🎵 A complete Python code snippet for creating an audio spectrum visualizer in Google Colab by Worachat W., Ph.D. 2025 🎵

This visualizer allows you to upload a WAV audio file and displays a bar-based frequency spectrum that updates in real-time as the audio plays. It uses Matplotlib for visualization and IPython for audio playback.

Audio Spectrum Visualizer in Google Colab

Overview

This code:

1. Uploads an audio file (WAV format) using Google Colab's file upload feature.
2. Processes the audio to compute its frequency spectrum using the Fast Fourier Transform (FFT).
3. Creates an animated bar plot to visualize the spectrum.
4. Provides an audio player to listen to the file while watching the visualization.

Prerequisites

- Run this code in a Google Colab notebook.
- Ensure your audio file is in WAV format (e.g., `sample.wav`).

How to Use

1. **Copy the Code**: Paste this code into a cell in a Google Colab notebook.
2. **Run the Cell**: Execute the cell by pressing `Shift + Enter`.
3. **Upload a File**: A file upload prompt will appear. Upload a WAV audio file from your computer.
4. **View the Output**:

- An animated bar plot will appear, showing the frequency spectrum.
- An audio player will display below the animation. Click "Play" to listen to the audio while watching the visualization.

Explanation

- **File Upload**: Uses `files.upload()` to let you upload a WAV file.
- **Audio Processing**: The `wave` library reads the file, and the data is converted to a NumPy array. Stereo audio is simplified to mono by taking the first channel.
- **STFT**: The audio is split into overlapping windows (1024 samples each, shifted by 512 samples). The FFT is computed for each window to get the frequency spectrum.
- **Visualization**: A bar plot with 64 bars (representing the first 64 frequency bins) updates dynamically using `FuncAnimation`. Each bar's height reflects the magnitude of a frequency bin.
- **Audio Playback**: `IPython.display.Audio` creates a playable audio widget with the processed audio data.

Notes

- **Supported Format**: This code works with WAV files. For other formats (e.g., MP3), you'd need a library like `librosa` (install with `!pip install librosa` and modify the code).
- **Y-Axis Scaling**: The `ax.set_ylim(0, 10000)` is a fixed limit. If the bars are too small or clipped, adjust this value based on your audio's volume.
- **Performance**: For long audio files, animation generation may take time. Use a short clip (e.g., 10-20 seconds) for best results.
- **Synchronization**: The animation and audio aren't perfectly synced but are close enough for a visual effect. Start the audio manually after the animation appears.

```
# Import necessary libraries
!pip install pydub

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from google.colab import files
```

```
import io
import wave
from IPython.display import Audio, display, HTML

# Step 1: Upload the audio file
print("Please upload a WAV audio file:")
uploaded = files.upload()

# Access the uploaded file
for filename in uploaded.keys():
    audio_data = uploaded[filename]
    audio_file = io.BytesIO(audio_data)

# Step 2: Read the audio file
with wave.open(audio_file, 'rb') as wf:
    sample_rate = wf.getframerate()  # e.g., 44100 Hz
    n_channels = wf.getnchannels()   # 1 for mono, 2 for stereo
    sample_width = wf.getsampwidth() # 1 for 8-bit, 2 for 16-bit
    n_frames = wf.getnframes()       # Total number of frames
    audio_data = wf.readframes(n_frames)  # Raw audio bytes

# Step 3: Convert audio data to NumPy array
if sample_width == 2:
    audio_np = np.frombuffer(audio_data, dtype=np.int16)
elif sample_width == 1:
    audio_np = np.frombuffer(audio_data, dtype=np.uint8) - 128  # Convert to signed
else:
    raise ValueError("Unsupported sample width")

# If stereo, use the first channel only
if n_channels > 1:
    audio_np = audio_np.reshape(-1, n_channels)[:, 0]

# Step 4: Define parameters for Short-Time Fourier Transform (STFT)
window_size = 1024  # Number of samples per FFT window
hop_size = 512     # Number of samples between successive windows
n_bins = 64        # Number of frequency bins to display
```

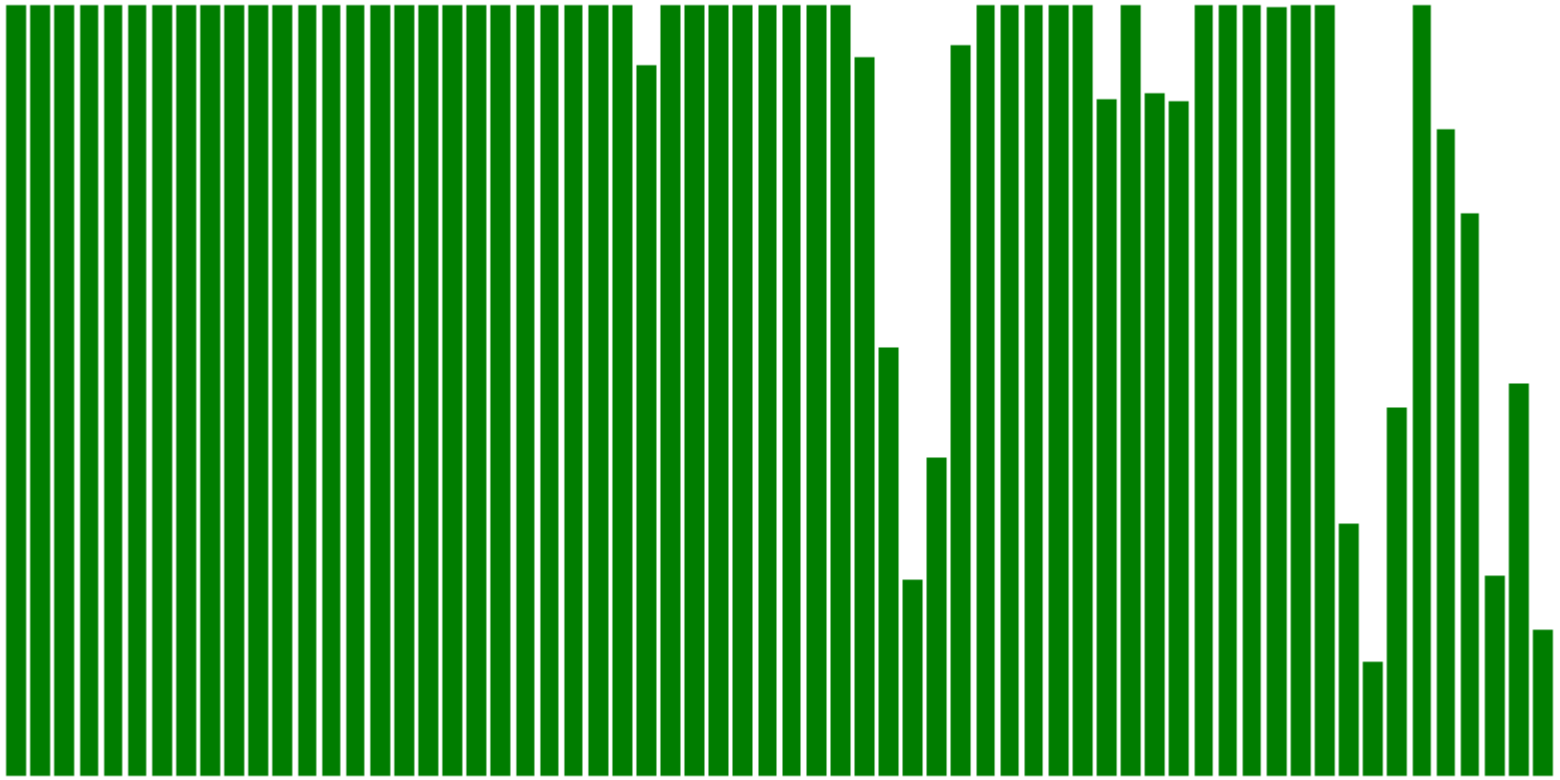
```
n_frames = int(np.floor((len(audio_np) - window_size) / hop_size)) + 1  # Total frames

# Step 5: Set up the visualization plot
fig, ax = plt.subplots(figsize=(10, 5))
ax.set_facecolor('black')          # Black background
ax.set_ylim(0, 10000)              # Y-axis limit (adjust if needed)
ax.set_xlim(-0.5, n_bins - 0.5)    # X-axis limit for 64 bars
ax.axis('off')                     # Hide axes
bars = ax.bar(range(n_bins), np.zeros(n_bins), color='green', width=0.8)  # Create bars

# Step 6: Define the animation update function
def update(frame):
    start = frame * hop_size
    if start + window_size > len(audio_np):
        return bars  # Stop if window exceeds audio length
    # Extract window and apply Hamming window
    windowed_data = audio_np[start:start + window_size] * np.hamming(window_size)
    # Compute FFT and get magnitudes
    fft_data = np.fft.fft(windowed_data, n=window_size)
    magnitudes = np.abs(fft_data[:n_bins])
    # Update bar heights
    for bar, height in zip(bars, magnitudes):
        bar.set_height(height)
    return bars

# Step 7: Create the animation
interval = (hop_size / sample_rate) * 1000  # Frame interval in milliseconds
ani = FuncAnimation(fig, update, frames=n_frames, interval=interval, blit=False)

# Step 8: Display the animation
display(HTML(ani.to_jshtml()))

# Step 9: Display the audio player
display(Audio(audio_np, rate=sample_rate))
```

```
Please upload a WAV audio file:
```

เลือกไฟล์ bbc_sherlock_london.wav

- **bbc_sherlock_london.wav**(audio/wav) - 5718748 bytes, last modified: 25/6/2568 - 100% done

```
Saving bbc_sherlock_london.wav to bbc_sherlock_london.wav
```



Once Loop Reflect

0:29 / 0:29

✅ This code has been modified to include **both a real-time waveform and spectrogram display** using `matplotlib.animation`.

Key updates:

- A subplot layout was added to show **two graphs**: a waveform (top) and a spectrogram (bottom).
- FFT data is visualized as a growing spectrogram (`imshow`) that updates in real-time.
- The waveform scrolls with each audio window.

```
# Import necessary libraries
!pip install pydub
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.animation import FuncAnimation
from google.colab import files
import io
import wave
from IPython.display import Audio, display, HTML

# Step 1: Upload the audio file
print("Please upload a WAV audio file:")
uploaded = files.upload()

# Access the uploaded file
for filename in uploaded.keys():
    audio_data = uploaded[filename]
    audio_file = io.BytesIO(audio_data)

# Step 2: Read the audio file
with wave.open(audio_file, 'rb') as wf:
    sample_rate = wf.getframerate()
    n_channels = wf.getnchannels()
    sample_width = wf.getsampwidth()
    n_frames = wf.getnframes()
    audio_data = wf.readframes(n_frames)
```

```
# Step 3: Convert audio data to NumPy array
if sample_width == 2:
    audio_np = np.frombuffer(audio_data, dtype=np.int16)
elif sample_width == 1:
    audio_np = np.frombuffer(audio_data, dtype=np.uint8) - 128
else:
    raise ValueError("Unsupported sample width")

if n_channels > 1:
    audio_np = audio_np.reshape(-1, n_channels)[:, 0]

# Step 4: Define parameters
window_size = 1024
hop_size = 512
n_bins = 512
n_frames = int(np.floor((len(audio_np) - window_size) / hop_size)) + 1

# Step 5: Set up plots for both waveform and spectrogram
fig, (ax_wave, ax_spec) = plt.subplots(2, 1, figsize=(12, 8))

# Waveform setup
ax_wave.set_title("Real-Time Waveform")
ax_wave.set_xlim(0, window_size)
ax_wave.set_ylim(-2**15, 2**15)
line_wave, = ax_wave.plot(np.zeros(window_size), color='cyan')

# Spectrogram setup
ax_spec.set_title("Real-Time Spectrogram")
ax_spec.set_facecolor('black')
spec_img = ax_spec.imshow(np.zeros((n_bins, 1)), aspect='auto', origin='lower',
                          extent=[0, 1, 0, sample_rate/2], cmap='magma')

# Step 6: Define animation update function
def update(frame):
    start = frame * hop_size
    if start + window_size > len(audio_np):
        return line_wave, spec_img
```

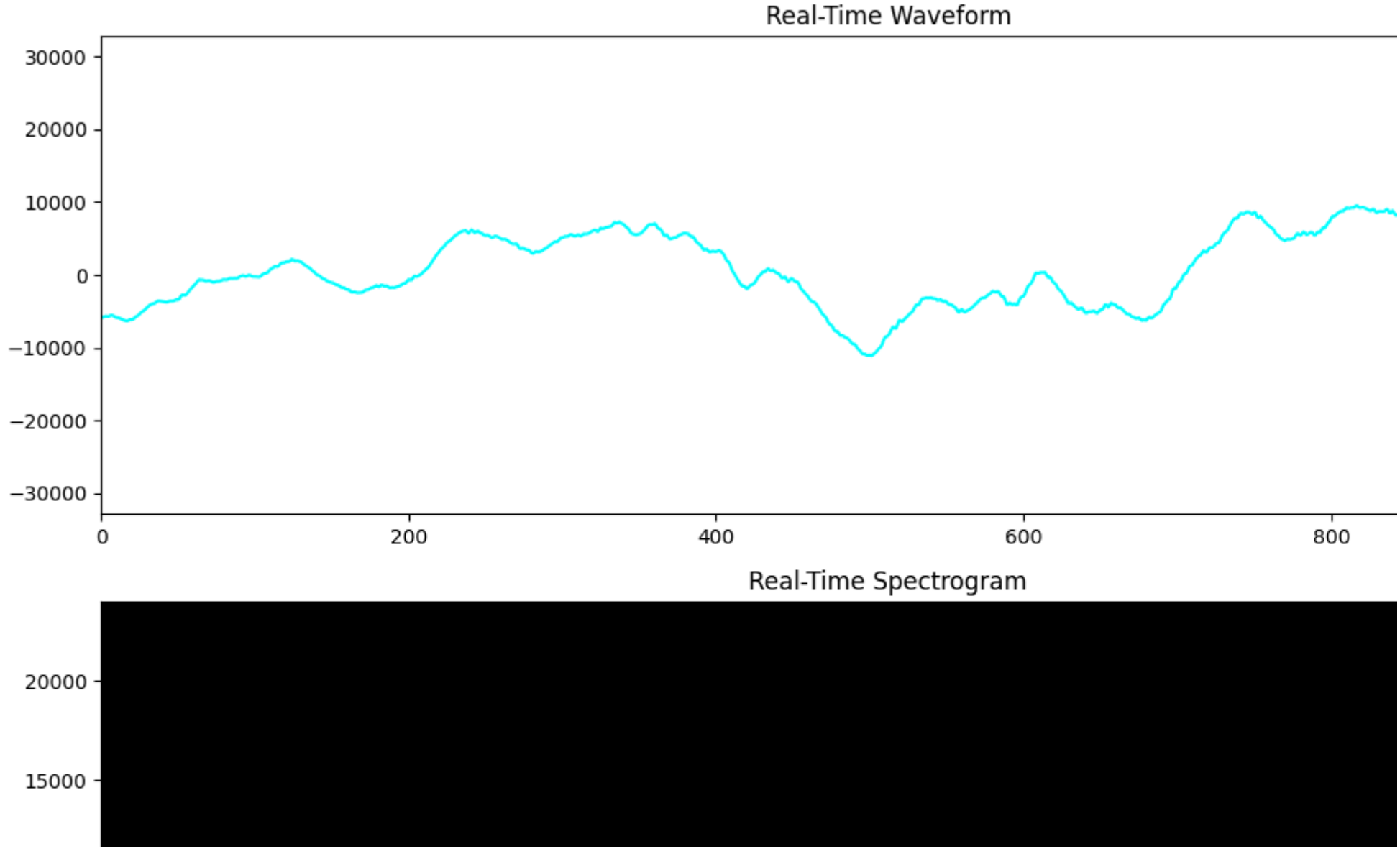
```
    # Extract current audio frame
    frame_data = audio_np[start:start + window_size]

    # Update waveform
    line_wave.set_ydata(frame_data)

    # Apply window and compute FFT
    windowed = frame_data * np.hamming(window_size)
    fft_data = np.abs(np.fft.rfft(windowed))
    fft_data = fft_data[:n_bins].reshape(-1, 1)

    # Update spectrogram
    current_spec = spec_img.get_array()
    updated_spec = np.hstack((current_spec, fft_data))
    spec_img.set_array(updated_spec)
    spec_img.set_extent([0, updated_spec.shape[1], 0, sample_rate/2])

    return line_wave, spec_img

# Step 7: Create the animation
interval = (hop_size / sample_rate) * 1000
ani = FuncAnimation(fig, update, frames=n_frames, interval=interval, blit=False)

# Step 8: Display animation and audio
plt.tight_layout()
display(HTML(ani.to_jshtml()))
display(Audio(audio_np, rate=sample_rate))
```

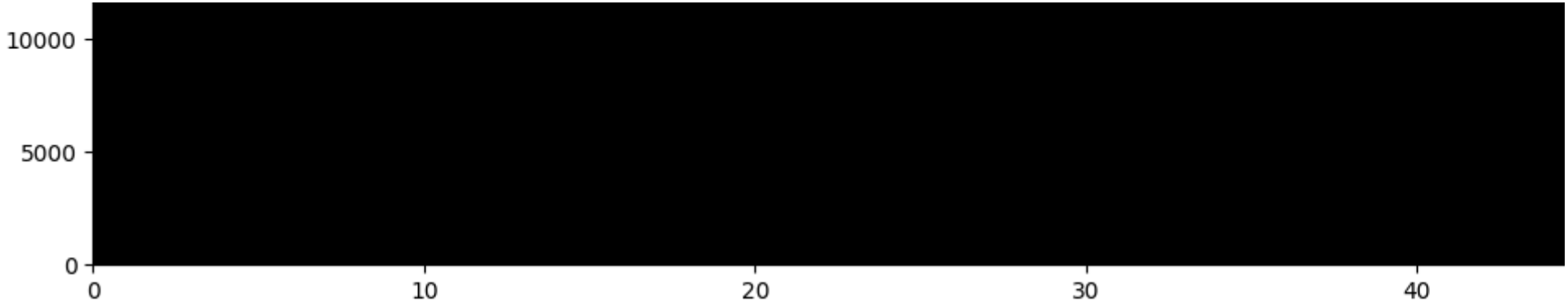
```
Requirement already satisfied: pydub in /usr/local/lib/python3.11/dist-packages (0.25.1)
Please upload a WAV audio file:
```

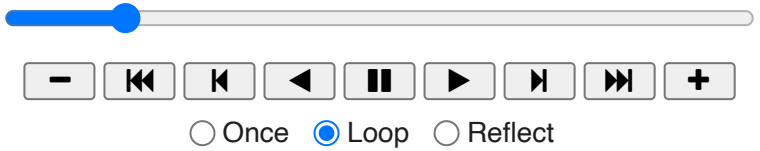
เลือกไฟล์ bbc_sherlock_london.wav

- **bbc_sherlock_london.wav**(audio/wav) - 5718748 bytes, last modified: 25/6/2568 - 100% done

```
Saving bbc_sherlock_london.wav to bbc_sherlock_london (1).wav
WARNING:matplotlib.animation:Animation size has reached 20973209 bytes, exceeding the limit of 20971520.0. If you're
```

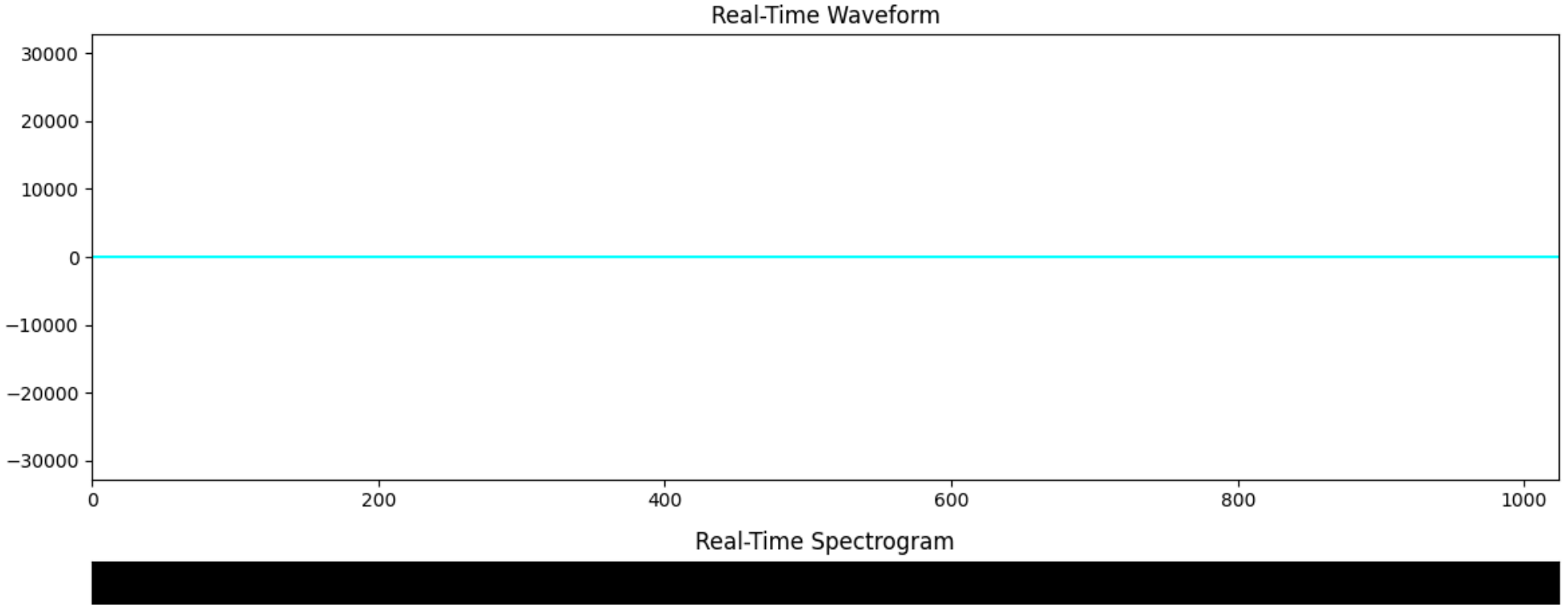



10000
5000
0
0
10
20
30
40



Once
Loop
Reflect

0:11 / 0:29


Real-Time Waveform
30000
20000
10000
0
−10000
−20000
−30000
0
200
400
600
800
1000
Real-Time Spectrogram