Echoes Reborn: A Comprehensive Pipeline for Historic Piano Audio Restoration

Johanna Smriti
Department of CSE
UC, Santa Cruz
ijegan@ucsc.edu

Peter Cai
Department of CSE
UC, Santa Cruz
yocai@ucsc.edu

Anannya P. Neog
Department of CSE
UC, Santa Cruz
aneog@ucsc.edu

Abstract

Historical piano recordings hold immense cultural value but are often marred by degradation, including gramophone noise, audio gaps, and loss of high-frequency details. To address these challenges, this paper presents a novel end-to-end neural pipeline that combines classification, de-noising, and audio inpainting to restore these recordings with high fidelity. Unlike traditional de-noising models, the paper presents an approach that preserves valuable signals and melodies even when the input audio is noise-free, ensuring integrity and consistency in the restored output. Using the MusicNet historic piano dataset, this method achieves state-of-the-art results, including a classification accuracy of 99.92% and a Signal-to-Noise Ratio (SNR) improvement of 21.21 dB in de-noising tasks. These results demonstrate the pipeline's effectiveness and its potential to advance the preservation of culturally significant audio archives.

1 Introduction

Preserving historical audio is vital for safeguarding the cultural and musical heritage encapsulated in early recordings, many of which suffer from significant degradation, such as gramophone noise, high-frequency loss, and audio gaps. As Michael De Sapio [1] observes, restoring classical music allows us to connect with the emotions and ideals of the past, underscoring the importance of reviving these performances for future generations.

Despite advances in audio restoration, current methods often face two critical challenges: preserving the integrity of pristine audio during de-noising and maintaining musical details amidst heavy degradation. Overprocessing can lead to the unintended loss of valuable sonic information, further compromising these recordings' historical and cultural value.

To address these limitations, the paper introduces Echoes Reborn, a unified, end-to-end neural pipeline tailored to restore and enhance historic piano recordings. This approach integrates classification, de-noising, and audio inpainting into a cohesive framework, uniquely capable of identifying and preserving noise-free audio during processing. This ensures the fidelity and integrity of the restored sound, overcoming a common drawback of existing methods. By prioritizing both the technical quality and artistic nuances of the music, Echoes Reborn advances the field of audio restoration, providing a robust tool for cultural preservation.

2 Related Work

Audio restoration has a rich history, with early foundational work by Takayuki Sasaki in the 1980s [2] exploring temporal noise localization and auditory rearrangement techniques. These experiments emphasized the importance of context-sensitive noise removal, setting the stage for modern restoration methodologies.

38th Conference on Neural Information Processing Systems (NeurIPS 2024).

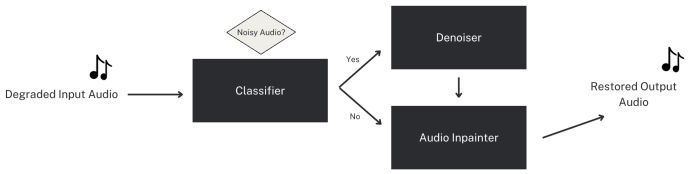


Figure 1: End-to-End Architecture

The advent of deep learning has revolutionized the field, enabling more sophisticated approaches to de-noising and audio enhancement. Notably, E. Moliner [3] introduced a diffusion-based generative equalizer (BABE-2), leveraging priors from diffusion models to enhance historical music recordings. This approach demonstrated the potential of generative methods for restoring degraded audio. Building on this, Moliner and V. Välimäki [4] developed a two-stage U-Net model specifically tailored for high-fidelity de-noising, achieving significant improvements in the restoration of historical recordings.

While these advancements have pushed the boundaries of what is possible in audio restoration, several challenges remain. Many existing methods struggle to preserve the fidelity of pristine audio, often overprocessing noise-free segments and inadvertently degrading their quality. Furthermore, gaps in audio restoration, such as the seamless reconstruction of missing segments, remain underexplored in current literature.

3 Methodology

The proposed method integrates three components: a classifier, de-noiser, and an audio inpainter into a unified pipeline to effectively restore degraded historic piano audio, as shown in Figure 1.

3.1 Classifier Design

The classifier determines whether an audio sample is noisy or clean, enabling conditional de-noising when necessary. Unlike existing methods that may apply de-noising indiscriminately, this approach avoids unnecessary processing of clean audio, preserving its fidelity.

The architecture is based on a convolutional neural network (CNN) optimized for spectrogram inputs (Figure 2). This design balances efficiency and accuracy, with:

- **Three convolutional layers:** Extract time-frequency features from spectrograms, progressively encoding noise patterns.
- **Dropout regularization:** Mitigates overfitting by randomly deactivating 30% of units during training.
- **Fully connected layers:** Map extracted features to a binary output (clean or noisy), leveraging softmax activation for classification probabilities.

3.1.1 Novelty and Design Rationale

The classifier addresses a common limitation in traditional restoration pipelines: overprocessing clean audio due to misclassification. By designing the CNN with a lightweight architecture and leveraging dropout, the model achieves robust classification while maintaining computational efficiency.

Key innovations include:

- **Spectrogram Representation:** Encoding time-frequency information enables the model to capture noise artifacts more effectively than raw waveforms.
- **Conditional De-noising:** Ensures pristine audio is preserved, avoiding the quality degradation common in indiscriminate de-noising methods.

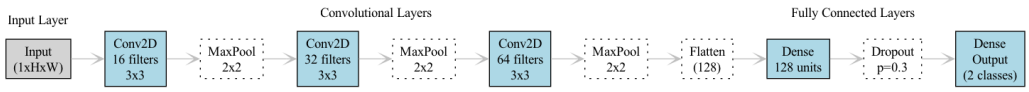


Figure 2: Classifier Architecture

3.1.2 Performance Evaluation

The classifier achieves a test accuracy of 99.92% on the MusicNet historic piano dataset, demonstrating its ability to reliably distinguish between clean and noisy audio samples. This high precision ensures de-noising operations are selectively applied, preserving the integrity of the restored audio. The model's performance outperforms benchmarks in audio classification, as shown by metrics such as accuracy and average test loss.

3.2 De-noising Model Design

The de-noising model restores clean audio signals from noisy spectrogram inputs, leveraging a Convolutional Autoencoder (CAE) architecture. It employs feature extraction in the encoder, followed by reconstruction in the decoder. The model is optimized to minimize loss metrics while enhancing perceptual quality.

3.2.1 Architecture

The autoencoder consists of an encoder-decoder structure, as shown in Figure 3.

Encoder: The encoder reduces the dimensionality of input features while preserving meaningful representations:

$$\mathbf{h}_l = \sigma(\mathbf{W}_l * \mathbf{x}_l + \mathbf{b}_l), \quad l \in \{1, 2\}$$

where:

- $\mathbf{x}_l$ is the input at layer l.
- $\mathbf{W}_l$ and $\mathbf{b}_l$ are weights and biases for layer l.
- $*$ denotes convolution.
- σ is the ReLU activation function, $\sigma(x) = \max(0, x)$.

Max-pooling is applied after the convolutional layers:

$$\mathbf{h}'_l = \text{MaxPool}(\mathbf{h}_l, k = 2, s = 2)$$

where k is the kernel size and s is the stride.

Decoder: The decoder reconstructs the original spectrogram from the compressed latent representation:

$$\mathbf{y}_l = \sigma(\mathbf{W}'_l * \mathbf{h}_l + \mathbf{b}'_l), \quad l \in \{3, 4\}$$

Upsampling layers increase spatial dimensions:

$$\mathbf{y}'_l = \text{UpSample}(\mathbf{y}_l, k = 2)$$

Cropping is applied to align the dimensions of reconstructed spectrograms:

$$\mathbf{y}_{\text{crop}} = \text{Crop}(\mathbf{y}'_l, \text{padding})$$

The final output is:

$$\hat{\mathbf{x}} = \sigma(\mathbf{W}_5 * \mathbf{y}_{\text{crop}} + \mathbf{b}_5)$$

Model Parameters: The autoencoder has 28,353 parameters:

- **Input Shape:** $(1025, 216, 1)$
- **Latent Features:** Encoded into $(513, 108, 32)$
- **Output Shape:** $(1025, 216, 1)$

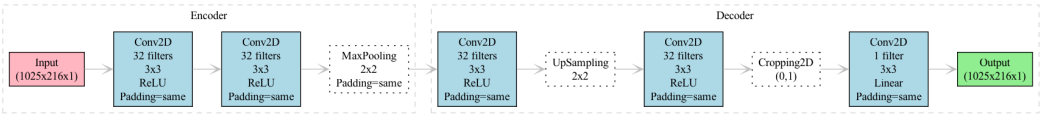


Figure 3: Audio De-noiser Architecture

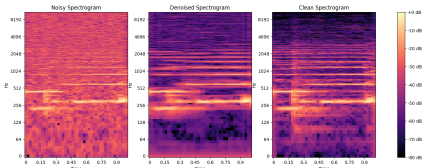


Figure 4: Visualization and Results

3.2.2 Performance Evaluation

The model's performance was evaluated on the test set using multiple metrics:

- **Validation Loss:** Improved to 0.0491 after 50 epochs.
- **MAE:** Achieved a mean absolute error of 0.0834.
- **SNR:** Recorded an average signal-to-noise ratio of 21.21 dB.
- **STFT Loss:** Calculated based on spectrogram differences:

$$\mathcal{L}_{\text{STFT}} = \frac{1}{N} \sum_{i=1}^{N} \|\mathcal{S}(\mathbf{x}_i) - \mathcal{S}(\hat{\mathbf{x}}_i)\|_1$$

  where $\mathcal{S}(\cdot)$ represents the Short-Time Fourier Transform (STFT).

Figure 4 shows visual comparisons of noisy, clean, and de-noised spectrograms, demonstrating the model's effectiveness in restoring historical audio. These results confirm the model's capability to restore both the integrity and perceptual quality of historical piano recordings.

3.3 Audio Inpainting

Audio inpainting aims to reconstruct missing segments in an audio signal, preserving continuity and quality. This section details an approach using Linear Predictive Coding (LPC) to inpaint silent gaps. The mathematical method utilized in this work aligns with the approach described by Andrés Marafioti [5], where deep neural networks (DNNs) were employed for audio inpainting tasks using time-frequency (TF) coefficients to address gaps in music signals.

3.3.1 Linear Predictive Coding (LPC)

Linear Predictive Coding (LPC) predicts future samples of a signal based on its past values. It is a widely-used method for modeling audio signals, effectively capturing their spectral envelope through prediction coefficients.

Key Components:

- **Prediction Coefficients:** Represent the signal properties and are computed to minimize the prediction error.
- **Error Signal:** The difference between the predicted and actual signal, minimized during LPC analysis.

The signal at sample n, $x[n]$, is expressed as:

$$x[n] = -\sum_{k=1}^{p} a_k x[n-k] + e[n]$$

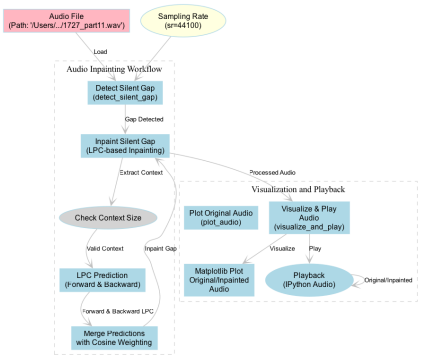


Figure 5: Audio Inpainting Model Workflow

where:

- a_k: Prediction coefficients.
- $e[n]$: Prediction error.
- p: Order of the LPC model.

3.3.2 Audio Inpainting Using LPC

LPC-based audio inpainting proceeds as follows:

1. Silent Gap Detection: Identify silent gaps using a sliding window to calculate the mean amplitude:

$$\mu = \frac{1}{N} \sum_{i=1}^{N} |x[i]|$$

Regions with $\mu <$ threshold are marked as gaps.

2. Context Extraction: Extract audio segments before and after the gap. Compute LPC coefficients using Burg's algorithm to predict forward and backward signals:

$$x_{\text{forw}}[n] = -\sum_{k=1}^{p} a_k x[n-k], \quad x_{\text{backw}}[n] = -\sum_{k=1}^{p} b_k x[n-k]$$

3. Signal Reconstruction: Blend the forward and backward predictions using a cosine-weighted function:

$$x_{\text{blend}}[n] = \cos^2\left(\frac{\pi n}{2L}\right) x_{\text{forw}}[n] + \cos^2\left(\frac{\pi(L-n)}{2L}\right) x_{\text{backw}}[n]$$

where L is the gap length. Combine the reconstructed gap with the original audio to produce the final output.

3.3.3 Results:

A key highlight is the restoration of audio gaps, demonstrated in Figure 6. This figure illustrates the effectiveness of the audio inpainting model, reconstructing the missing segment while maintaining the harmonic and temporal consistency of the original recording.

4 Limitations

The proposed pipeline is specifically optimized for historic piano recordings, which may limit its applicability to other instruments or genres without significant retraining or fine-tuning. While the audio inpainting module effectively reconstructs missing segments, it struggles with larger gaps,

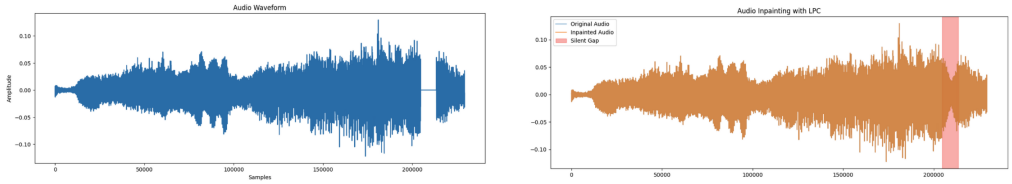


Figure 6: Result Comparison: Audio with Gap (Left) vs. Restored Audio (Right)

occasionally introducing perceptual artifacts. Leveraging GANs trained on larger and more diverse datasets could address these limitations, significantly enhancing the model's ability to handle broader contexts and improve restoration quality.

5 Future Work

While *Echoes Reborn* has demonstrated significant advancements, several areas for improvement remain. Inspired by recent research, the following directions are proposed:

- **Advanced Architectures:** Incorporate diffusion-based models for audio restoration, as shown by Jean-Marie Lemercier [6], which balance interpretability and high performance for complex noise patterns.
- **Dataset Augmentation:** Expand datasets to include diverse instruments and noise types, following the approach of Ethan Manilow [7], who introduced MUSDB18-HQ for audio separation tasks.
- **Real-Time Processing:** Develop lightweight, low-latency models for real-time audio restoration, inspired by Alexandre Défossez [8], who optimized Demucs for dynamic audio processing.
- **Perceptual Metrics:** Integrate perceptual quality metrics like ViSQOL, as validated by Andrew Hines [9], to align system outputs with human auditory preferences.
- **Audio Inpainting:** Enhance generative methods for reconstructing missing audio, building on Andrés Marafioti [10], who employed GANs to effectively address audio gaps.

By exploring these advancements, the aim is to enhance the robustness, scalability, and applicability of *Echoes Reborn* for real-world scenarios.

6 Conclusion

In this work, Echoes Reborn has been introduced, a robust and comprehensive neural pipeline designed for the restoration of historic piano recordings. By integrating classification, de-noising, and audio inpainting into a unified framework, this approach addresses long-standing challenges in audio restoration, including the preservation of pristine audio quality and the seamless reconstruction of heavily degraded segments. The system's capability to selectively apply restoration processes, guided by its high-precision classification component, ensures minimal overprocessing and preserves the artistic and cultural integrity of the restored audio.

Through extensive evaluation on the MusicNet historic piano dataset, Echoes Reborn demonstrated state-of-the-art performance, achieving a classification accuracy of 99.92% and de-noising SNR improvement of 21.21 dB. These results underscore the efficacy of this pipeline in balancing technical rigor with musical authenticity, making it a valuable tool for audio preservationists and researchers.

Beyond its technical contributions, this work highlights the transformative potential of modern machine learning techniques in cultural heritage preservation. The pipeline not only restores lost musical details but also revives the emotional and historical essence encapsulated in these recordings, enabling future generations to connect with the artistic expressions of the past, resulting in rebirth of the timeless Echoes.

References

[1] Michael De Sapio (2024) Early Music and the Conservation of Culture. In, *The Imaginative Conservative*, Available at: https://theimaginativeconservative.org/2024/08/early-music-conservation-culture-michael-de-sapio.html.

[2] Takayuki, Sasaki. (1980) Sound restoration and temporal localization of noise in speech and music sounds. In, *Tohoku University*, Available at: https://psycnet.apa.org/record/1982-11282-001.

[3] Eloi Moliner, Maija Turunen, Filip Elvander & Vesa Välimäki. (2024) A Diffusion-Based Generative Equalizer for Music Restoration. In, *Proceedings of the 27th International Conference on Digital Audio Effects (DAFx24), Guildford, United Kingdom*, Available at: https://arxiv.org/pdf/2403.18636.

[4] Eloi Moliner & Vesa Välimäki. (2022) A Two-Stage U-Net for High-Fidelity De-noising of Historical Recordings. In, *ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Available at: https://ieeexplore.ieee.org/document/9746977.

[5] Andrés Marafioti, Nicki Holighaus, Piotr Majdak & Nathanaël Perraudin. (2022) Audio inpainting of music by means of neural networks. In *146th AES Convention*. Available at: https://arxiv.org/abs/1810.12138.

[6] Jean-Marie Lemercier, Julius Richter & Simon Welker. (2023) Diffusion Models for Audio Restoration. In, *Acoustics Lab, Department of Information and Communications Engineering, Aalto University, Espoo, Finland*, Available at: https://arxiv.org/pdf/2402.09821.

[7] Ethan Manilow, Gordon Wichern, Prem Seetharaman & Jonathan Le Roux. (2019) Cutting Music Source Separation Some Slakh: A Dataset to Study the Impact of Training Data Quality and Quantity. In, *IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, Available at: 10.1109/WASPAA.2019.8937170.

[8] Alexandre Défossez, Nicolas Usunier, Léon Bottou, Francis Bach. (2019) Demucs: Deep Extractor for Music Sources with extra unlabeled data remixed. In, Cornell University, Available at: https://arxiv.org/abs/1909.01174.

[9] Andrew Hines, Jan Skoglund, Anil Kokaram & Naomi Harte. (2015) "ViSQOL: An Objective Speech Quality Model." In, IWAENC 2012; International Workshop on Acoustic Signal Enhancement, Available at: https://ieeexplore.ieee.org/abstract/document/6309421.

[10] Andrés Marafioti, Nathanaël Perraudin, Nicki Holighaus & Piotr Majdak. (2019) A Context Encoder For Audio Inpainting. In, IEEE/ACM Transactions on Audio, Speech, and Language Processing , Available at: 10.1109/TASLP.2019.2947232.

Echoes Reborn: A Comprehensive Pipeline for Historic Piano Audio Restoration

Echoes Reborn is an advanced, end-to-end audio restoration pipeline designed to restore and enhance historical recordings. It combines deep learning with LPC (Linear Predictive Coding)-based inpainting techniques to remove noise, fill gaps, and preserve the fidelity of musical details. This pipeline ensures that clean audio remains untouched while seamlessly restoring degraded sections. For a detailed report on the methodology, results, and performance of this restoration pipeline, please refer to the Echoes Reborn Restoration Report.

Project Structure

```
Echoes-Reborn/
├── audioInpainter/
│   ├── audio_inpainting_model_LPC.ipynb      # Notebook for LPC-based
audio inpainting
│   ├── lpc_ml.ipynb                          # Supporting LPC
implementation details
│   ├── lpc_arch.png                          # Diagram of the
Architecture of the Audio Inpainter
├── classifier/
│   ├── class-epoch-8.pth                     # Pre-trained classifier
model weights
│   ├── classifier.ipynb                      # Classifier model usage
notebook
│   └── classifier_training.ipynb             # Notebook for
classifier model training
│   └── spectrogram_classifier.png            # Diagram of the
Architecture of the Classifier
├── denoiser/
│   ├── weights/
│   │   └── autoencoder/                      # Directory for
autoencoder weights
│   ├── cleaned_audio.wav                     # Example denoised
output
│   ├── denoisingAudio5sec2.h5                # Pre-trained
autoencoder weights
│   ├── DenoisingAudioMSEOriginalFinal.ipynb  # Denoising process
notebook
│   ├── denoisingLoader.ipynb                 # Denoising audio loader
│   └── noisy_example.wav                     # Example noisy audio
input
│   └── audio_denoiser.png                    # Diagram of the
Architecture of the Denoiser
├── pipeline.ipynb                            # Full pipeline
```

```
combining classification, denoising, and inpainting
├── pipelineArchitechture.png                # Pipeleine Architecture
- Workflow Diagram
├── Data Preprocessing                       # Directory for
dataPreprocessing done for throughout the project
│   ├── addNoise.py                          # Adds noise to a given
audio folder with .wav files
│   ├── addSilentGap.py                      # Adds a silent gap in
the audio (for inpainting)
│   ├── deleteSmallAudio.py                  # Deltes scmall audios
less than 5secs to ensure consistency in the dataset
│   ├── denoiseTriming.py                    # Trims all audio in the
files to be 5 seconds
└── README.md
```

Pipeline Overview

The pipeline consists of the following components:

1. **Audio Classification**
   - Identifies whether input audio is `Noisy` or `Clean` using a CNN-based classifier.
   - Input: Mel spectrogram of the audio.
   - Output: Classification label (`Noisy` or `Clean`).
2. **Denoising**
   - Applies an autoencoder to remove noise from audio classified as `Noisy`.
   - Preserves the integrity of `Clean` audio by bypassing the denoising step.
3. **Audio Inpainting**
   - Detects and fills silent gaps in the audio using LPC-based techniques.
   - Ensures smooth transitions and restores missing content without artifacts.
4. **Visualization**
   - Displays waveforms and spectrograms of both the original and processed audio for comparison.

Installation

To set up the project, ensure the following dependencies are installed:

```
pip install torch torchaudio tensorflow librosa matplotlib soundfile
opencv-python scipy spectrum
```

Usage

Run the Full Pipeline

Use the pipeline.ipynb notebook to process audio through all pipeline stages: - Input: Path to an audio file. - Output: Restored audio with noise removed and gaps filled.

Steps: 1. Open the pipeline.ipynb file. 2. Configure the paths for: - Classifier weights (classifier/class-epoch-8.pth). - Autoencoder weights (denoiser/weights/autoencoder/denoisingAudio5sec2.h5). - Input audio file. 3. Execute the cells sequentially.

Individual Modules

Run specific modules independently for targeted tasks:

- Audio Classification: Use classifier/classifier.ipynb to classify input audio.
- Denoising: Use denoiser/denoisingLoader.ipynb for noise removal.
- Audio Inpainting: Use audio-inpainter/audio_inpainting_model_LPC.ipynb to fill gaps in degraded audio.

```
import torchaudio
import matplotlib.pyplot as plt
from IPython.display import Audio, display
import torch
from tensorflow.keras.models import load_model
import numpy as np
import librosa
import soundfile as sf
import cv2
import os
import torch.nn as nn
from scipy.signal import lfiltic, lfilter
from spectrum.burg import _arburg2


class SpectrogramClassifier(torch.nn.Module):
    def __init__(self, num_classes=2):
        super(SpectrogramClassifier, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, stride=1,
padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1,
padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)

        self.fc1 = None
        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = self.relu(self.conv3(x))
        x = self.pool(x)

        x = x.view(x.size(0), -1)

        if self.fc1 is None:
            self.fc1 = nn.Linear(x.size(1), 128).to(x.device)

        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x
```

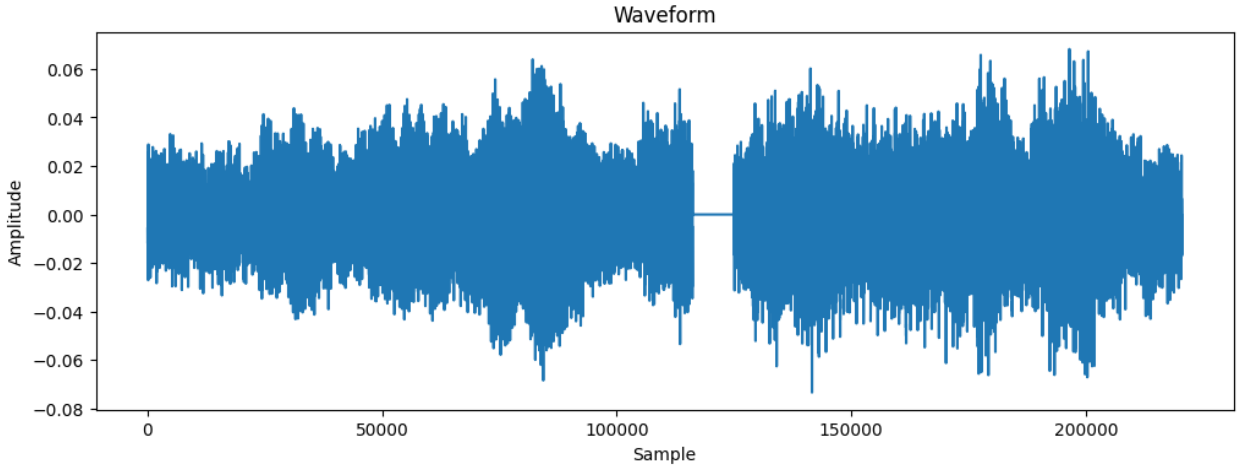
```
def predict_audio(audio_path, model_path, device='cuda'):
    model = SpectrogramClassifier(num_classes=2)
    device = torch.device('cuda' if torch.cuda.is_available() else
'cpu')
    model = SpectrogramClassifier(num_classes=2)
    map_location = device
    model.load_state_dict(torch.load(model_path, map_location=device,
weights_only=True), strict=False)
    model.to(device)
    model.eval()

    sample_rate = 44100
    n_mels = 64
    waveform, sr = torchaudio.load(audio_path)

    if sr != sample_rate:
        resampler = torchaudio.transforms.Resample(orig_freq=sr,
new_freq=sample_rate)
        waveform = resampler(waveform)

    mel_spec = torchaudio.transforms.MelSpectrogram(
        sample_rate=sample_rate,
        n_fft=2048,
        hop_length=512,
        n_mels=64
    )(waveform)

    mel_spec = mel_spec.unsqueeze(0).to(device)

    with torch.no_grad():
        outputs = model(mel_spec)
        _, predicted = torch.max(outputs, 1)

    label = "Noisy" if predicted.item() == 1 else "Clean"
    return label


#Denoiser
class MeanVarianceNorm:
    def fit(self, data_to_fit):
        self.mean = np.mean(data_to_fit)
        self.std = np.std(data_to_fit)

    def normalize(self, data_to_transform):
        return (data_to_transform - self.mean) / self.std

    def denormalize(self, data_to_transform):
        return (data_to_transform * self.std) + self.mean
```

```
def predict_denoise_audio(noisy_audio_filepath, autoencoder,
sample_rate=16000, duration_s=5):
    data, _ = librosa.load(noisy_audio_filepath, sr=sample_rate)
    duration = int(sample_rate * duration_s)

    if len(data) < duration:
        data = np.pad(data, (0, duration - len(data)), 'constant')
    elif len(data) > duration:
        data = data[:duration]

    S_noisy = np.abs(librosa.stft(data, n_fft=2048))[:-1, :]
    S_noisy_resized = cv2.resize(S_noisy, (173, 1024)).T
    N = MeanVarianceNorm()
    N.fit(S_noisy_resized)
    S_noisy_n = N.normalize(S_noisy_resized)

    S_noisy_n = np.expand_dims(S_noisy_n.T, axis=-1)
    S_noisy_n = np.expand_dims(S_noisy_n, axis=0)

    predicted_S_clean_n = autoencoder.predict(S_noisy_n)
    predicted_S_clean = N.denormalize(predicted_S_clean_n[0])

    S_noisy_phase = np.angle(librosa.stft(data, n_fft=2048))[:-1, :]
    S_noisy_phase_resized = cv2.resize(S_noisy_phase, (173,
S_noisy_phase.shape[0])).T
    S_noisy_phase_resized = S_noisy_phase_resized.T

    S_clean_complex = predicted_S_clean.squeeze() * np.exp(1j *
S_noisy_phase_resized)
    data_clean_inv = librosa.istft(S_clean_complex)

    output_path = 'denoised_audio.wav'
    sf.write(output_path, data_clean_inv, sample_rate)
    return output_path


# Audio Inpainting
def inpaint_audio(audio_path, sr, lpc_order, silence_duration_ms=200,
threshold=1e-4):
    def detect_silent_gap(audio, sr, silence_duration_ms=200,
threshold=0.001):
        silence_samples = int((silence_duration_ms / 1000) * sr)
        abs_audio = np.abs(audio)

        for i in range(len(audio) - silence_samples):
            window = abs_audio[i : i + silence_samples]
            if np.mean(window) < threshold:
                gap_start = i
```

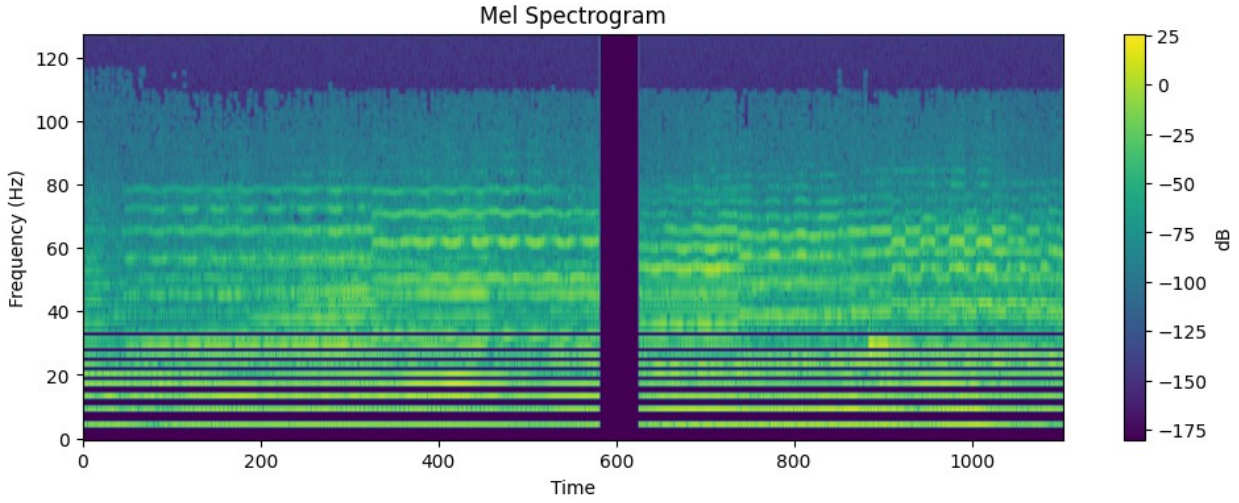
```
                gap_end = i + silence_samples
                return gap_start, gap_end

        print("No sufficiently long silent gap found.")
        return None

    def inpaint_audio_with_lpc(audio, gap_start, gap_end, lpc_order):
        context_size = 3000
        previous_sig = audio[max(0, gap_start - context_size) :
gap_start]
        next_sig = audio[gap_end : min(len(audio), gap_end +
context_size)]

        if len(previous_sig) == 0 or len(next_sig) == 0:
            print("Insufficient context for LPC-based inpainting.")
            return audio

        lpc_signal = LPC(previous_sig, next_sig, gap_start, gap_end,
lpc_order)
        audio[gap_start:gap_end] = np.real(lpc_signal)
        return audio

    def LPC(previous_sig, next_sig, gap_start, gap_end, lpc_order):
        target_length = gap_end - gap_start

        ab, _, _ = _arburg2(previous_sig, lpc_order)
        Zb = lfiltic(b=[1], a=ab, y=previous_sig[:-lpc_order-1:-1])
        forw_pred, _ = lfilter(b=[1], a=ab,
x=np.zeros((target_length)), zi=Zb)

        next_sig = np.flipud(next_sig)
        af, _, _ = _arburg2(next_sig, lpc_order)
        Zf = lfiltic([1], af, next_sig[:-lpc_order-1:-1])
        backw_pred, _ = lfilter([1], af, np.zeros((target_length)),
zi=Zf)
        backw_pred = np.flipud(backw_pred)

        t = np.linspace(0, np.pi/2, target_length)
        sqCos = np.cos(t)**2
        sigout = sqCos * forw_pred + np.flipud(sqCos) * backw_pred
        return sigout

    audio, file_sr = torchaudio.load(audio_path)
    audio = audio.squeeze().numpy()

    if file_sr != sr:

        resampler = torchaudio.transforms.Resample(orig_freq=file_sr,
new_freq=sr)
        audio = resampler(torch.tensor(audio)).numpy()
```

```
    print(f"Audio loaded with shape: {audio.shape}, duration:
{len(audio) / sr:.2f} seconds")

    gap = detect_silent_gap(audio, sr, silence_duration_ms, threshold)
    if gap is None:
        print("No silent gap detected. Returning original audio.")
        return audio_path

    gap_start, gap_end = gap
    print(f"Detected silent gap from {gap_start} to {gap_end}
samples.")

    inpainted_audio = inpaint_audio_with_lpc(audio, gap_start,
gap_end, lpc_order)
    print("Inpainting completed.")
    inpainted_audio_path = audio_path.replace(".wav",
"_inpainted.wav")
    sf.write(inpainted_audio_path, inpainted_audio, sr)
    print(f"Inpainted audio saved to: {inpainted_audio_path}")

    return inpainted_audio_path



def plot_waveform_and_spectrogram(audio_path):
    waveform, sr = torchaudio.load(audio_path)
    if waveform.ndim > 1:
        waveform = waveform.mean(dim=0)

    plt.figure(figsize=(12, 4))
    plt.plot(waveform.numpy())
    plt.title("Waveform")
    plt.xlabel("Sample")
    plt.ylabel("Amplitude")
    plt.show()

    mel_spec = torchaudio.transforms.MelSpectrogram(sample_rate=sr)
(waveform)
    plt.figure(figsize=(12, 4))
    plt.imshow(20 * np.log10(mel_spec.numpy() + 1e-9), aspect='auto',
origin='lower')
    plt.title("Mel Spectrogram")
    plt.xlabel("Time")
    plt.ylabel("Frequency (Hz)")
    plt.colorbar(label="dB")
    plt.show()


# Full Pipeline
```

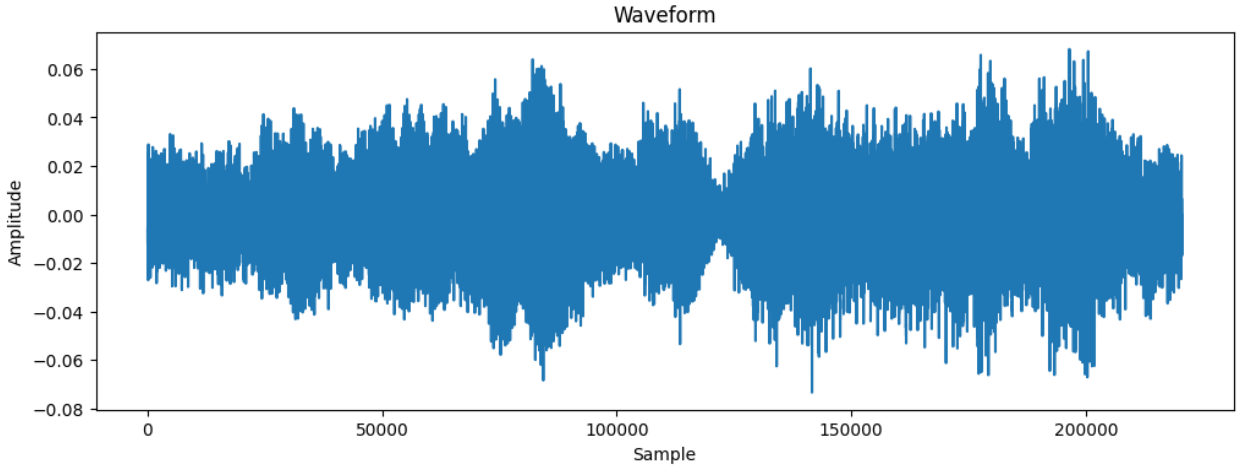
```
def process_audio_pipeline(audio_path, classifier_model_path,
autoencoder, sr=44100, lpc_order=600):
    print("Original Audio:")
    plot_waveform_and_spectrogram(audio_path)
    label = predict_audio(audio_path, classifier_model_path)
    print(f"Classifier Result: {label}")
    audio_path1=audio_path
    if label == "Noisy":
        audio_path1 = predict_denoise_audio(audio_path, autoencoder)
        inpainted_audio_path = inpaint_audio(audio_path1, sr,
lpc_order)
    else:
        inpainted_audio_path = inpaint_audio(audio_path1, sr,
lpc_order)
    print("Final Audio:")
    plot_waveform_and_spectrogram(inpainted_audio_path)
    return audio_path, inpainted_audio_path


if __name__ == "__main__":
    classifier_model_path =
"/Users/johannasmriti/Downloads/musicnet/class-epoch-8.pth"
    autoencoder = load_model("/Users/johannasmriti/Downloads/Final
working model/Docs/denoiser/denoisingAudio5sec2.h5")


WARNING:absl:Compiled the loaded model, but the compiled metrics have
yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.

#clean audio with gap
input_audio_path
="/Users/johannasmriti/Downloads/musicnet/Denoising_Audio/data_with_ma
sk/clean/1727_part4.wav"
original_audio, final_audio = process_audio_pipeline(input_audio_path,
classifier_model_path, autoencoder)

print("Original Audio:")
display(Audio(original_audio, rate=44100))

print("Final Processed Audio:")
display(Audio(final_audio, rate=44100))

Original Audio:
```

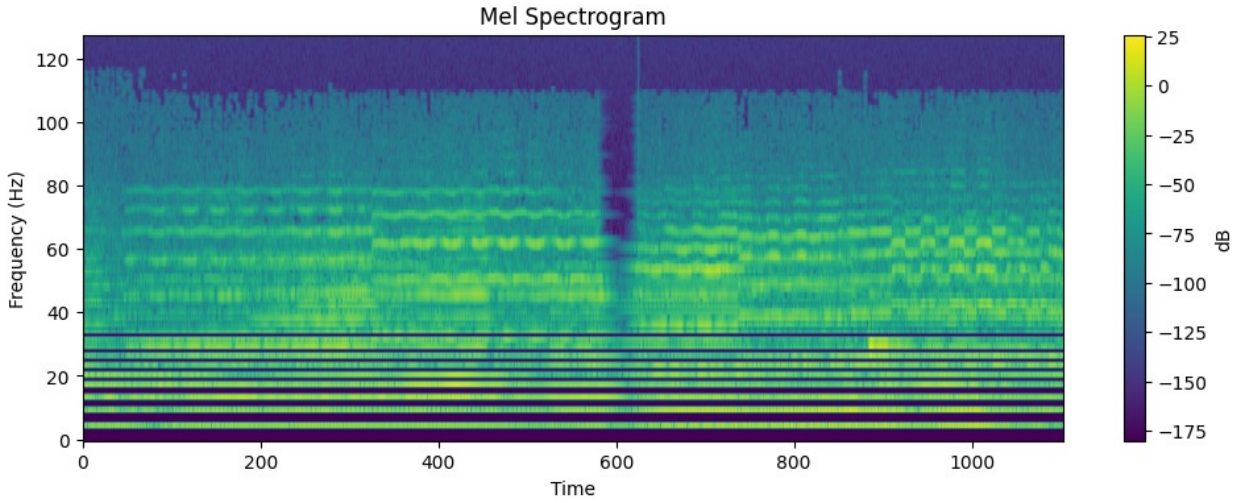



```
/Users/johannasmriti/Library/Python/3.9/lib/python/site-packages/
torchaudio/functional/functional.py:584: UserWarning: At least one mel
filterbank has all zero values. The value for `n_mels` (128) may be
set too high. Or, the value for `n_freqs` (201) may be set too low.
  warnings.warn(
```




```
Classifier Result: Clean
Audio loaded with shape: (220500,), duration: 5.00 seconds
Detected silent gap from 116133 to 124953 samples.
Inpainting completed.
Inpainted audio saved to:
/Users/johannasmriti/Downloads/musicnet/Denoising_Audio/data_with_mask
/clean/1727_part4_inpainted.wav
Final Audio:
```

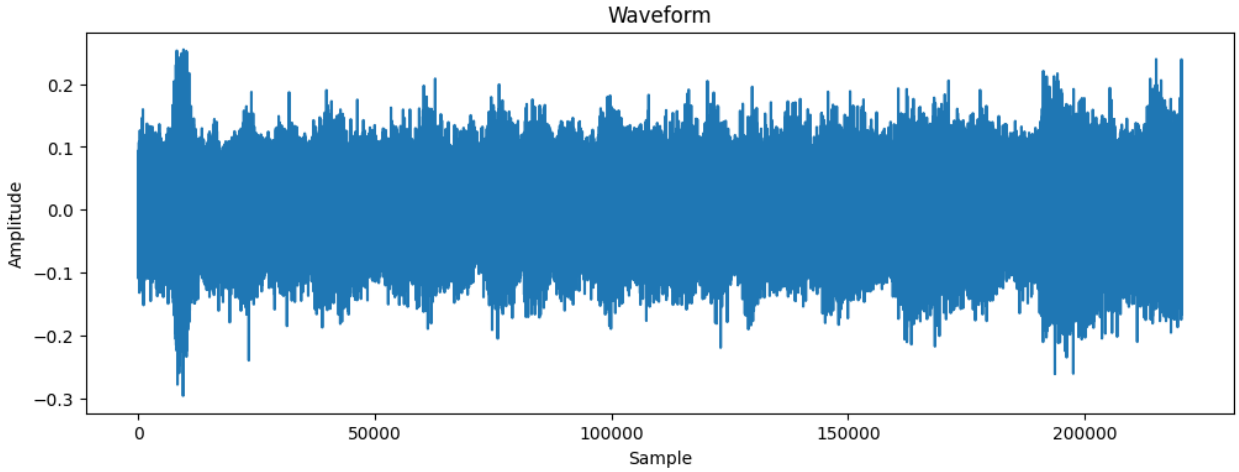


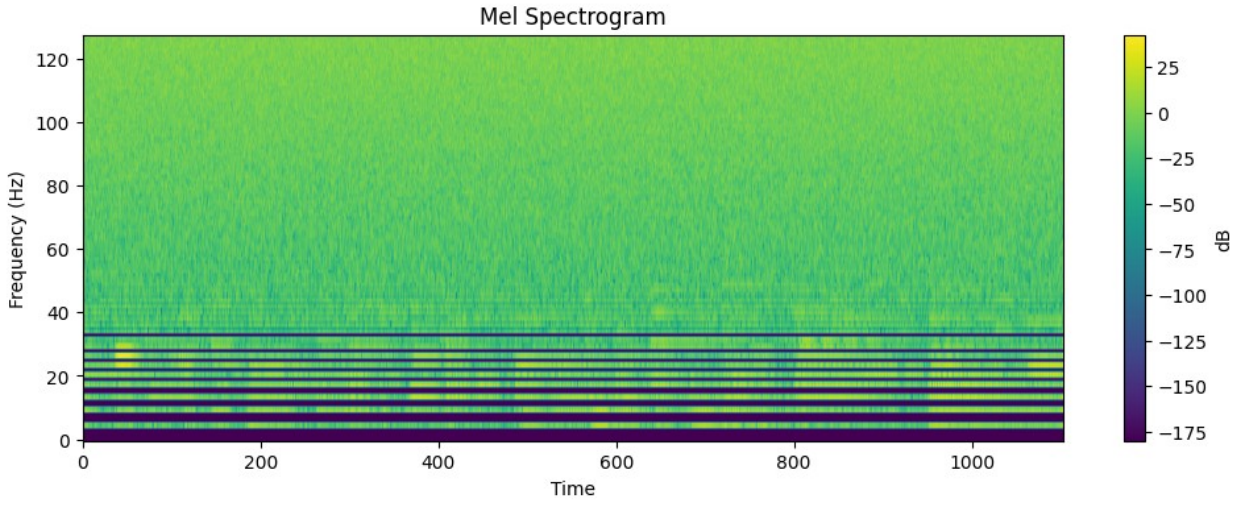



```
Original Audio:

<IPython.lib.display.Audio object>

Final Processed Audio:

<IPython.lib.display.Audio object>
```

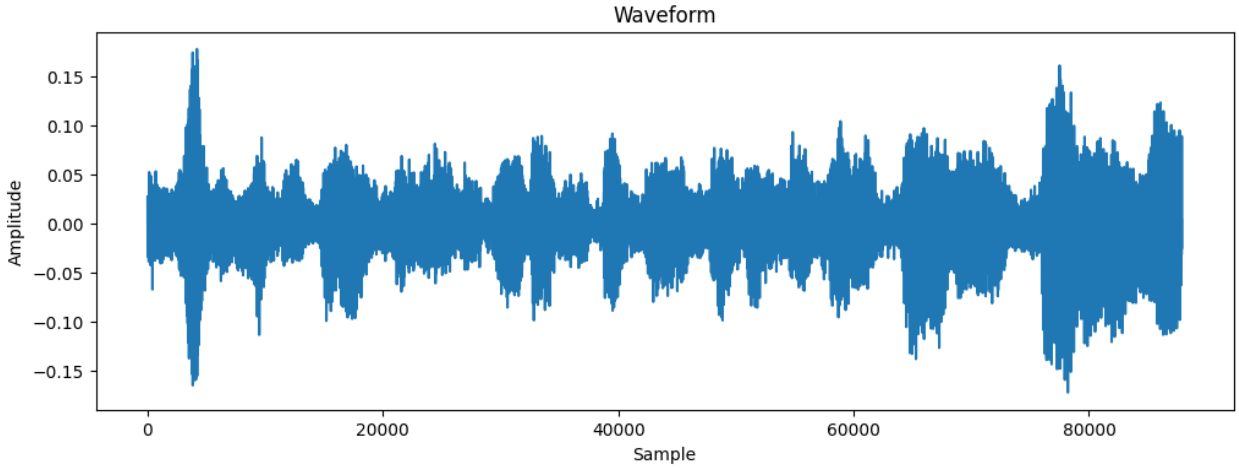
```
#Noisy Audio with no Gap
input_audio_path ="/Users/johannasmriti/Downloads/Final working
model/Docs/denoiser/noisy_example.wav"
original_audio, final_audio = process_audio_pipeline(input_audio_path,
classifier_model_path, autoencoder)

print("Original Audio:")
display(Audio(original_audio, rate=44100))

print("Final Processed Audio:")
display(Audio(final_audio, rate=44100))
```

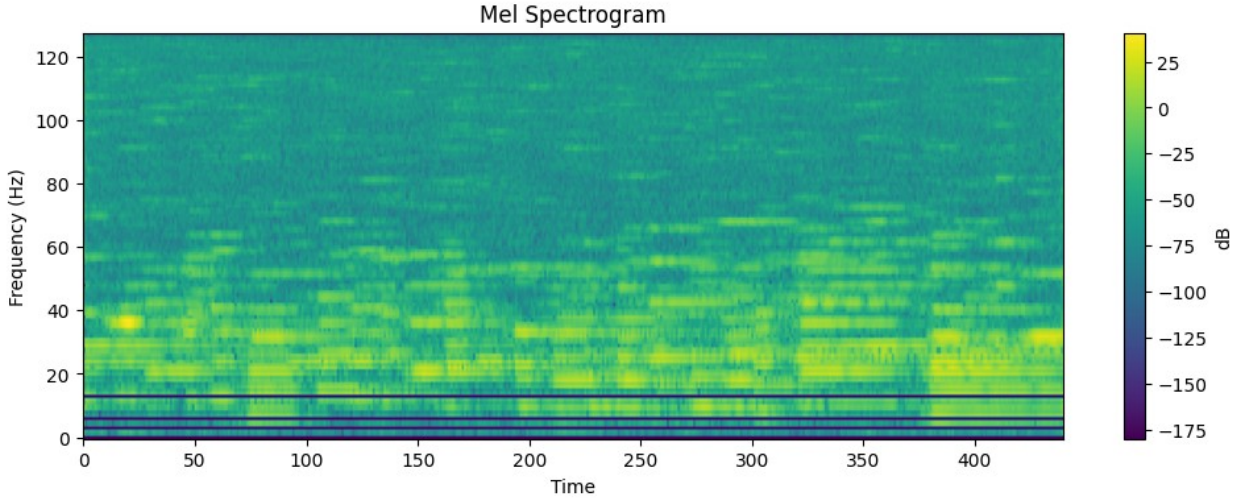
```
Original Audio:
```






```
Classifier Result: Noisy
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 181ms/step
Audio loaded with shape: (87892,), duration: 5.49 seconds
No sufficiently long silent gap found.
No silent gap detected. Returning original audio.
Final Audio:
```

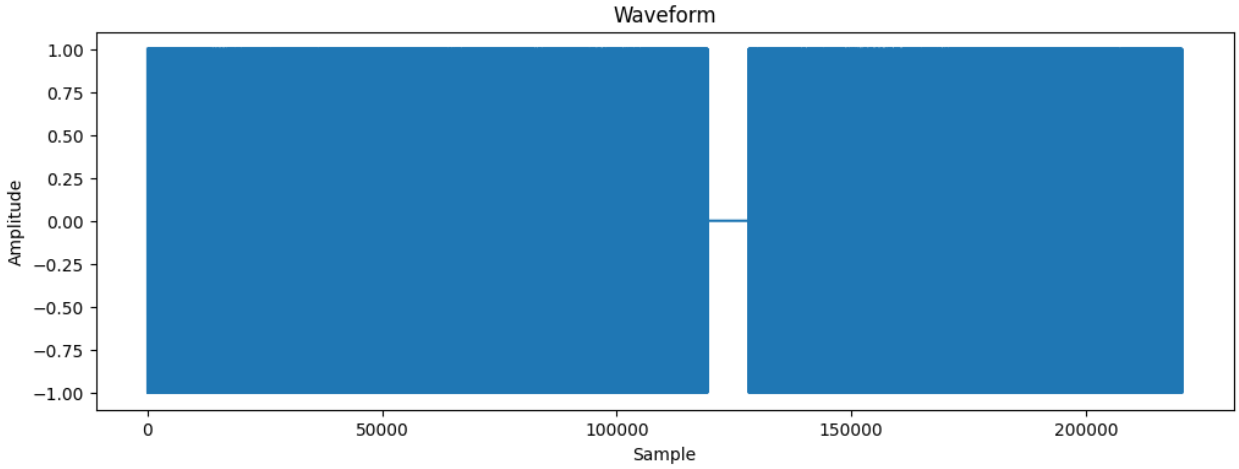


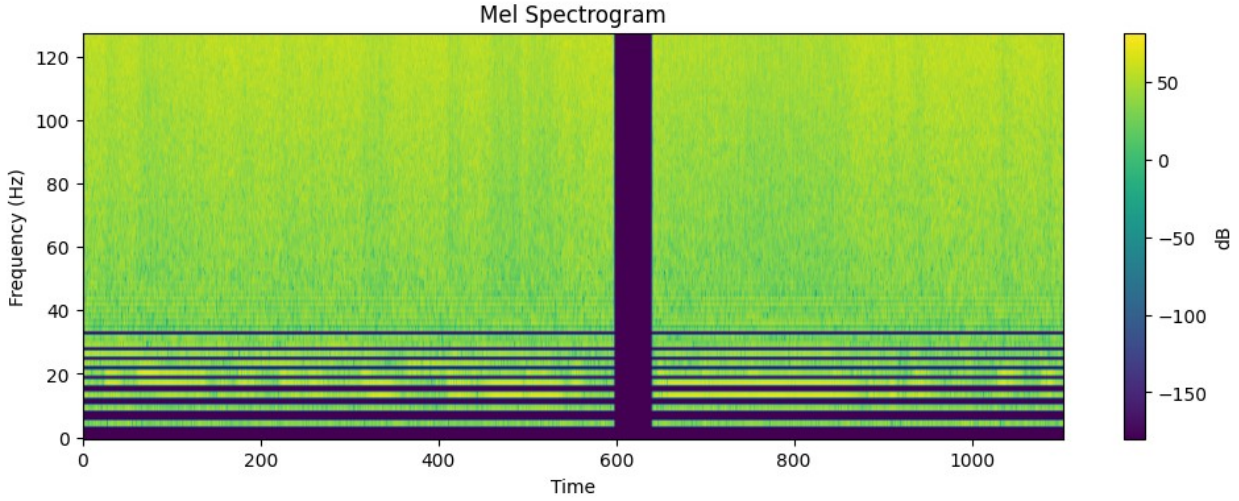



```
Original Audio:

<IPython.lib.display.Audio object>

Final Processed Audio:

<IPython.lib.display.Audio object>
```

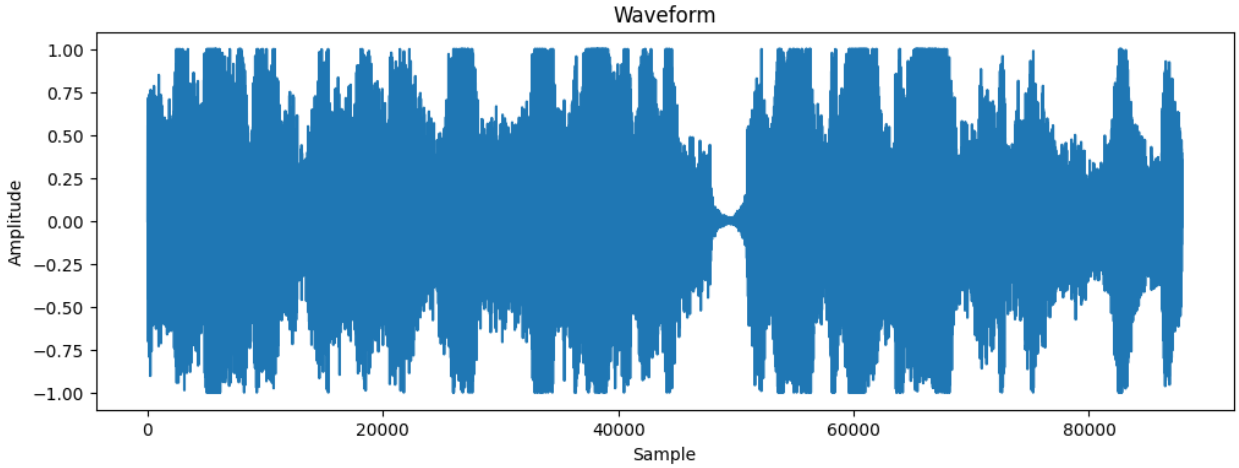
```
#With noise and Gap
input_audio_path
="/Users/johannasmriti/Downloads/samples/noisy_gap/2080_part7.wav"
original_audio, final_audio = process_audio_pipeline(input_audio_path,
classifier_model_path, autoencoder)

print("Original Audio:")
display(Audio(original_audio, rate=44100))

print("Final Processed Audio:")
display(Audio(final_audio, rate=44100))
```

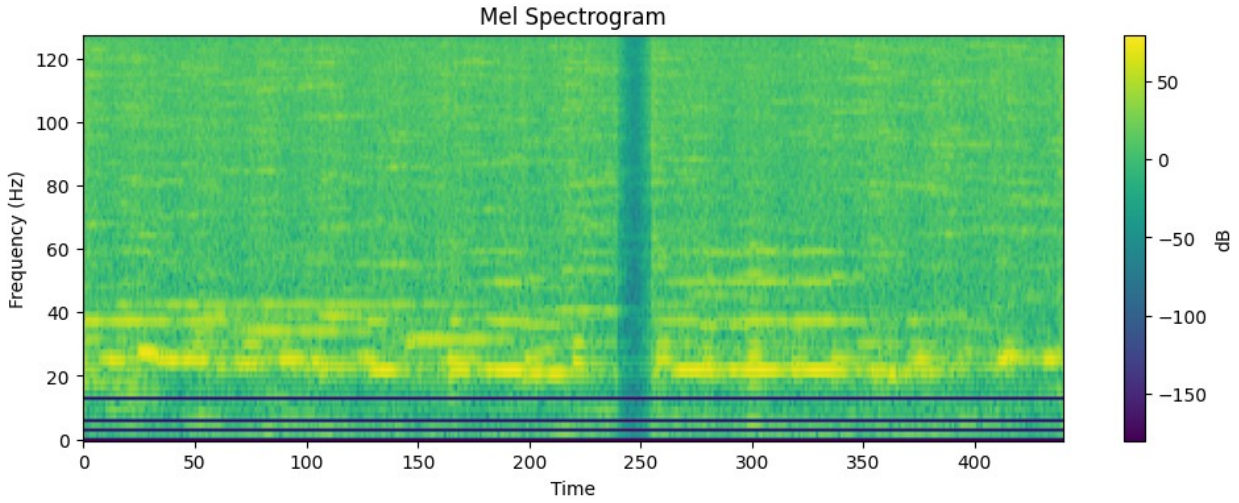
```
Original Audio:
```






```
Classifier Result: Noisy
Audio loaded with shape: (220500,), duration: 5.00 seconds
Detected silent gap from 119349 to 128169 samples.
Inpainting completed.
Inpainted audio saved to:
/Users/johannasmriti/Downloads/samples/noisy_gap/2080_part7_inpainted.
wav
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 158ms/step
Final Audio:
```






```
Original Audio:
<IPython.lib.display.Audio object>
Final Processed Audio:
<IPython.lib.display.Audio object>
```

```
import torch
import torchaudio
import torch.nn as nn

class SpectrogramClassifier(nn.Module):
    def __init__(self, num_classes=2):
        super(SpectrogramClassifier, self).__init__()
        self.conv1 = nn.Conv2d(1, 16, kernel_size=3, stride=1,
padding=1)
        self.conv2 = nn.Conv2d(16, 32, kernel_size=3, stride=1,
padding=1)
        self.conv3 = nn.Conv2d(32, 64, kernel_size=3, stride=1,
padding=1)
        self.pool = nn.MaxPool2d(kernel_size=2, stride=2)
        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.3)

        self.fc2 = nn.Linear(128, num_classes)

    def forward(self, x):
        x = self.relu(self.conv1(x))
        x = self.pool(x)
        x = self.relu(self.conv2(x))
        x = self.pool(x)
        x = self.relu(self.conv3(x))
        x = self.pool(x)

        x = x.view(x.size(0), -1)

        in_features = x.size(1)

        self.fc1 = nn.Linear(in_features, 128)

        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.fc2(x)
        return x


def predict_audio(audio_path, model_path, device='cuda'):
    model = SpectrogramClassifier(num_classes=2)
    map_location = torch.device('cuda') if torch.cuda.is_available()
else torch.device('cpu')

    # Load model weights with strict=False to ignore missing/extra
keys
    state_dict = torch.load(model_path, map_location=map_location)
    model.load_state_dict(state_dict, strict=False)

    model.to(device)
```

```
    model.eval()

    sample_rate = 44100
    n_mels = 64

    waveform, sr = torchaudio.load(audio_path)

    if sr != sample_rate:
        resampler = torchaudio.transforms.Resample(orig_freq=sr,
new_freq=sample_rate)
        waveform = resampler(waveform)

    mel_spec = torchaudio.transforms.MelSpectrogram(
        sample_rate=sample_rate,
        n_fft=2048,
        hop_length=512,
        n_mels=n_mels
    )(waveform)

    mel_spec = mel_spec.unsqueeze(0)

    mel_spec = mel_spec.to(device)

    with torch.no_grad():
        outputs = model(mel_spec)
        _, predicted = torch.max(outputs, 1)

    label = "Noisy" if predicted.item() == 1 else "Clean"

    probabilities = torch.nn.functional.softmax(outputs, dim=1)
    confidence = probabilities[0][predicted.item()].item()

    return label, confidence

if __name__ == "__main__":
    model_path = "/Users/johannasmriti/Downloads/Final working
model/classifier/class-epoch-8.pth"
    device = torch.device("cuda" if torch.cuda.is_available() else
"cpu")

    audio_file = "/Users/johannasmriti/Downloads/Final working
model/denoiser/noisy_example.wav"

    label, confidence = predict_audio(audio_file, model_path, device)

    print(f"Audio file: {audio_file}")
    print(f"Prediction: {label}")
    print(f"Confidence: {confidence:.2%}")
```

```
Audio file: /Users/johannasmriti/Downloads/Final working
model/denoiser/noisy_example.wav
Prediction: Noisy
Confidence: 65.89%

/var/folders/yz/1fd13cns6gvbhv48blbqlsjh0000gn/T/
ipykernel_63310/3051018123.py:42: FutureWarning: You are using
`torch.load` with `weights_only=False` (the current default value),
which uses the default pickle module implicitly. It is possible to
construct malicious pickle data which will execute arbitrary code
during unpickling (See
https://github.com/pytorch/pytorch/blob/main/SECURITY.md#untrusted-
models for more details). In a future release, the default value for
`weights_only` will be flipped to `True`. This limits the functions
that could be executed during unpickling. Arbitrary objects will no
longer be allowed to be loaded via this mode unless they are
explicitly allowlisted by the user via
`torch.serialization.add_safe_globals`. We recommend you start setting
`weights_only=True` for any use case where you don't have full control
of the loaded file. Please open an issue on GitHub for any issues
related to this experimental feature.
  state_dict = torch.load(model_path, map_location=map_location)
```

```
from tensorflow.keras.models import load_model
import IPython.display as ipd
import numpy as np
```

```
/Users/johannasmriti/Library/Python/3.9/lib/python/site-packages/
urllib3/__init__.py:35: NotOpenSSLWarning: urllib3 v2 only supports
OpenSSL 1.1.1+, currently the 'ssl' module is compiled with 'LibreSSL
2.8.3'. See: https://github.com/urllib3/urllib3/issues/3020
  warnings.warn(
```

```
autoencoder=load_model('/Users/johannasmriti/Downloads/Final working
model/denoiser/denoisingAudio5sec2.h5')
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have
yet to be built. `model.compile_metrics` will be empty until you train
or evaluate the model.
```

```
class MeanVarianceNorm():
    def fit(self, data_to_fit):
        self.mean = np.mean(data_to_fit)
        self.std = np.std(data_to_fit)
    def normalize(self, data_to_transform):
        return (data_to_transform - self.mean) / self.std
    def denormalize(self, data_to_transform):
        return (data_to_transform * self.std) + self.mean

import numpy as np
import librosa
import soundfile as sf
import cv2

def predict_denoise_audio(noisy_audio_filepath, sample_rate=16000,
duration_s=5):
    data, _ = librosa.load(noisy_audio_filepath, sr=sample_rate)
    duration = int(sample_rate * duration_s)

    if len(data) < duration:
        data = np.pad(data, (0, duration - len(data)), 'constant')
    elif len(data) > duration:
        data = data[:duration]
    S_noisy = np.abs(librosa.stft(data, n_fft=2048))[:-1, :]

    S_noisy_resized = cv2.resize(S_noisy, (173, 1024)).T

    N = MeanVarianceNorm()
    N.fit(S_noisy_resized)
    S_noisy_n = N.normalize(S_noisy_resized)


    S_noisy_n = np.expand_dims(S_noisy_n.T, axis=-1)
```

```
    S_noisy_n = np.expand_dims(S_noisy_n, axis=0)

    predicted_S_clean_n = autoencoder.predict(S_noisy_n)

    predicted_S_clean = N.denormalize(predicted_S_clean_n[0])

    print(f"Predicted Spectrogram Shape: {predicted_S_clean.shape}")
    print(f"Predicted Spectrogram Values: {predicted_S_clean[:5]}")

    S_noisy_phase = np.angle(librosa.stft(data, n_fft=2048))[:-1, :]

    S_noisy_phase_resized = cv2.resize(S_noisy_phase, (173,
S_noisy_phase.shape[0])).T

    S_noisy_phase_resized = S_noisy_phase_resized.T

    S_clean_complex = predicted_S_clean.squeeze() * np.exp(1j *
S_noisy_phase_resized)

    data_clean_inv = librosa.istft(S_clean_complex)

    if np.any(np.isnan(data_clean_inv)):
        print("Warning: Cleaned audio contains NaN values.")
        return None
    if data_clean_inv.size == 0:
        print("Error: Cleaned audio is empty.")
        return None

    sf.write('cleaned_audio.wav', data_clean_inv, sample_rate)


    return 'cleaned_audio.wav'

noisy_audio_filepath = '/Users/johannasmriti/Downloads/Final working
model/denoiser/noisy_example.wav'
predicted_audio = predict_denoise_audio(noisy_audio_filepath)
```

```
WARNING:tensorflow:5 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x319fe1550> triggered tf.function retracing. Tracing is
expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.
```

```
WARNING:tensorflow:5 out of the last 6 calls to <function
TensorFlowTrainer.make_predict_function.<locals>.one_step_on_data_dist
ributed at 0x319fe1550> triggered tf.function retracing. Tracing is
expensive and the excessive number of tracings could be due to (1)
creating @tf.function repeatedly in a loop, (2) passing tensors with
different shapes, (3) passing Python objects instead of tensors. For
(1), please define your @tf.function outside of the loop. For (2),
@tf.function has reduce_retracing=True option that can avoid
unnecessary retracing. For (3), please refer to
https://www.tensorflow.org/guide/function#controlling_retracing and
https://www.tensorflow.org/api_docs/python/tf/function for  more
details.

1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 123ms/step
Predicted Spectrogram Shape: (1024, 173, 1)
Predicted Spectrogram Values: [[[0.48827326]
  [0.5385084 ]
  [0.54499036]
  [0.54445165]
  [0.5754477 ]
  [0.5573375 ]
  [0.6145939 ]
  [0.61949146]
  [0.5542169 ]
  [0.48801988]
  [0.488376  ]
  [0.48771477]
  [0.50992227]
  [0.45970878]
  [0.42681384]
  [0.42956442]
  [0.42088658]
  [0.43957362]
  [0.4336178 ]
  [0.42948133]
  [0.47261778]
  [0.4555321 ]
  [0.45722216]
  [0.434875  ]
  [0.37542406]
  [0.382062  ]
  [0.3660467 ]
  [0.3681812 ]
  [0.3631523 ]
  [0.3671157 ]
  [0.37342206]
  [0.35309026]
  [0.375843  ]
  [0.3820594 ]
  [0.38864356]
```

```
 [0.4029808 ]
 [0.41366315]
 [0.40251154]
 [0.38710445]
 [0.37545177]
 [0.39307365]
 [0.40918124]
 [0.3959294 ]
 [0.39950693]
 [0.3957676 ]
 [0.3824878 ]
 [0.37428555]
 [0.37046862]
 [0.36590603]
 [0.37520465]
 [0.39066496]
 [0.40746897]
 [0.4263418 ]
 [0.4360031 ]
 [0.4338496 ]
 [0.4325502 ]
 [0.40542   ]
 [0.3851381 ]
 [0.38773677]
 [0.3835216 ]
 [0.36046016]
 [0.36233178]
 [0.37317625]
 [0.38837507]
 [0.39856774]
 [0.38790205]
 [0.39722538]
 [0.3808299 ]
 [0.3714764 ]
 [0.36813504]
 [0.371343  ]
 [0.38486293]
 [0.3712006 ]
 [0.35372767]
 [0.34079874]
 [0.33243334]
 [0.33679947]
 [0.3486562 ]
 [0.36427554]
 [0.3741573 ]
 [0.3671772 ]
 [0.36720538]
 [0.3814268 ]
 [0.40480137]
```

```
 [0.4109522 ]
 [0.38707298]
 [0.50378734]
 [0.6380796 ]
 [0.6196355 ]
 [0.5363779 ]
 [0.4867549 ]
 [0.47911906]
 [0.4539673 ]
 [0.4183191 ]
 [0.40970087]
 [0.41925293]
 [0.4007674 ]
 [0.3989647 ]
 [0.41028166]
 [0.406106  ]
 [0.40334213]
 [0.43324953]
 [0.4782595 ]
 [0.47729486]
 [0.54912084]
 [0.61555886]
 [0.59787077]
 [0.55349207]
 [0.4695531 ]
 [0.42699224]
 [0.43076986]
 [0.43283075]
 [0.4207437 ]
 [0.4060265 ]
 [0.3937846 ]
 [0.3794979 ]
 [0.37888047]
 [0.40070122]
 [0.39985186]
 [0.39977044]
 [0.3890054 ]
 [0.37824538]
 [0.37680683]
 [0.3694608 ]
 [0.34659687]
 [0.34171894]
 [0.35855258]
 [0.36242542]
 [0.35757294]
 [0.3704348 ]
 [0.37623656]
 [0.37244332]
 [0.38494715]
```

```
  [0.4060428 ]
  [0.4552788 ]
  [0.50203323]
  [0.60140496]
  [0.70182633]
  [0.6622996 ]
  [0.5650984 ]
  [0.5348653 ]
  [0.5399245 ]
  [0.58647686]
  [0.66893274]
  [0.67735904]
  [0.6673894 ]
  [0.5961973 ]
  [0.5028328 ]
  [0.43692115]
  [0.4276796 ]
  [0.40651625]
  [0.43567201]
  [0.47079772]
  [0.46824333]
  [0.43453735]
  [0.40074587]
  [0.4185806 ]
  [0.413194  ]
  [0.4066435 ]
  [0.3789468 ]
  [0.38718936]
  [0.38962808]
  [0.38831952]
  [0.401799  ]
  [0.39140308]
  [0.37986508]
  [0.37417865]
  [0.3939068 ]
  [0.41632122]
  [0.4029968 ]
  [0.4078046 ]
  [0.38807908]
  [0.37006518]]

 [[0.35107777]
  [0.42272657]
  [0.49873424]
  [0.6905126 ]
  [0.738885  ]
  [1.0143559 ]
  [1.290719  ]
  [1.4034    ]
```

```
 [1.2167292 ]
 [0.77025706]
 [0.47166395]
 [0.40814507]
 [0.51363075]
 [0.48163575]
 [0.4120555 ]
 [0.3179804 ]
 [0.2665011 ]
 [0.27886054]
 [0.27559343]
 [0.2779507 ]
 [0.35050923]
 [0.44060516]
 [0.42361864]
 [0.2710678 ]
 [0.2404738 ]
 [0.19587597]
 [0.14811337]
 [0.14566791]
 [0.12589741]
 [0.13626558]
 [0.14168817]
 [0.1435051 ]
 [0.15059444]
 [0.1627954 ]
 [0.17713591]
 [0.19895473]
 [0.20656347]
 [0.20831558]
 [0.21535367]
 [0.19729751]
 [0.21117619]
 [0.20590034]
 [0.20051393]
 [0.21582243]
 [0.18237084]
 [0.15952632]
 [0.1791499 ]
 [0.16570598]
 [0.16039291]
 [0.16441315]
 [0.1783438 ]
 [0.21257108]
 [0.21635604]
 [0.22801542]
 [0.23846737]
 [0.24357778]
 [0.23787582]
```

```
 [0.2324641 ]
 [0.2009075 ]
 [0.18898794]
 [0.16679999]
 [0.13753992]
 [0.14710337]
 [0.16289008]
 [0.17284113]
 [0.18214679]
 [0.18500489]
 [0.18652603]
 [0.1905142 ]
 [0.16077599]
 [0.15076485]
 [0.13991266]
 [0.1272459 ]
 [0.11357707]
 [0.11321467]
 [0.11729753]
 [0.1280384 ]
 [0.13432229]
 [0.13714415]
 [0.1429714 ]
 [0.15812194]
 [0.15372768]
 [0.16670087]
 [0.20110112]
 [0.2105802 ]
 [0.19616789]
 [0.2933354 ]
 [0.6109506 ]
 [0.806383  ]
 [0.46566504]
 [0.28896496]
 [0.2930438 ]
 [0.29509133]
 [0.24810827]
 [0.2204186 ]
 [0.21481222]
 [0.19195458]
 [0.1838817 ]
 [0.20001274]
 [0.20379627]
 [0.21736231]
 [0.24591646]
 [0.31577423]
 [0.36506584]
 [0.51723695]
 [0.76219326]
```

```
 [0.89255047]
 [0.71645445]
 [0.41932178]
 [0.2604042 ]
 [0.23411953]
 [0.23519632]
 [0.20922226]
 [0.2225295 ]
 [0.24289244]
 [0.23235112]
 [0.22790155]
 [0.23391327]
 [0.2228126 ]
 [0.21372974]
 [0.19213483]
 [0.16652289]
 [0.16116342]
 [0.15899184]
 [0.14755708]
 [0.14301604]
 [0.14009029]
 [0.1282273 ]
 [0.13797867]
 [0.13847697]
 [0.13711083]
 [0.17531791]
 [0.17773172]
 [0.18688843]
 [0.28746358]
 [0.3872305 ]
 [0.6228456 ]
 [0.9647957 ]
 [1.0373837 ]
 [0.71484184]
 [0.4326741 ]
 [0.38893113]
 [0.46270046]
 [0.56280774]
 [0.6664135 ]
 [0.73218966]
 [0.6256144 ]
 [0.428805  ]
 [0.30987227]
 [0.22249568]
 [0.21095157]
 [0.27068412]
 [0.31057748]
 [0.39623392]
 [0.25595185]
```

```
  [0.22692922]
  [0.21752605]
  [0.22041032]
  [0.23974541]
  [0.22288567]
  [0.2166627 ]
  [0.22405997]
  [0.22262838]
  [0.24388817]
  [0.22708014]
  [0.21681026]
  [0.22084177]
  [0.21187705]
  [0.19374317]
  [0.19573626]
  [0.21344489]
  [0.23119631]
  [0.21900973]]

 [[0.31748894]
  [0.40424252]
  [0.6726155 ]
  [1.2152073 ]
  [1.717977  ]
  [2.185817  ]
  [2.5055587 ]
  [2.5237167 ]
  [2.1162922 ]
  [1.4827645 ]
  [0.9505055 ]
  [0.7082959 ]
  [0.6831619 ]
  [0.6156455 ]
  [0.39070988]
  [0.25719818]
  [0.21646768]
  [0.24411958]
  [0.29555124]
  [0.3368082 ]
  [0.47160256]
  [0.6187816 ]
  [0.50859153]
  [0.35138822]
  [0.27589017]
  [0.17937365]
  [0.15279105]
  [0.11660874]
  [0.09022456]
  [0.10343707]
```

```
 [0.09980142]
 [0.1161418 ]
 [0.12890148]
 [0.14090765]
 [0.13114429]
 [0.13837105]
 [0.13150102]
 [0.12704879]
 [0.13519311]
 [0.13766867]
 [0.14813697]
 [0.11460721]
 [0.10036623]
 [0.104958  ]
 [0.09577519]
 [0.10551369]
 [0.14031005]
 [0.16130328]
 [0.18485436]
 [0.18447712]
 [0.19472656]
 [0.22933775]
 [0.25618398]
 [0.2534855 ]
 [0.21756676]
 [0.2047196 ]
 [0.22912037]
 [0.23244315]
 [0.19854319]
 [0.15500495]
 [0.12902206]
 [0.1153577 ]
 [0.12159175]
 [0.11837053]
 [0.11834413]
 [0.13391829]
 [0.12752002]
 [0.12807202]
 [0.12705642]
 [0.10854107]
 [0.10901761]
 [0.10847497]
 [0.0987668 ]
 [0.09353477]
 [0.08800387]
 [0.08887482]
 [0.10323536]
 [0.11671388]
 [0.14184481]
```

```
 [0.15971753]
 [0.18534768]
 [0.1902703 ]
 [0.21480107]
 [0.24399623]
 [0.2868937 ]
 [0.28778622]
 [0.35575822]
 [0.6656916 ]
 [0.728205  ]
 [0.3842319 ]
 [0.23049396]
 [0.23684734]
 [0.25468585]
 [0.22053567]
 [0.1844868 ]
 [0.17499757]
 [0.15371603]
 [0.14906192]
 [0.14717847]
 [0.14797169]
 [0.16790292]
 [0.21094558]
 [0.31094137]
 [0.40796083]
 [0.4989625 ]
 [0.7188002 ]
 [0.7999368 ]
 [0.59084845]
 [0.38141996]
 [0.25680017]
 [0.26945108]
 [0.2488066 ]
 [0.21738464]
 [0.21326157]
 [0.19645855]
 [0.17817411]
 [0.17771673]
 [0.15802461]
 [0.1483798 ]
 [0.13826948]
 [0.14750397]
 [0.13656527]
 [0.12806022]
 [0.13147223]
 [0.10562009]
 [0.09773374]
 [0.09752899]
 [0.08551347]
```

```
  [0.08965027]
  [0.08873159]
  [0.10337138]
  [0.13043094]
  [0.13446188]
  [0.16725117]
  [0.28490895]
  [0.39064395]
  [0.48191524]
  [0.72223264]
  [0.78420967]
  [0.58994484]
  [0.37706435]
  [0.30731624]
  [0.3707291 ]
  [0.3434182 ]
  [0.3301212 ]
  [0.3338472 ]
  [0.269919  ]
  [0.20191625]
  [0.20795757]
  [0.15835184]
  [0.20093742]
  [0.3421708 ]
  [0.47708175]
  [0.3953686 ]
  [0.19119373]
  [0.16641268]
  [0.20940143]
  [0.21711788]
  [0.21383396]
  [0.18049309]
  [0.14735782]
  [0.13393056]
  [0.15700033]
  [0.16406745]
  [0.1599957 ]
  [0.17015451]
  [0.16728663]
  [0.16561249]
  [0.16649184]
  [0.181775  ]
  [0.16367939]
  [0.16713315]
  [0.18536249]]

 [[0.29095802]
  [0.3124774 ]
  [0.53174436]
  [1.021106  ]
```

```
 [1.2896245 ]
 [1.2501627 ]
 [1.1429392 ]
 [1.0383635 ]
 [0.9488546 ]
 [0.7378742 ]
 [0.6415813 ]
 [0.4686864 ]
 [0.40506554]
 [0.32480538]
 [0.19863781]
 [0.19874719]
 [0.18159685]
 [0.20279175]
 [0.24051327]
 [0.24839458]
 [0.29640517]
 [0.29978734]
 [0.26607904]
 [0.227036  ]
 [0.21236098]
 [0.18090525]
 [0.13881701]
 [0.12086219]
 [0.12061971]
 [0.13183933]
 [0.11894506]
 [0.11656553]
 [0.11034954]
 [0.09712142]
 [0.10382664]
 [0.10251737]
 [0.0879277 ]
 [0.09068167]
 [0.0936057 ]
 [0.10441506]
 [0.11599886]
 [0.09606147]
 [0.09463716]
 [0.09216142]
 [0.09040093]
 [0.1175372 ]
 [0.14036989]
 [0.18133143]
 [0.18922794]
 [0.15705004]
 [0.17975426]
 [0.21310872]
 [0.2565511 ]
```

```
 [0.25619882]
 [0.19714269]
 [0.16156355]
 [0.14563704]
 [0.1602337 ]
 [0.14174837]
 [0.10012472]
 [0.08732146]
 [0.10546798]
 [0.09554178]
 [0.10313648]
 [0.10840458]
 [0.10140419]
 [0.09062684]
 [0.09757763]
 [0.08414906]
 [0.07966375]
 [0.09487915]
 [0.10175097]
 [0.1040352 ]
 [0.09632915]
 [0.08783442]
 [0.11121941]
 [0.12102962]
 [0.13825423]
 [0.16019669]
 [0.1711559 ]
 [0.1780411 ]
 [0.19043306]
 [0.21742386]
 [0.19863755]
 [0.22462225]
 [0.22869179]
 [0.27740332]
 [0.38228247]
 [0.33898792]
 [0.21508321]
 [0.17321628]
 [0.17252871]
 [0.16478589]
 [0.15072623]
 [0.1446423 ]
 [0.12873906]
 [0.12688673]
 [0.12446058]
 [0.10968506]
 [0.11612093]
 [0.14097512]
 [0.16849774]
```

```
 [0.23627096]
 [0.2948969 ]
 [0.27425948]
 [0.3013092 ]
 [0.2760466 ]
 [0.2180365 ]
 [0.22993591]
 [0.18879715]
 [0.19611937]
 [0.19576707]
 [0.16863608]
 [0.15415648]
 [0.14594257]
 [0.13647199]
 [0.11590439]
 [0.12531054]
 [0.12633342]
 [0.11049771]
 [0.10906011]
 [0.11268669]
 [0.10779411]
 [0.10408729]
 [0.09931302]
 [0.1063714 ]
 [0.10998321]
 [0.08809012]
 [0.07224071]
 [0.07437056]
 [0.08533543]
 [0.09412628]
 [0.11310756]
 [0.12788117]
 [0.20078248]
 [0.2521201 ]
 [0.24313653]
 [0.25304252]
 [0.22730967]
 [0.23187128]
 [0.20382747]
 [0.18287891]
 [0.20040259]
 [0.18264103]
 [0.16241911]
 [0.13586628]
 [0.11643571]
 [0.14814773]
 [0.15386412]
 [0.10820502]
 [0.13588262]
```

```
  [0.36888066]
  [0.46748662]
  [0.3336223 ]
  [0.2066584 ]
  [0.18844923]
  [0.21291083]
  [0.18235436]
  [0.15069717]
  [0.10436463]
  [0.09788209]
  [0.09620965]
  [0.09294528]
  [0.11650056]
  [0.12251669]
  [0.13729537]
  [0.13238257]
  [0.11693501]
  [0.1416201 ]
  [0.15876734]
  [0.09420311]
  [0.10026884]
  [0.14770186]]

 [[0.20404729]
  [0.1878232 ]
  [0.23597041]
  [0.3315136 ]
  [0.3301407 ]
  [0.1900782 ]
  [0.08519745]
  [0.13199371]
  [0.18130618]
  [0.27821267]
  [0.264864  ]
  [0.2194151 ]
  [0.16056451]
  [0.14132291]
  [0.09363461]
  [0.12668651]
  [0.12220204]
  [0.12622935]
  [0.13482231]
  [0.09195   ]
  [0.06584811]
  [0.0538227 ]
  [0.0557546 ]
  [0.09847939]
  [0.13457519]
  [0.18817872]
```

```
 [0.11496907]
 [0.11825895]
 [0.11998808]
 [0.10866648]
 [0.12388676]
 [0.09537286]
 [0.06108934]
 [0.04909962]
 [0.04896319]
 [0.05380654]
 [0.04886514]
 [0.05880803]
 [0.06250942]
 [0.08592832]
 [0.07437241]
 [0.05593461]
 [0.06323981]
 [0.04488289]
 [0.03366774]
 [0.05421543]
 [0.09522134]
 [0.11520398]
 [0.10115522]
 [0.07536376]
 [0.07060796]
 [0.06884342]
 [0.08162975]
 [0.09075397]
 [0.10016102]
 [0.08631521]
 [0.05898386]
 [0.09065902]
 [0.0873605 ]
 [0.0600791 ]
 [0.04526329]
 [0.05798846]
 [0.05906278]
 [0.06403816]
 [0.06028825]
 [0.04294968]
 [0.05048442]
 [0.09544981]
 [0.06157422]
 [0.06760508]
 [0.09561777]
 [0.08631599]
 [0.08180231]
 [0.06395072]
 [0.06944692]
```

```
 [0.0940097 ]
 [0.09080833]
 [0.10841852]
 [0.13791138]
 [0.14536881]
 [0.14061159]
 [0.12961316]
 [0.13904321]
 [0.12745929]
 [0.16132864]
 [0.1474644 ]
 [0.09303224]
 [0.10844201]
 [0.07935363]
 [0.061378  ]
 [0.07733321]
 [0.08596134]
 [0.08567166]
 [0.09950745]
 [0.08998531]
 [0.07569593]
 [0.07441419]
 [0.06540656]
 [0.07007694]
 [0.06486493]
 [0.08708704]
 [0.0863086 ]
 [0.10991949]
 [0.08482045]
 [0.07440275]
 [0.05338979]
 [0.07497293]
 [0.10091674]
 [0.09017336]
 [0.10048264]
 [0.08122379]
 [0.09031451]
 [0.08197743]
 [0.0730617 ]
 [0.08541495]
 [0.10724193]
 [0.06824529]
 [0.07497376]
 [0.08773518]
 [0.08917624]
 [0.05649298]
 [0.05662179]
 [0.05806226]
 [0.05643308]
```

```
  [0.0645054 ]
  [0.07261181]
  [0.0816884 ]
  [0.05679458]
  [0.04432398]
  [0.03463554]
  [0.02965558]
  [0.0475719 ]
  [0.07842004]
  [0.06573355]
  [0.10673785]
  [0.1279499 ]
  [0.14134169]
  [0.12724441]
  [0.10437554]
  [0.10547143]
  [0.09066516]
  [0.08079696]
  [0.03372598]
  [0.03826475]
  [0.06763136]
  [0.07515854]
  [0.06452793]
  [0.09360009]
  [0.09225035]
  [0.06310791]
  [0.05599028]
  [0.12599355]
  [0.18013775]
  [0.11526555]
  [0.0836888 ]
  [0.10670954]
  [0.15367472]
  [0.11905342]
  [0.0912928 ]
  [0.06766337]
  [0.03752816]
  [0.05677521]
  [0.04463506]
  [0.06126255]
  [0.08848923]
  [0.11820102]
  [0.12033641]
  [0.08341604]
  [0.08811849]
  [0.06576234]
  [0.02904934]
  [0.0547151 ]
  [0.12757355]]]
```

```
ipd.Audio('cleaned_audio.wav')
```

```
<IPython.lib.display.Audio object>
```

```
ipd.Audio(noisy_audio_filepath)
```

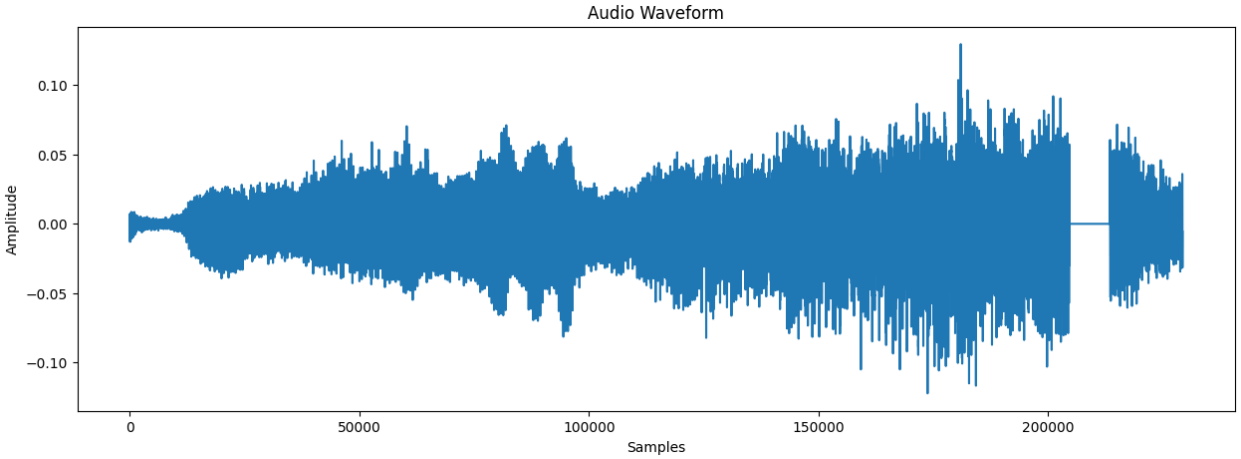
```
<IPython.lib.display.Audio object>
```

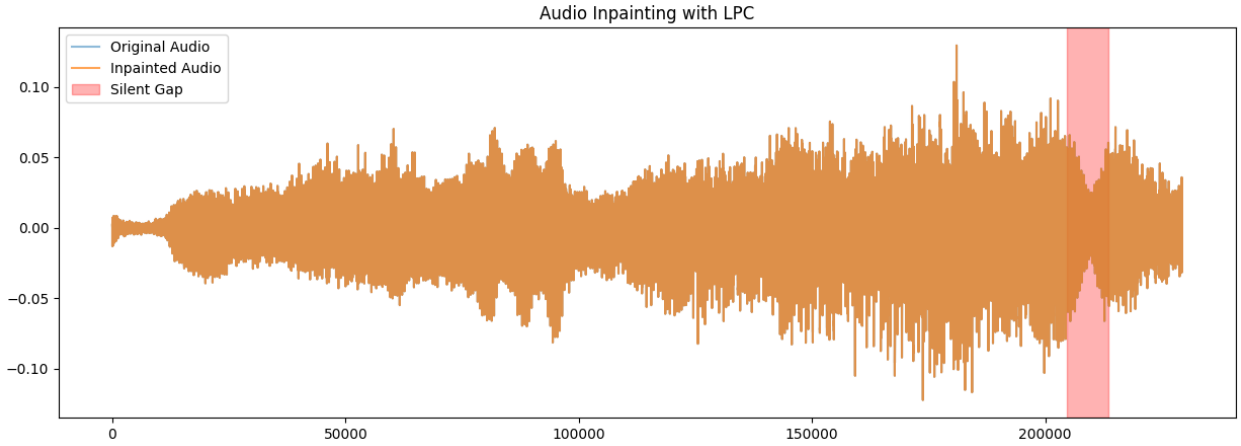
```
import os
import torchaudio
import numpy as np
import matplotlib.pyplot as plt
from scipy.signal import lfiltic, lfilter
from IPython.display import Audio, display
from spectrum.burg import _arburg2

def detect_silent_gap(audio, sr, silence_duration_ms=200,
threshold=0.001):
    silence_samples = int((silence_duration_ms / 1000) * sr)
    abs_audio = np.abs(audio)

    for i in range(len(audio) - silence_samples):
        window = abs_audio[i : i + silence_samples]
        if np.mean(window) < threshold:
            gap_start = i
            gap_end = i + silence_samples
            return gap_start, gap_end

    print("No sufficiently long silent gap found.")
    return None

def LPC(previous_sig, next_sig, gap_start, gap_end, lpc_order):
    target_length = gap_end - gap_start

    ab, _, _ = _arburg2(previous_sig, lpc_order)
    Zb = lfiltic(b=[1], a=ab, y=previous_sig[:-lpc_order-1:-1])
    forw_pred, _ = lfilter(b=[1], a=ab, x=np.zeros((target_length)),
zi=Zb)

    next_sig = np.flipud(next_sig)
    af, _, _ = _arburg2(next_sig, lpc_order)
    Zf = lfiltic([1], af, next_sig[:-lpc_order-1:-1])
    backw_pred, _ = lfilter([1], af, np.zeros((target_length)), zi=Zf)
    backw_pred = np.flipud(backw_pred)

    t = np.linspace(0, np.pi/2, target_length)
    sqCos = np.cos(t)**2
    sigout = sqCos * forw_pred + np.flipud(sqCos) * backw_pred
    return sigout

def inpaint_audio_with_lpc(audio, gap_start, gap_end, lpc_order):
    context_size = 3000
    previous_sig = audio[max(0, gap_start - context_size) : gap_start]
    next_sig = audio[gap_end : min(len(audio), gap_end +
context_size)]

    if len(previous_sig) == 0 or len(next_sig) == 0:
        print("Insufficient context for LPC-based inpainting.")
```

```
        return audio

    lpc_signal = LPC(previous_sig, next_sig, gap_start, gap_end,
lpc_order)
    audio[gap_start:gap_end] = lpc_signal
    return audio

def process_audio_file_with_lpc(file_path, sr, lpc_order,
silence_duration_ms=200, threshold=1e-4):
    audio, _ = torchaudio.load(file_path)
    audio = audio.squeeze().numpy()

    print(f"Input audio shape: {audio.shape}, duration: {len(audio) /
sr:.2f} seconds")

    gap = detect_silent_gap(audio, sr, silence_duration_ms, threshold)
    if gap is None:
        return audio, audio, None, None

    gap_start, gap_end = gap
    print(f"Detected silent gap from sample {gap_start} to
{gap_end}.")

    inpainted_audio = inpaint_audio_with_lpc(audio, gap_start,
gap_end, lpc_order)

    print(f"Processed audio shape: {inpainted_audio.shape}, duration:
{len(inpainted_audio) / sr:.2f} seconds")
    return audio, inpainted_audio, gap_start, gap_end

def visualize_and_play(original, inpainted, gap_start, gap_end, sr):
    plt.figure(figsize=(15, 5))
    plt.plot(original, label="Original Audio", alpha=0.5)
    plt.plot(inpainted, label="Inpainted Audio", alpha=0.7)
    if gap_start is not None and gap_end is not None:
        plt.axvspan(gap_start, gap_end, color='red', alpha=0.3,
label="Silent Gap")
    plt.legend()
    plt.title("Audio Inpainting with LPC")
    plt.show()

    print(f"Original audio duration: {len(original) / sr:.2f}
seconds")
    print(f"Inpainted audio duration: {len(inpainted) / sr:.2f}
seconds")

    print("Original Audio:")
    display(Audio(original, rate=sr))

    print("Inpainted Audio:")
```

```
    display(Audio(inpainted, rate=sr))

if __name__ == "__main__":
    file_path =
'/Users/johannasmriti/Downloads/musicnet/data_with_gaps/1727_part11.wa
v'
    sr = 44100
    silence_duration_ms = 200
    lpc_order = 600

    original_audio, inpainted_audio, gap_start, gap_end =
process_audio_file_with_lpc(
        file_path, sr, lpc_order,
silence_duration_ms=silence_duration_ms
    )

    def plot_audio(audio, sr):
        plt.figure(figsize=(15, 5))
        plt.plot(audio)
        plt.title("Audio Waveform")
        plt.xlabel("Samples")
        plt.ylabel("Amplitude")
        plt.show()

    audio, _ = torchaudio.load(file_path)
    plot_audio(audio.squeeze().numpy(), sr)

    visualize_and_play(original_audio, inpainted_audio, gap_start,
gap_end, sr)
```

```
Input audio shape: (229317,), duration: 5.20 seconds
Detected silent gap from sample 204641 to 213461.
Processed audio shape: (229317,), duration: 5.20 seconds

/var/folders/yz/1fd13cns6gvbhv48blbqlsjh0000gn/T/
ipykernel_14016/1517374373.py:51: ComplexWarning: Casting complex
values to real discards the imaginary part
  audio[gap_start:gap_end] = lpc_signal
```






```
Original audio duration: 5.20 seconds
Inpainted audio duration: 5.20 seconds
Original Audio:

<IPython.lib.display.Audio object>

Inpainted Audio:

<IPython.lib.display.Audio object>
```