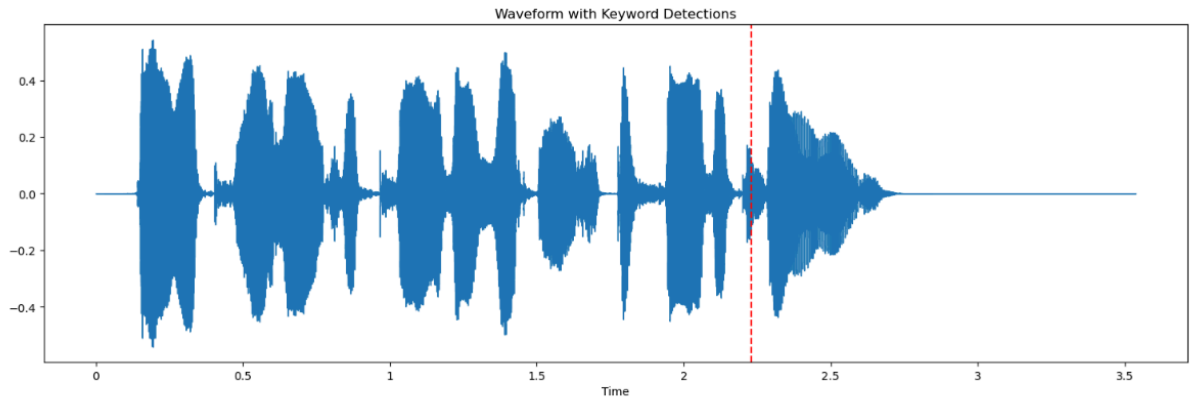


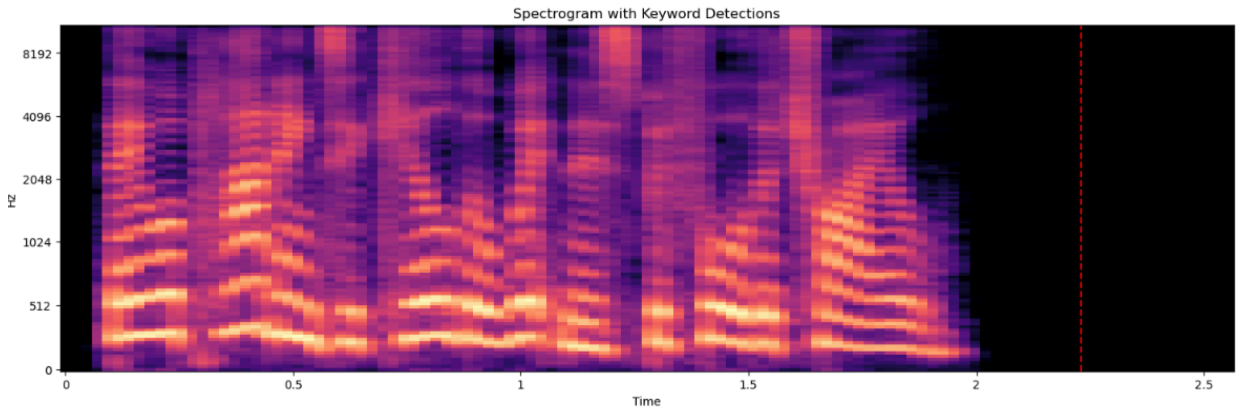
They were planning to kill the witness before trial (British accent)

```
Audio duration: 3.54s | Feature shape: (111, 20)
Scanning |██████████████████████████-------| 83%
Keyword detected at 2.23s (distance: 2.0174)
Scanning |████████████████████████████████| 100%
```



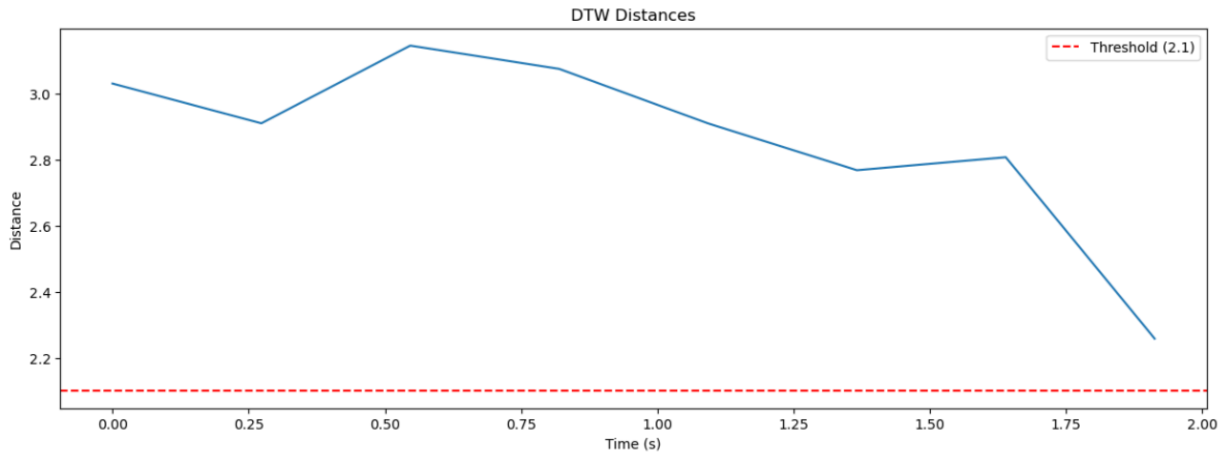
```
[2.230855855855856]
```





They started to shoot without warning (American accent)

```
Audio duration: 1.91s | Feature shape: (60, 20)
Scanning |████████████████████████████████████████| 100%
```



```
[]
```

No keyword detected, so empty list is displayed

6. PERFORMANCE EVALUATION:

EVALUATION METRICS

- **Threshold Sensitivity:**
  - The system uses a distance threshold (2.1) to determine keyword presence
  - Lowering the threshold increases sensitivity but may lead to false positives
  - Raising the threshold reduces false positives but may miss actual occurrences
- **Detection Accuracy:**
  - True Positives: Correctly identified keyword instances
  - False Positives: Incorrectly identified matches
  - False Negatives: Missed keyword instances
  - The system provides visualization to manually verify detection accuracy
- **Computational Efficiency:**
  - The DTW algorithm has O(n*m) time complexity where n and m are sequence lengths
  - The step_size parameter balances computational efficiency and detection accuracy
  - Progress bar indicates processing time for longer audio files
- **Robustness Analysis:**
  - Z-score normalization helps handle different recording conditions

    - The simple phoneme transform shows potential for improved phoneme-level matching
    - DTW algorithm inherently handles speaking rate variations

PERFORMANCE CONSIDERATIONS

- **Window Size Effect:**
    - The window_size_factor (default=2) controls the size of the comparison window
    - Larger windows capture more context but increase processing time
    - Smaller windows are faster but may miss contextual information
- **Step Size Impact:**
    - Smaller step sizes (default=5) provide more granular matching but increase computation
    - Larger steps speed up processing but may skip potential matches
- **Feature Selection:**
    - The current implementation uses 20 MFCC coefficients
    - More coefficients could capture detailed spectral information
    - Fewer coefficients would reduce computational load but might decrease accuracy
- **Threshold Selection:**
    - The code suggests a threshold of 2.1, which is relatively high
    - The system provides feedback on minimum distance encountered to help adjust threshold

7. CONCLUSION AND FUTURE WORK

Conclusions

1. **DTW-Based Approach**:
    - The implementation uses Dynamic Time Warping for template matching
    - This approach handles temporal variations in speech effectively
    - The algorithm is relatively simple yet effective for single keyword detection
2. **Feature Representation**:
    - MFCCs provide a compact, perceptually relevant representation
    - Normalization improves robustness to recording conditions
    - The simple phoneme transform shows potential for improvement
3. **Visualization Tools**:

    - The system includes comprehensive visualization tools
    - Waveform and spectrogram plots help verify detection results
    - DTW distance plots help in threshold selection and performance analysis
4. **Limitations**:
    - The system requires a clean reference recording of the keyword
    - Performance may degrade in noisy environments
    - The fixed threshold approach may not be optimal for all keywords or conditions

Future Work

1. **Advanced Feature Engineering**:
    - Implement more sophisticated phoneme transformations
    - Explore additional acoustic features (spectral flux, zero-crossing rate)
    - Investigate delta and delta-delta coefficients for capturing dynamics
2. **Adaptive Thresholding**:
    - Develop methods for automatic threshold selection
    - Implement keyword-specific thresholds based on phonetic complexity
    - Consider adaptive thresholds based on noise conditions
3. **Machine Learning Extensions**:
    - Integrate supervised learning for improved classification
    - Implement a Hidden Markov Model (HMM) for better temporal modeling
    - Explore deep learning approaches (CNNs, RNNs) for feature extraction
4. **Multi-Keyword Detection**:
    - Extend the system to detect multiple keywords simultaneously
    - Implement efficient algorithms for multi-pattern matching
    - Develop a keyword spotting language model
5. **Noise Robustness**:
    - Implement noise reduction preprocessing
    - Explore robust feature extraction methods
    - Develop environment-adaptive normalization techniques
6. **Real-time Processing**:
    - Optimize the algorithm for real-time applications
    - Implement buffer-based processing for streaming audio
    - Explore parallel processing for faster computation
7. **User Interface Improvements**:
    - Develop a more intuitive threshold selection interface
    - Add options for batch processing multiple files
    - Implement export functionality for detection results