Implementation of MFCC Feature Extraction and Comparison with Python Package Functions

Ⅰ.Introduction

1. **Background and Applications of MFCC**
   Mel Frequency Cepstral Coefficients (MFCC) are widely used in fields like speech recognition and speaker identification. MFCC is based on the auditory perception of human ears, which are more sensitive to certain frequency ranges, particularly between 200Hz and 5000Hz, where speech clarity is most affected. The human ear has a phenomenon known as "masking effect," where louder sounds can mask quieter ones, making them harder to perceive. Generally, low-frequency sounds can mask high-frequency sounds more easily than the reverse, and the critical bandwidth is narrower at lower frequencies compared to higher frequencies.
   To mimic this behavior, a set of bandpass filters are arranged from low to high frequencies, more densely in the low frequencies and sparser in the high frequencies, based on the critical bandwidth. The signal is filtered through these filters, and the energy output from each filter is taken as a basic feature of the signal. After further processing, these features can be used as input for speech recognition systems. MFCC is not dependent on the signal's properties, making no assumptions or limitations about the input. It leverages auditory models, thus offering better robustness and recognition performance under low signal-to-noise ratio conditions compared to other methods like Linear Predictive Cepstral Coefficients (LPCC).

- MFCC
  - MFCC Extraction Process
    - Pre-emphasis: Amplifying high-frequency components to balance the spectral tilt.
    - Short-Time Fourier Transform (STFT): Segmenting the signal into short frames and applying Fourier transformation.
    - Mel-filterbank: Applying a set of Mel-scale triangular filters.
    - Logarithm: Taking the log of the filterbank outputs.
    - Discrete Cosine Transform (DCT): Reducing data dimensionality and obtaining the cepstral coefficients.
    - Selecting MFCCs: Usually, the first 12 coefficients are selected, plus one energy term, making it 13 in total.
- MFCC Output
  - Typically, 12 coefficients are used, along with the frame's energy, yielding 13 coefficients.

    - The 13 coefficients are computed for each frame of the signal, and then their first and second-order derivatives (deltas) are calculated.
        - Forward Difference

$$\Delta x[n] = x[n+1] - x[n]$$

        - Backward Difference

$$\Delta x[n] = x[n] - x[n-1]$$

        - Central Difference (minimizes error)

$$\Delta x[n] = (x[n+1] - x[n-1])/2$$

      Where x[n] represents the 13 coefficients of frame n.
    - The MFCC output is a 2D array (frames x coefficients), which can be visualized as a heatmap.
- Advantages and Disadvantages of MFCC
    - Advantages
        - Compared to Mel spectrograms, MFCC uses fewer data points while capturing important spectral information. Mel spectrograms typically use 80 filters, whereas MFCC typically uses 39 coefficients (13 base + 13 delta + 13 delta-delta).
        - The coefficients are less correlated, providing better discriminative power.
        - Extracts fundamental frequency and formant information while filtering out irrelevant data.
        - Performs well in models like Gaussian Mixture Models (GMM).
    - Disadvantages
        - More computationally intensive compared to Mel spectrograms, since it is computed based on them.
        - Less robust to additive noise.
        - Heavily engineered features, leading to higher risk of overfitting (empirical risk).
        - Not suitable for speech synthesis as there is no straightforward inverse transformation to reconstruct the signal from MFCC.

2. **Experiment Requirements**
   The task is to implement MFCC feature extraction for a given speech signal and provide detailed code annotations. The results of the custom implementation will be compared with the outputs from Python's MFCC library functions. Differences between the two implementations will be analyzed and possible reasons for any discrepancies will be discussed.

Ⅱ. Theoretical Background

1. **Pre-emphasis**
   Pre-emphasis aims to boost the high-frequency components of the signal, flattening its spectrum and ensuring that the signal maintains the same signal-to-noise ratio across the entire frequency range, from low to high frequencies. This process also helps to remove the effects of the vocal cords and lips, compensating for the suppression of high frequencies

caused by the speech production system. The pre-emphasis process can be thought of as passing the speech signal through a high-pass filter:

$$H(z) = 1 - \mu z^{-1}$$

In the time domain, it is expressed as:

$$y(n) = x(n) - \alpha \cdot x(n-1)$$

Where α is typically set between 0.95 and 0.97 to balance the energy between high and low frequencies.

2. **Windowing**
   To mitigate spectral leakage, each frame of the signal is multiplied by a window function, such as the Hamming window. Assuming that the signal after framing is S(n),where n=0,1,...,N-1 , and N is the number of samples in a frame, the signal after applying the Hamming window becomes:

$$S'(n) = S(n) \times W(n)$$

   where W(n) is the window function.

3. **Short-Time Fourier Transform (STFT)**
   The short-time Fourier transform is used to analyze the time-frequency characteristics of non-stationary signals, capturing dynamic frequency changes in the signal. By applying the Fourier transform to each windowed segment of the signal, STFT provides a joint representation of time and frequency. The STFT is defined as:

$$X(t, f) = \sum_{n=-\infty}^{\infty} x[n] w[n-t] e^{-j2\pi f n}$$

4. **Mel Filter Bank**

- Characteristics
  - A Mel filter bank consists of multiple triangular filters. Each filter has a maximum value of 1 at its center frequency and gradually tapers off to 0 on either side. Each filter covers a certain frequency range, and adjacent filters have overlapping boundaries where their responses meet at zero.
  - For frequencies below 1000 Hz, the center frequencies of the filters are distributed on a linear scale. For frequencies above 1000 Hz, the center frequencies follow a logarithmic scale.
  - The Mel scale is a scale of pitches judged by listeners to be equal in distance from one another, making it more aligned with how humans perceive sound.
- Function
  The Mel filter bank simulates the human ear's sensitivity to different frequencies by weighting the energy in various frequency bands. More filters are allocated to the low-frequency regions, reflecting the human ear's higher sensitivity to low frequencies, while fewer filters are used in the high-frequency regions, reflecting lower sensitivity.

5. **Logarithm**
   Human perception of loudness is non-linear. For a sound to be perceived as twice as loud, its energy must increase by a factor of 8. This means that when a sound is already sufficiently loud, further increases in energy are less noticeable. The logarithmic operation compresses the dynamic range of the signal, making the features more aligned with human auditory

perception. It also enables the use of techniques like cepstral mean subtraction (CMS), which helps normalize differences between different communication channels.

6. **Discrete Cosine Transform (DCT)**

- The calculation of the cepstrum involves applying the discrete Fourier transform (DFT) to the signal, taking the logarithm, and then performing the inverse discrete Fourier transform. In the case of MFCC, the last step is replaced with a discrete cosine transform (DCT).
- Features
    - Use of only cosine functions
      Unlike DFT, which uses both sine and cosine functions, DCT relies solely on cosine functions to represent the signal. As a result, DCT generates real-valued coefficients.
    - Energy compression
      DCT has the property of concentrating most of the signal's energy in the lower frequency components, meaning that most of the important information is compressed into the first few coefficients. This makes DCT useful for compression and reducing redundancy in data.
    - Boundary characteristics
      DCT assumes that the signal is symmetric at the boundaries, making it well-suited for processing finite-length signals and reducing spectral leakage.
- Mathematical Definition

$$X[k] = \sum_{n=0}^{N-1} x[n] \cdot \cos\left(\frac{\pi}{N} \cdot \left(n + \frac{1}{2}\right) \cdot k\right), \quad k = 0, 1, \ldots, N-1$$

  Where:
    - x[n] is the input signal.
    - X[k] is the k-th DCT coefficient.
    - N is the length of the signal.
- Advantages
    - Simplified version of the discrete Fourier transform (DFT).
    - Results in real numbers.
    - Decouples overlapping weights between different Mel filter banks, making the extracted features more independent from each other, which is beneficial for machine learning tasks.
    - Reduces dimensionality by transforming the log-power spectrum into MFCC features.

7. **Dynamic Feature Extraction**
   Dynamic feature extraction refers to the process of extracting meaningful features from time-series data or dynamic signals. In the case of MFCC, the first and second-order derivatives (deltas) are calculated over time to capture the temporal evolution of the speech signal.

8. **Feature Transformation**
   Feature transformation involves transforming the extracted features to improve the usability and performance of the model. For example, after computing the Mel frequency energy spectrum, further processing can yield the compact representation of the cepstral coefficients.

Ⅲ.Experimental Steps

1. **Data Preparation**

- Recording Materials
  - Long recording

> 在全球范围内，各国正努力应对气候变化和环境问题。最近，许多科技公司和研究机构开始利用先进的技术推动可持续发展。

  - Short recording

> 你好

- Used speech signal data
  - Clear recording
    - Length: 14s
    - Recording environment: Quiet, spacious room
  - Noisy recording
    - Length: 13s
    - Recording environment: Spacious room with piano music playing
  - Short recording
    - Length: 1s
    - Recording environment: Quiet, spacious room

2. **MFCC Self Implementation Process**

- Pre-emphasis
  Pre-emphasis is performed to attenuate low-frequency components and enhance high-frequency components, aiming to balance the spectrum and make subsequent feature extraction more effective.

```
def pre_emphasis(signal, alpha=0.97):
  # np.append keeps the first sample unchanged, applying the formula y(t) = x(t) - α*x(t-1) to subsequent samples
  emphasized_signal = np.append(signal[0], signal[1:] - alpha * signal[:-1])
  return emphasized_signal
```

- Framing
  The speech signal is divided into small segments (frames), each lasting for a short period, during which the signal can be assumed to be quasi-stationary, facilitating frequency domain analysis.

```
  def framing(signal, frame_size, frame_stride, sample_rate):
    # Frame length and step: frame_size = 0.025 seconds, frame_stride = 0.01 seconds
    frame_length = int(frame_size * sample_rate)
    frame_step = int(frame_stride * sample_rate)
    signal_length = len(signal)
```

```
    # Calculate the required number of frames
    num_frames = int(np.ceil(float(np.abs(signal_length - frame_length)) /
frame_step)) + 1

    # Zero-pad the signal to ensure all frames have the same length
    pad_signal_length = num_frames * frame_step + frame_length
    z = np.zeros((pad_signal_length - signal_length))
    pad_signal = np.append(signal, z)

    # Create frame indices
    indices = np.tile(np.arange(0, frame_length), (num_frames, 1)) +
np.tile(np.arange(0, num_frames * frame_step, frame_step), (frame_length,
1)).T
    frames = pad_signal[indices.astype(np.int32, copy=False)]
    return frames
```

- Windowing
  Apply a window function (Hamming window in this experiment) to each frame to reduce edge effects and smooth transitions in the signal.

```
def windowing(frames, frame_length):
  # Apply the Hamming window to each frame, causing the signal to gradually
decay at both ends
  frames *= np.hamming(frame_length)
  return frames
```

- Short-Time Fourier Transform (STFT)
  STFT is used to convert the time-domain signal into the frequency domain, calculating the power spectrum (frequency domain energy distribution) for each frame.

```
def stft(frames, NFFT):
  # Calculate the magnitude of the Fourier transform for each frame
  mag_frames = np.absolute(np.fft.rfft(frames, NFFT))
  # Calculate the power spectrum (normalized square)
  pow_frames = ((1.0 / NFFT) * (mag_frames ** 2))
  return pow_frames
```

- Mel Filter Banks
  The power spectrum is passed through a Mel-scale filter bank, simulating the nonlinear perception of frequencies by the human ear. The filter bank uses the Mel scale, with linear distribution at low frequencies and logarithmic distribution at high frequencies.

```
def mel_filter_banks(pow_frames, nfilt, NFFT, sample_rate):
  # Convert frequencies to the Mel scale
  low_freq_mel = 0
  high_freq_mel = (2595 * np.log10(1 + (sample_rate / 2) / 700))
  mel_points = np.linspace(low_freq_mel, high_freq_mel, nfilt + 2)  # Create
equally spaced points on the Mel scale
  hz_points = (700 * (10**(mel_points / 2595) - 1))  # Convert Mel scale
points back to linear frequencies
  bin = np.floor((NFFT + 1) * hz_points / sample_rate).astype(int)  #
Calculate the corresponding FFT bin values for each frequency point
```

```
  # Create the filter bank
  fbank = np.zeros((nfilt, int(np.floor(NFFT / 2 + 1))))
  for m in range(1, nfilt + 1):
      f_m_minus = bin[m - 1]   # Left frequency point
      f_m = bin[m]             # Middle frequency point
      f_m_plus = bin[m + 1]    # Right frequency point

      # Left rise
      for k in range(f_m_minus, f_m):
          fbank[m - 1, k] = (k - bin[m - 1]) / (bin[m] - bin[m - 1])
      # Right fall
      for k in range(f_m, f_m_plus):
          fbank[m - 1, k] = (bin[m + 1] - k) / (bin[m + 1] - bin[m])

  # Calculate filter bank output
  filter_banks = np.dot(pow_frames, fbank.T)
  filter_banks = np.where(filter_banks == 0, np.finfo(float).eps,
filter_banks)  # Avoid log(0)
  filter_banks = 20 * np.log10(filter_banks)  # Convert to dB
  return filter_banks
```

- Discrete Cosine Transform (DCT)
  DCT is used to transform the output of the Mel filter bank into Mel Frequency Cepstral Coefficients (MFCCs). The first 13 coefficients are retained as the final features.

```
def mfcc(filter_banks, num_ceps):
  # Apply DCT to each frame and retain the first num_ceps coefficients
  mfcc = scipy.fftpack.dct(filter_banks, type=2, axis=1, norm='ortho')[:,
:num_ceps]
  return mfcc
```

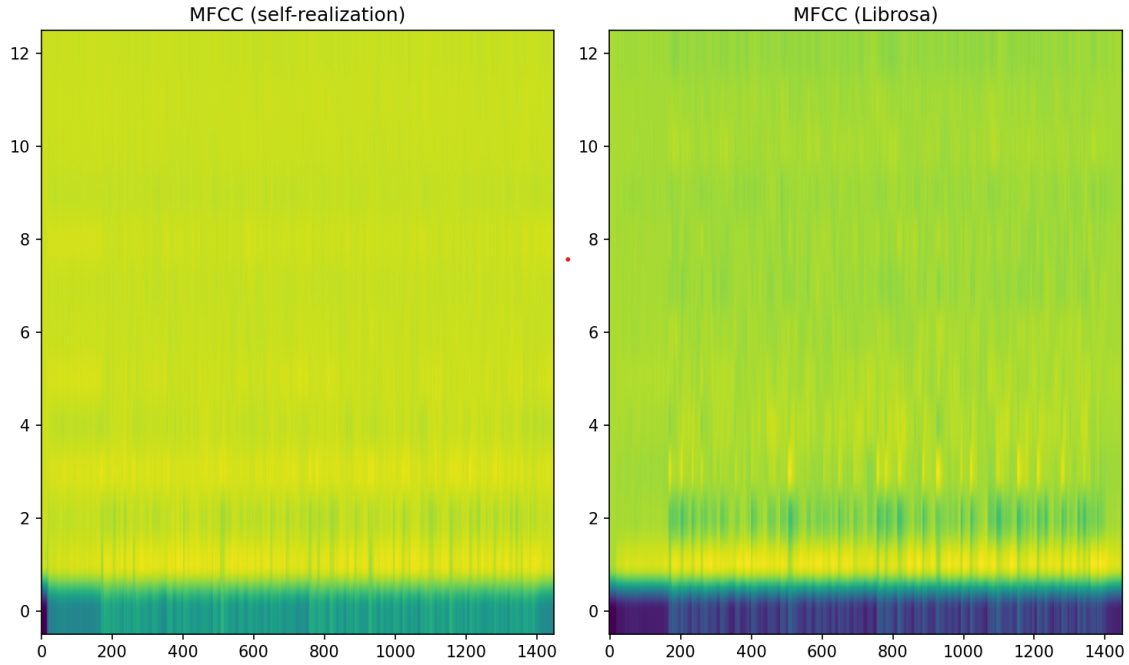
- Dynamic Feature Extraction
  The dynamic features of the MFCC are extracted by calculating the first-order difference over time to capture temporal changes in speech.

```
def delta(mfcc, N=2):
  num_frames = len(mfcc)
  denominator = 2 * sum([i**2 for i in range(1, N+1)])  # Normalization
factor
  delta_feat = np.empty_like(mfcc)

  # Boundary copy
  padded = np.pad(mfcc, ((N, N), (0, 0)), mode='edge')

  # Calculate the difference for each frame
  for t in range(num_frames):
      delta_feat[t] = np.dot(np.arange(-N, N+1), padded[t: t+2*N+1]) /
denominator
  return delta_feat
```

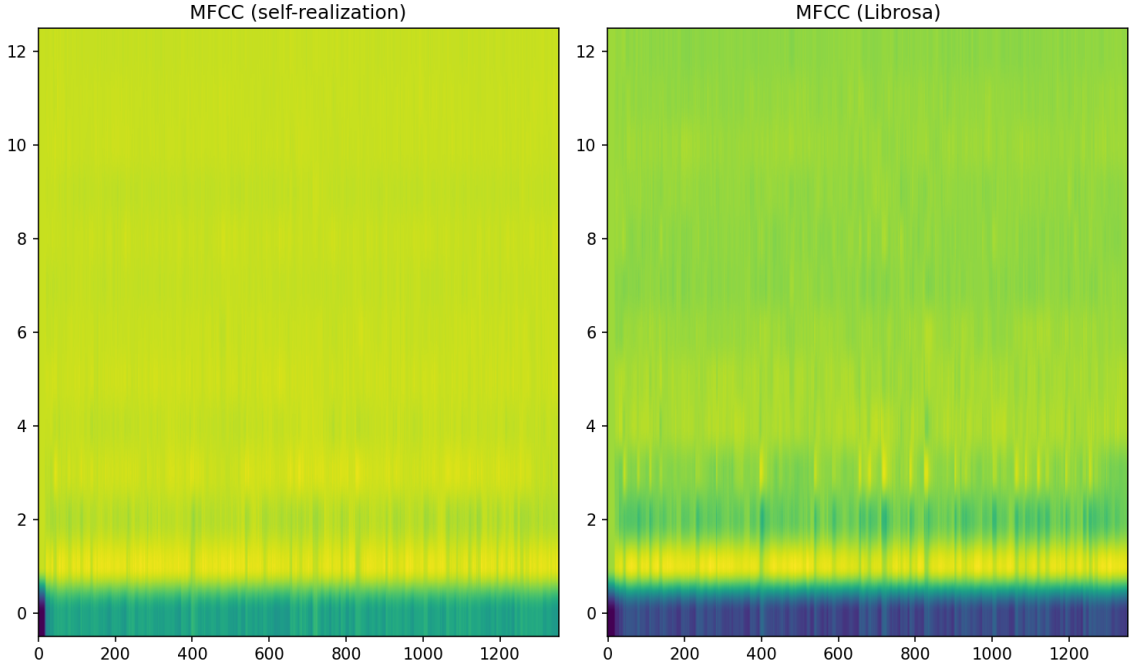
IV.Results and Comparison

1. **Experimental Results**

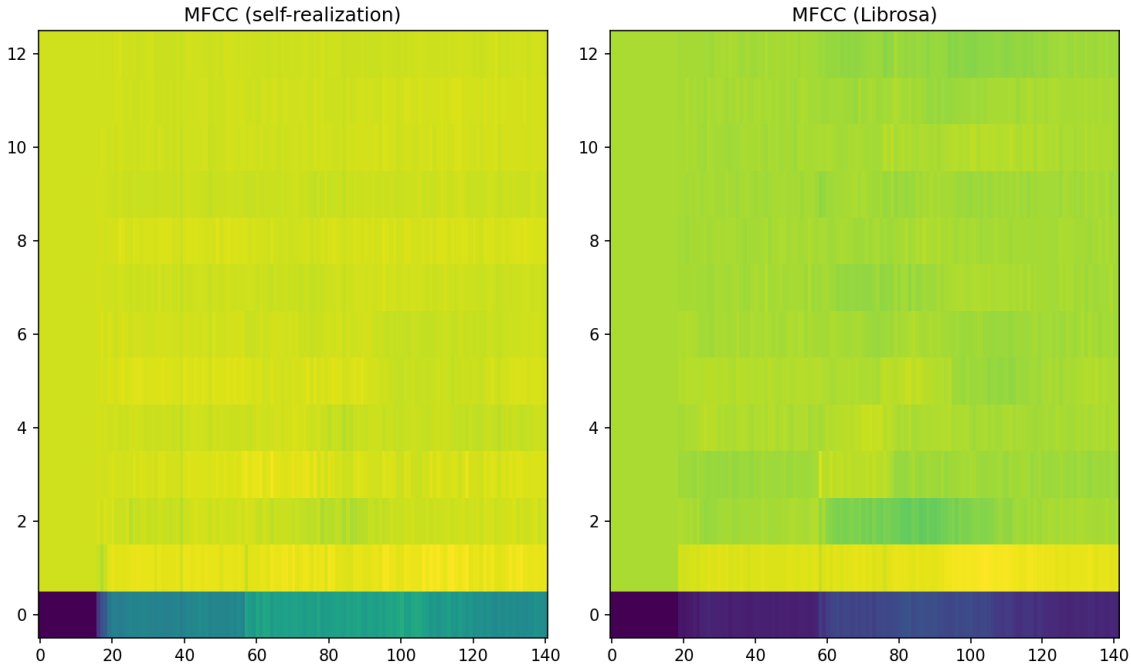
- Clear recording



- Noisy recording



- Short recording



2. **Comparison**

- Shape and Value Distribution
  The experimental results show that the MFCC features from the custom implementation are very close to the MFCC features generated by the Librosa library in terms of shape and general value distribution. This indicates that the custom implementation correctly executed each step of the MFCC process, especially in the preprocessing, Short-Time Fourier Transform (STFT), and Mel filter bank stages, producing results similar to the standard library.
- Color Gradient
  In the custom MFCC, the energy is higher in the low-frequency region (0-2 area), resulting in brighter colors, with no apparent low-energy regions (color gradient tending towards bright yellow). On the other hand, the MFCC from Librosa shows more low-frequency detail, with colors gradually transitioning from deep purple to yellow, highlighting more noticeable frequency variation and energy differences. This suggests that the custom MFCC might lack precision in capturing frequency variation in the low-frequency range.
- Spectral Detail
  In terms of detail, the Librosa version displays more intricate changes in the spectrum (clearer waveforms) and shows more texture and spectral variation (vertical stripes in the image) in the high- and mid-frequency regions. In contrast, the custom implementation's high-frequency spectral features appear smoother, lacking distinct details, with less spectral fluctuation compared to the Librosa version. This indicates that Librosa captures more details in the high-frequency range.
- Data Scale
  The overall value range of the custom MFCC differs from Librosa's default settings.
- Time Resolution
  On the time axis (x-axis), the Librosa result shows more horizontal variation and energy distribution, while the custom implementation appears more consistent and smooth, suggesting a loss of time-related detail.

V .Analysis of Differences

1. **Reasons for Differences**

- Mel Filter Design
  The design of the Mel filter is likely the main cause of the differences. In the custom code, the mapping from Mel frequency to frequency points and the weight distribution of the filters might differ slightly from Librosa's default implementation. Librosa's filter bank may be better suited for capturing fine spectral variations.
- Power Spectrum Calculation
  In this experiment, the power spectrum was calculated using FFT, whereas Librosa might include some additional optimizations or preprocessing (e.g., normalization). Differences in power spectrum calculation methods and numerical processing (such as logarithmic transformation or normalization) could lead to variations in the final MFCC.
- Pre-emphasis and Windowing
  Although the impact of pre-emphasis and windowing on the overall result might be small, differences in parameter selection (such as the alpha value or the type of window function) can affect the short-term characteristics of the signal.
- Discrete Cosine Transform (DCT)
  The DCT used in the custom implementation might differ from Librosa in terms of type and normalization method. Different DCT implementations may result in varying MFCC coefficient outputs.

VI.Conclusion

1. **Experimental Findings**

- Importance of Dynamic Feature Extraction
  Dynamic features, such as first-order and second-order differences, capture the temporal changes in MFCCs, allowing for the detection of dynamic variations in speech signals. Combining these differential features with static MFCC features enhances the descriptive power of speech signals. In this experiment, by comparing static and dynamic features, we observed the temporal variation trends of speech signals, which is crucial for tasks like speech recognition.

2. **Directions for Optimization and Improvement**

- Optimizing Window Function Selection
  In the experiment, a Hamming window was used to reduce the boundary effects of the frames. Different window functions (such as Hanning, rectangular, or Blackman windows) may affect the smoothness of the spectrum. Future research could explore and experiment with different types of window functions to identify the one best suited for specific speech tasks.
- Parameter Adjustment
  - Frame Length and Frame Shift
    The current frame length is set to 25 ms, with a frame shift of 10 ms, which is standard in many speech processing tasks. However, depending on specific applications (e.g., speech with varying speaking speeds), adjusting the frame length and shift could balance time and frequency resolution.
  - Number of Mel Filters
    The number of Mel filters (currently set to 26) directly affects the frequency resolution of the MFCCs. Future experiments could adjust the number of filters to assess their impact on speech recognition tasks.

- Improving Computational Efficiency
  In the current implementation, steps like the Short-Time Fourier Transform (STFT) and Discrete Cosine Transform (DCT) are computed frame by frame. Using optimized parallel computation or convolution techniques could enhance the speed and efficiency of feature extraction, especially when processing large-scale speech data.
- Automated Feature Selection
  The current approach extracts a fixed number of MFCC coefficients (e.g., 13), which is based on traditional settings. Data-driven methods, such as Principal Component Analysis (PCA) or other dimensionality reduction techniques, could be used to automatically select the most important feature coefficients, potentially improving model performance.
- Noise Handling and Robustness
  Different noise scenarios can be introduced to test the robustness of the custom MFCC implementation. To improve performance in noisy environments, more complex preprocessing steps like noise suppression or speech enhancement techniques could be incorporated.
- Quantitative Evaluation of Results
  In addition to visual comparisons with MFCC features generated by Librosa, quantitative evaluation metrics (such as mean squared error or correlation coefficients) could be introduced to precisely measure the differences between the custom and Librosa MFCCs. This would help identify finer distinctions and guide further optimization efforts.