In [ ]:
```
# --- Google Colab Setup ---
from google.colab import files  # For uploading/downloading files in Colab
!pip install pyabf              # Library to load .abf (Axon Binary Files)
!pip install PyWavelets         # Library for wavelet transforms

# --- Data Handling ---
import pandas as pd             # For data manipulation
import numpy as np              # For numerical operations

# --- Signal Processing ---
import pyabf                    # For loading electrophysiological .abf files
from scipy.signal import butter, filtfilt  # For bandpass filtering
import pywt                     # For wavelet decomposition
from scipy.stats import kstest # For statistical testing (KS-test)

# --- Machine Learning & Analysis ---
from sklearn.model_selection import train_test_split  # For train-test split
from sklearn.decomposition import PCA                # For dimensionality reduction
import plotly.express as px

from sklearn.cluster import KMeans, SpectralClustering    # For clustering
import hdbscan
from sklearn.metrics import silhouette_score         # For cluster quality assessment

# --- Plotting ---
import matplotlib.pyplot as plt       # For static plotting
import plotly.express as px           # For interactive plotting
import seaborn as sns

# --- Miscellaneous ---
import pickle                         # For saving/loading models or results
from matplotlib.cm import get_cmap
import os
```

```
Collecting pyabf
  Downloading pyabf-2.3.8-py3-none-any.whl.metadata (3.0 kB)
Requirement already satisfied: matplotlib>=2.1.0 in /usr/local/lib/python3.11/dist-packages (from pyabf) (3.10.0)
Requirement already satisfied: numpy>=1.13.3 in /usr/local/lib/python3.11/dist-packages (from pyabf) (2.0.2)
Requirement already satisfied: pytest>=3.0.7 in /usr/local/lib/python3.11/dist-packages (from pyabf) (8.3.5)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.1.0->p
yabf) (1.3.2)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.1.0->pyab
f) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.1.0->
pyabf) (4.57.0)
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.1.0->
pyabf) (1.4.8)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.1.0->py
abf) (24.2)
Requirement already satisfied: pillow>=8 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.1.0->pyabf)
(11.2.1)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.1.0->p
yabf) (3.2.3)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.11/dist-packages (from matplotlib>=2.1.
0->pyabf) (2.9.0.post0)
Requirement already satisfied: iniconfig in /usr/local/lib/python3.11/dist-packages (from pytest>=3.0.7->pyabf) (2.1.
0)
Requirement already satisfied: pluggy<2,>=1.5 in /usr/local/lib/python3.11/dist-packages (from pytest>=3.0.7->pyabf)
(1.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.7->matplo
tlib>=2.1.0->pyabf) (1.17.0)
Downloading pyabf-2.3.8-py3-none-any.whl (53 kB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.0/53.0 kB 2.0 MB/s eta 0:00:00
Installing collected packages: pyabf
Successfully installed pyabf-2.3.8
Collecting PyWavelets
  Downloading pywavelets-1.8.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (9.0 kB)
Requirement already satisfied: numpy<3,>=1.23 in /usr/local/lib/python3.11/dist-packages (from PyWavelets) (2.0.2)
Downloading pywavelets-1.8.0-cp311-cp311-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (4.5 MB)
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 4.5/4.5 MB 19.9 MB/s eta 0:00:00
Installing collected packages: PyWavelets
Successfully installed PyWavelets-1.8.0
```

In [ ]:
```
'''==========================
        FUNCTIONS BLOCK
=========================='''
```

```python
# ---------------------------
# 1. ABF File Handling & Inspection
# ---------------------------

def load_abf_data(filepath):
    """
    Load ABF file and return metadata + all sweep data.
    """
    abf = pyabf.ABF(filepath)
    all_sweeps = [abf.setSweep(sweep) or abf.sweepY.copy() for sweep in range(abf.sweepCount)]
    data = np.array(all_sweeps)
    print(f"Data: {data.shape} \nY-Sweep: {abf.sweepY.shape}")
    return {
        "data": data,
        "sampling_rate": abf.dataRate,
        "sweep_count": abf.sweepCount,
        "units": abf.sweepUnitsY,
        "channel_name": abf.channelList,
        "y-axis": abf.sweepY,
        "sweep_units": abf.sweepUnitsY
    }

def check_channels(filepath):
    """
    Print all available channels in ABF file and sample values from each.
    """
    abf = pyabf.ABF(filepath)
    print(f"Available channels: {abf.channelList}\n")
    for ch in range(abf.channelCount):
        abf.setSweep(sweepNumber=0, channel=ch)
        print(f"Channel {ch} ({abf.channelList[ch]}): First value of sweepY = {abf.sweepY[0]}")

def inspect_abf(filepath):
    """
    Show protocol and metadata of an ABF file, plot sweep 0.
    """
    abf = pyabf.ABF(filepath)
    print("Filepath:", filepath)
    print("Protocol:", abf.protocol)
    print("Sweep count:", abf.sweepCount)
    print("Channel count:", abf.channelCount)
```

```
    print("Channel names (adc):", abf.adcNames)
    print("Channel units:", abf.adcUnits)
    print("Sampling rate (Hz):", abf.dataRate)
    print("Sweep point count:", abf.sweepPointCount)
    print("Sweep length (s):", abf.sweepPointCount / abf.dataRate)
    print("Tag times (sec):", abf.tagTimesSec)
    print("Tag comments:", abf.tagComments)

    abf.setSweep(0)
    plt.plot(abf.sweepX, abf.sweepY)
    plt.title(f"Sweep 0 | Units: {abf.sweepUnitsY}")
    plt.xlabel("Time (s)")
    plt.ylabel(abf.sweepUnitsY)
    plt.show()
    print("Mode:", "Episodic (Sweeps)" if abf.sweepCount > 0 else "Continuous (Gap Free?)")

# ---------------------------
# 2. Signal Preprocessing
# ---------------------------

def bandpass_filter(data, lowcut, highcut, fs, order=3):
    """
    Apply a bandpass filter to a 1D signal.
    """
    nyq = 0.5 * fs
    b, a = butter(order, [lowcut / nyq, highcut / nyq], btype="band")
    return filtfilt(b, a, data)

# ---------------------------
# 3. Peak Detection & Refinement
# ---------------------------
#EPSC
def detect_minima_v2(signal, delta=-15.0, slope_thresh=-0.3, distance=20,
                     stop=100_000, step=10_000, visualize_all=True):
    """
    Detect **minima** using amplitude and slope thresholds.

    Returns:
    - Dictionary with detected trough (minimum) indices and detection parameters.
    """
    minima_indices, last_trough = [], -distance
    for i in range(1, len(signal) - 1):
```

```
        if signal[i] < delta and (signal[i] - signal[i - 1]) < slope_thresh:
            if i - last_trough >= distance:
                minima_indices.append(i)
                last_trough = i

    minima_indices = np.array(minima_indices)
    print(f"🔍 Total detected minima: {len(minima_indices)}")

    vis_minima = minima_indices if visualize_all else minima_indices[:10]
    for idx in vis_minima:
        start = max(0, idx - step // 2)
        end = min(len(signal), idx + step // 2)
        plt.figure(figsize=(12, 3))
        plt.plot(np.arange(start, end), signal[start:end], alpha=0.6)
        plt.plot(idx, signal[idx], 'bo', label="Detected Minimum")
        plt.title(f"Amplitude + Slope Minima Detection | Samples {start}-{end}")
        plt.xlabel("Sample Index")
        plt.grid(True)
        plt.legend()
        plt.tight_layout()
        plt.show()

    return {
        "peak_indices": minima_indices,
        "params": {"delta": delta, "slope_thresh": slope_thresh, "distance": distance}
    }

def refine_minima_to_apex(signal, minima_indices, search_radius=90):
    """
    Refine each detected minimum to the local minimum within a window.
    """
    refined_minima = []
    for i in minima_indices:
        start = max(0, i - search_radius)
        end = min(len(signal), i + search_radius)
        refined_minima.append(start + np.argmin(signal[start:end]))
    return np.array(refined_minima)

def visualize_refined_minima(signal, refined_minima, num_minima=5, window=100):
    """
    Plot signal centered on each refined EPSC-like trough.
    """
```

```
    for idx in refined_minima[:num_minima]:
        start = max(0, idx - window // 2)
        end = min(len(signal), idx + window // 2)
        plt.figure(figsize=(10, 3))
        plt.plot(np.arange(start, end), signal[start:end], label="Filtered Signal")
        plt.axvline(idx, color='blue', linestyle='--', label='Refined Minimum')
        plt.title(f"Refined EPSC Minimum at Index {idx}")
        plt.xlabel("Sample Index")
        plt.grid(True)
        plt.legend()
        plt.tight_layout()
        plt.show()


### iPSC
def detect_peaksv2(signal, delta=15.0, slope_thresh=0.3, distance=20,
                   stop=100_000, step=10_000, visualize_all=True): #    For iPSC
    """
    Detect peaks using amplitude and slope thresholds. IPSC

    Parameters:
    - visualize_all: if False, only the first 10 peaks are visualized (not truncated from the results).

    Returns:
    - Dictionary containing all peak indices and detection parameters.
    """
    peak_indices, last_peak = [], -distance
    for i in range(1, len(signal) - 1):
        if signal[i] > delta and (signal[i] - signal[i - 1]) > slope_thresh:
            if i - last_peak >= distance:
                peak_indices.append(i)
                last_peak = i

    peak_indices = np.array(peak_indices)
    print(f"🔎 Total detected peaks: {len(peak_indices)}")

    # Visualization only (does not modify returned data)
    vis_peaks = peak_indices if visualize_all else peak_indices[:10]

    for peak in vis_peaks:
        start = max(0, peak - step // 2)
        end = min(len(signal), peak + step // 2)
```

```
        plt.figure(figsize=(12, 3))
        plt.plot(np.arange(start, end), signal[start:end], alpha=0.6)
        plt.plot(peak, signal[peak], 'ro', label="Detected Peak")
        plt.title(f"Amplitude + Slope Detection | Samples {start}-{end}")
        plt.xlabel("Sample Index")
        plt.grid(True)
        plt.legend()
        plt.tight_layout()
        plt.show()

    return {
        "peak_indices": peak_indices,
        "params": {"delta": delta, "slope_thresh": slope_thresh, "distance": distance}
    }



def refine_peaks_to_apex(signal, peak_indices, search_radius=90):
    """
    Refine each detected peak to its local maximum within a window.
    """
    refined_peaks = []
    for i in peak_indices:
        start = max(0, i - search_radius)
        end = min(len(signal), i + search_radius)
        refined_peaks.append(start + np.argmax(signal[start:end]))
    return np.array(refined_peaks)

def visualize_refined_peaks(signal, refined_peaks, num_peaks=5, window=100):
    """
    Plot signal centered on each refined peak.
    """
    for idx in refined_peaks[:num_peaks]:
        start = max(0, idx - window // 2)
        end = min(len(signal), idx + window // 2)
        plt.figure(figsize=(10, 3))
        plt.plot(np.arange(start, end), signal[start:end], label="Filtered Signal")
        plt.axvline(idx, color='red', linestyle='--', label='Refined Peak')
        plt.title(f"Refined Peak at Index {idx}")
        plt.xlabel("Sample Index")
        plt.grid(True)
        plt.legend()
```

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

# ---------------------------
# 4. Visualization
# ---------------------------

def plot_mean_waveforms_per_cluster(waveforms, labels, title="Mean Waveform per Cluster",
                                    normalize=False, ax=None):
    """
    Plot average waveform of each cluster.
    """
    unique_labels = np.unique(labels)
    if ax is None:
        fig, ax = plt.subplots(figsize=(10, 5))

    for label in unique_labels:
        cluster_waveforms = waveforms[labels == label]
        if normalize:
            cluster_waveforms = np.array([
                w / np.max(np.abs(w)) if np.max(np.abs(w)) != 0 else w
                for w in cluster_waveforms
            ])
        ax.plot(np.mean(cluster_waveforms, axis=0), label=f"Cluster {label}")

    ax.set_title(title)
    ax.set_xlabel("Sample Index")
    ax.set_ylabel("Normalized Amplitude" if normalize else "Amplitude")
    ax.legend()
    ax.grid(True)

def plot_cluster_means_from_dict(waveform_dict, normalize=True):
    """
    Plot mean waveform for each cluster across multiple methods.
    """
    n = len(waveform_dict)
    fig, axs = plt.subplots(1, n, figsize=(5 * n, 4), sharey=True)
    axs = axs if isinstance(axs, np.ndarray) else [axs]

    for ax, (method, (waveforms, labels)) in zip(axs, waveform_dict.items()):
        plot_mean_waveforms_per_cluster(waveforms, labels, title=method,
                                        normalize=normalize, ax=ax)
```

```
    fig.suptitle("Mean Waveform per Cluster by Method", fontsize=16)
    plt.tight_layout()
    plt.show()

def plot_amplitude_distribution_across_clusters(waveforms, clustering_results):
    """
    Create boxplots of peak amplitudes for each cluster across methods.
    """
    peak_amplitudes = np.max(waveforms, axis=1)
    fig, axs = plt.subplots(1, 3, figsize=(18, 5), sharey=True)

    for i, (method, labels) in enumerate(clustering_results.items()):
        df = pd.DataFrame({
            "Cluster": labels,
            "Peak Amplitude (pA)": peak_amplitudes
        })
        sns.boxplot(data=df, x="Cluster", y="Peak Amplitude (pA)", ax=axs[i])
        axs[i].set_title(f"{method.upper()} Clustering")
        axs[i].grid(True)

    plt.tight_layout()
    plt.show()


def cluster_visualize_overlayed_waveforms(waveforms, clustering_results, method="kmeans", cluster_number=0, normalize
    """
    Plots overlayed waveforms for a specific cluster from a given method.

    Parameters:
        waveforms (np.array): Waveforms (normalized or raw), shape (n_waveforms, n_samples)
        clustering_results (dict): Dictionary with method keys and label arrays
        method (str): Clustering method name, e.g., 'kmeans'
        cluster_number (int): Cluster index to visualize
        normalize (bool): Normalize each waveform individually before plotting
    """
    if method not in clustering_results:
        raise ValueError(f"❌ Method '{method}' not found in clustering results.")

    cluster_indices = np.where(clustering_results[method] == cluster_number)[0]

    if len(cluster_indices) == 0:
```

```
        print(f"⚠️ No waveforms found in Cluster {cluster_number} for method '{method}'.")
        return

    print(f"📊 [{method.upper()}] Overlaying {len(cluster_indices)} waveforms from Cluster {cluster_number}")

    plt.figure(figsize=(5, 5))
    for idx in cluster_indices:
        waveform = waveforms[idx]
        if normalize:
            waveform = waveform / np.max(np.abs(waveform)) if np.max(np.abs(waveform)) != 0 else waveform
        plt.plot(waveform, alpha=0.6)

    plt.title(f"{method.upper()} Cluster {cluster_number} — Waveforms")
    plt.xlabel("Sample Index")
    plt.ylabel("Amplitude" + (" (Normalized)" if normalize else ""))
    plt.grid(True)
    plt.tight_layout()
    plt.show()


# ---------------------------
# 5. Add-ons
# ---------------------------

# -- Inspect individual and grouped waveforms
def plot_waveform_inspection(waveforms, normalized_waveforms, index_wave=3, n_examples=5):
    fig, axs = plt.subplots(1, 3, figsize=(19, 5))

    # 1. Single raw waveform
    axs[0].plot(waveforms[index_wave])
    axs[0].set_title("Non-Normalized Waveform")
    axs[0].set_xlabel("Sample Index")
    axs[0].set_ylabel("Amplitude (pA)")
    axs[0].grid(True)

    # 2. Same waveform after normalization
    axs[1].plot(normalized_waveforms[index_wave])
    axs[1].set_title("Normalized Waveform")
    axs[1].set_xlabel("Sample Index")
    axs[1].set_ylabel("Normalized Amplitude")
    axs[1].grid(True)
```

```
    # 3. Plot N normalized waveforms from index
    end_index = min(index_wave + n_examples, len(normalized_waveforms))
    for i in range(index_wave, end_index):
        axs[2].plot(normalized_waveforms[i], label=f"Waveform {i}")
    axs[2].set_title(f"Normalized Waveforms [{index_wave} to {end_index - 1}]")
    axs[2].set_xlabel("Sample Index")
    axs[2].set_ylabel("Amplitude")
    axs[2].legend()
    axs[2].grid(True)

    plt.tight_layout()
    plt.show()

# ================================
# 📉 PCA CLUSTER PLOTS
# ================================

def plot_pca_clusters(X_pca, clustering_results, titles=None):
    """
    Plot PCA-reduced data for different clustering methods.
    """
    methods = list(clustering_results.keys())
    n_methods = len(methods)

    if titles is None:
        titles = [method.capitalize() + " Clustering" for method in methods]

    fig, axs = plt.subplots(1, n_methods, figsize=(6 * n_methods, 5), sharey=True)
    if n_methods == 1:
        axs = [axs]

    for ax, method, title in zip(axs, methods, titles):
        labels = clustering_results[method]
        for label in np.unique(labels):
            mask = labels == label
            ax.scatter(X_pca[mask, 0], X_pca[mask, 1], label=f"Cluster {label}", alpha=0.6)
        ax.set_title(title)
        ax.set_xlabel("PC1")
        ax.set_ylabel("PC2")
        ax.legend()
        ax.grid(True)
```

```
    fig.suptitle("PCA Cluster Visualization", fontsize=16)
    plt.tight_layout()
    plt.show()

# ================================
# 📊 SILHOUETTE SCORES
# ================================

def compute_silhouette_scores(X_pca, clustering_results):
    """
    Compute silhouette scores for each clustering result.
    """
    scores = {}
    for method, labels in clustering_results.items():
        if method.lower() == 'hdbscan':
            valid_mask = labels != -1
            if len(np.unique(labels[valid_mask])) > 1 and valid_mask.sum() > 0:
                score = silhouette_score(X_pca[valid_mask], labels[valid_mask])
            else:
                score = None
        else:
            score = silhouette_score(X_pca, labels)
        scores[method.capitalize()] = score
    return scores

def plot_silhouette_scores(scores):
    """
    Bar plot of silhouette scores with annotations.
    """
    methods = list(scores.keys())
    values = [v if v is not None else 0 for v in scores.values()]
    colors = ['steelblue', 'darkorange', 'seagreen']

    plt.figure(figsize=(8, 5))
    bars = plt.bar(methods, values, color=colors)
    plt.ylabel("Silhouette Score")
    plt.title("Clustering Quality (Silhouette Scores)")
    plt.ylim(0, 1)

    for bar, method in zip(bars, methods):
        score = scores[method]
        label = f"{score:.2f}" if score is not None else "N/A"
```

```
        plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() + 0.02, label,
                 ha='center', va='bottom', fontsize=10)

    plt.grid(True, axis='y', linestyle='--', alpha=0.5)
    plt.tight_layout()
    plt.show()

def print_silhouette_summary(scores, clustering_results):
    """
    Print qualitative summary of clustering quality.
    """
    print("🧠 Silhouette Score Summary:\n")
    for method, score in scores.items():
        method_key = method.lower()
        labels = clustering_results[method_key]

        if method_key == 'hdbscan':
            labels = labels[labels != -1]

        n_clusters = len(np.unique(labels))

        if score is None or n_clusters <= 1:
            print(f"❌ {method}: No valid clusters (score unavailable)")
        else:
            interpretation = (
                "🌟 Excellent" if score > 0.7 else
                "✅ Good" if score > 0.5 else
                "⚠️ Weak" if score > 0.2 else
                "🔻 Poor"
            )
            print(f"- {method}: {score:.3f} with {n_clusters} cluster(s) → {interpretation}")

# ========================================
# 🔍 Clustering Overlays on Raw Signal Trace
# ========================================

def create_cluster_map(peak_indices, cluster_labels, signal_length, pad=3):
    """
    Maps clustering labels to their corresponding peak locations in the signal timeline.

    Parameters:
    - peak_indices: list/array of peak locations
```

```
    - cluster_labels: clustering label for each peak
    - signal_length: total length of the signal
    - pad: number of samples before and after peak to assign the label

    Returns:
    - cluster_map: 1D array (length = signal_length), values = cluster labels
                   Default background = -99 (unlabeled)
    """
    cluster_map = np.full(signal_length, -99)  # -99 = background/unassigned
    for i, peak in enumerate(peak_indices):
        if peak - pad >= 0 and peak + pad < signal_length:
            cluster_map[peak - pad : peak + pad] = cluster_labels[i]
    return cluster_map


def get_color_palette(n_clusters, cmap_name="tab10"):
    """
    Generate a list of distinct colors based on number of clusters.

    Parameters:
    - n_clusters: total number of clusters
    - cmap_name: colormap name from matplotlib (e.g., 'tab10', 'Set2', etc.)

    Returns:
    - list of RGBA color tuples
    """
    cmap = get_cmap(cmap_name)
    return [cmap(i % cmap.N) for i in range(n_clusters)]


def plot_cluster_maps_across_methods(filtered_data, peak_indices, clustering_results, start=0, end=100_000, pad=3):
    """
    Plot a zoomed-in region of the filtered trace with clustering overlays from multiple methods.

    Parameters:
    - filtered_data: 1D numpy array of filtered voltage trace
    - peak_indices: array of peak positions
    - clustering_results: dict of clustering method -> cluster labels
    - start, end: window of signal to visualize
    - pad: number of samples to spread each cluster label around peak

    Output:
```

```
    - Multi-panel plot showing trace and clustering overlays from all methods
    """
    methods = list(clustering_results.keys())
    n_methods = len(methods)

    fig, axs = plt.subplots(n_methods + 1, 1, figsize=(16, 3.2 * (n_methods + 1)), sharex=True)

    x_range = np.arange(start, end)

    # Panel 0: Raw filtered signal
    axs[0].plot(x_range, filtered_data[start:end], color='black', label="Filtered Trace", alpha=0.6)
    axs[0].set_title("Filtered Trace (Baseline)")
    axs[0].set_ylabel("Amplitude (pA)")
    axs[0].grid(True)

    # Cluster overlays per method
    for idx, method in enumerate(methods):
        labels = clustering_results[method]
        cluster_map = create_cluster_map(peak_indices, labels, len(filtered_data), pad=pad)

        axs[idx + 1].plot(x_range, filtered_data[start:end], color='lightgray', alpha=0.4, label="Filtered Trace")

        unique_labels = np.unique(labels)
        n_clusters = len(unique_labels[unique_labels != -1])
        color_map = get_color_palette(n_clusters)

        for i, label in enumerate(unique_labels):
            mask = cluster_map[start:end] == label
            if label == -1:
                color = 'red'
                label_str = "Noise"
            else:
                color = color_map[i % len(color_map)]
                label_str = f"Cluster {label}"

            axs[idx + 1].scatter(
                x_range[mask],
                filtered_data[start:end][mask],
                s=10,
                color=color,
                alpha=0.7,
                label=label_str,
```

```
                marker='o'
            )

        axs[idx + 1].set_title(f"{method.upper()} Clustering Overlay")
        axs[idx + 1].set_ylabel("Amplitude (pA)")
        axs[idx + 1].legend(loc="upper right", fontsize=9)
        axs[idx + 1].grid(True)

    axs[-1].set_xlabel("Sample Index")
    fig.suptitle("Clustered Event Overlays Across Clustering Methods", fontsize=16, y=1.01)
    plt.tight_layout()
    plt.show()

def print_pipeline_params(param_dict):
    """
    Nicely print the parameters used in the waveform clustering pipeline.
    """
    print("\n====== ⚙️ Pipeline Parameter Summary ======")
    for k, v in param_dict.items():
        print(f"• {k}: {v}")
    print("=" * 40 + "\n")


def load_and_filter_abf(
    abf_filepath,
    low_pass=5,
    high_pass=3000,
    filter_order=3,
    plot=True
):
    """
    Load entire ABF trace, filter it, and optionally return raw/filtered traces with plots.
    """
    abf_result = load_abf_data(abf_filepath)
    raw = abf_result["y-axis"]
    fs = abf_result["sampling_rate"]

    filtered = bandpass_filter(
        raw,
        lowcut=low_pass,
        highcut=high_pass,
        fs=fs,
```

```
        order=filter_order
    )
    if plot:
        fig, axs = plt.subplots(2, 1, figsize=(12, 6), sharex=True)
        axs[0].plot(raw, color="steelblue")
        axs[0].set_title("Raw Trace")
        axs[0].set_ylabel("Amplitude (pA)")

        axs[1].plot(filtered, color="orange")
        axs[1].set_title(f"Filtered Trace ({low_pass}-{high_pass} Hz)")
        axs[1].set_xlabel("Sample Index")
        axs[1].set_ylabel("Amplitude (pA)")

        plt.tight_layout()
        plt.show()

    return {
        "raw": raw,
        "filtered": filtered,
        "sampling_rate": fs
    }
```

In [ ]:

```
# Mount Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Define your base folder path
base_path = "/content/drive/My Drive/Python Projects/iPSC"
print("Folders inside:", os.listdir(base_path))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-3-874ec34ddddd> in <cell line: 0>()
      1 # Mount Google Drive
      2 from google.colab import drive
----> 3 drive.mount('/content/drive')
      4
      5 # Define your base folder path

/usr/local/lib/python3.11/dist-packages/google/colab/drive.py in mount(mountpoint, force_remount, timeout_ms, readonl
y)
     98 def mount(mountpoint, force_remount=False, timeout_ms=120000, readonly=False):
     99   """Mount your Google Drive at the specified mountpoint path."""
--> 100   return _mount(
    101       mountpoint,
    102       force_remount=force_remount,

/usr/local/lib/python3.11/dist-packages/google/colab/drive.py in _mount(mountpoint, force_remount, timeout_ms, epheme
ral, readonly)
    277             'https://research.google.com/colaboratory/faq.html#drive-timeout'
    278         )
--> 279       raise ValueError('mount failed' + extra_reason)
    280     elif case == 4:
    281       # Terminate the DriveFS binary before killing bash.

ValueError: mount failed
```

In [ ]:

```python
# 1. Loaded 20 folders into data_dict
# Each folder contains
# "abf" --> object with trace
# "csv" --> pandas df with atleat one columns related to peaks


# Get all folders that match the pattern "T1 Set N", sorted by the number
folder_names = sorted(
    [f for f in os.listdir(base_path) if f.startswith("T1 Set")],
    key=lambda x: int(x.split()[-1])  # Sort by the number at the end
)

# Dictionary to store data
data_dict = {}
```

```
# Loop through each folder
for folder in folder_names:
    folder_path = os.path.join(base_path, folder)

    # Find the ABF and CSV files
    abf_files = [f for f in os.listdir(folder_path) if f.endswith(".abf")]
    csv_files = [f for f in os.listdir(folder_path) if f.endswith(".csv")]

    if not abf_files or not csv_files:
        print(f"⚠️ Skipping {folder} — ABF or CSV file missing")
        continue

    abf_path = os.path.join(folder_path, abf_files[0])
    csv_path = os.path.join(folder_path, csv_files[0])

    # Load the files
    abf = pyabf.ABF(abf_path)
    df = pd.read_csv(csv_path)

    # Store in dictionary
    data_dict[folder] = {"abf": abf, "csv": df}

print(f"\n✅ Loaded data for {len(data_dict)} folders.")
```

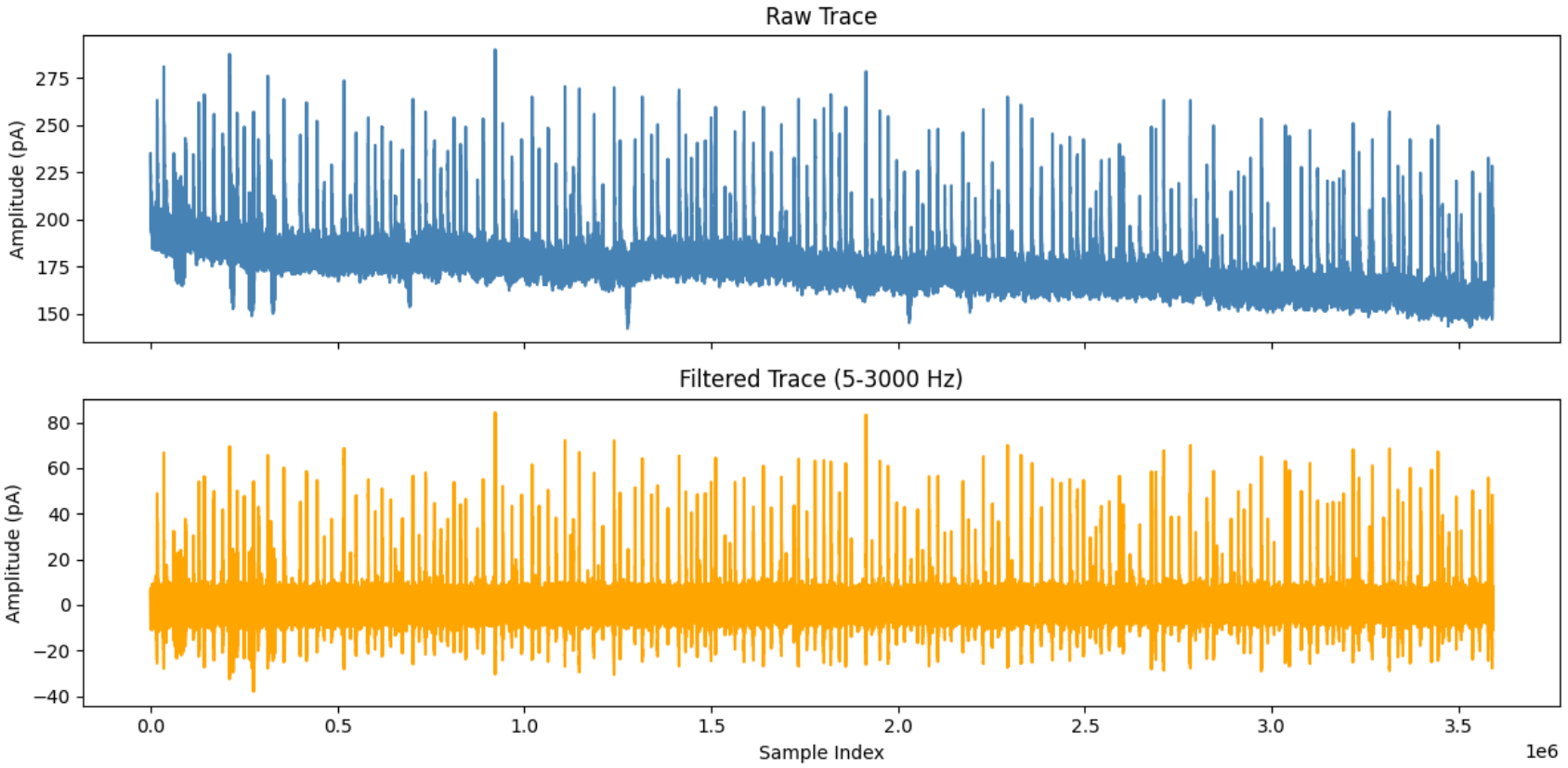
```
✅ Loaded data for 20 folders.
```

In [ ]:

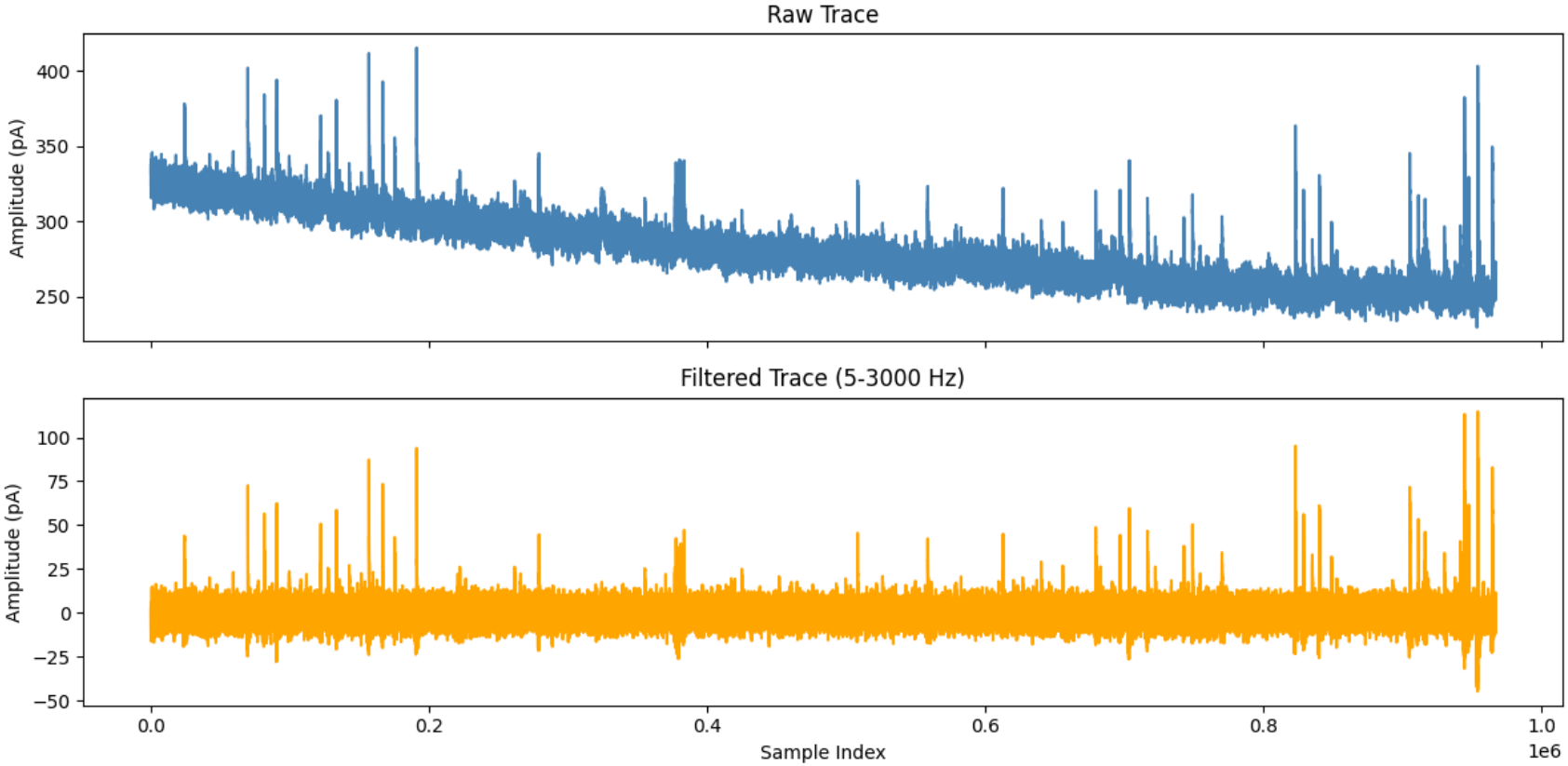
```
filtered_data_dict = {}
column_name = "time of peak"

for folder in data_dict:
  entry = data_dict[folder]
  df = entry["csv"]

  if not isinstance(df, pd.DataFrame):
    print(f"⚠️ {folder} already contains a Series or invalid format")
    continue

  # Check for exact match
  lower_cols = [col.lower() for col in df.columns]
  if column_name in lower_cols:
    actual_col = df.columns[lower_cols.index(column_name)]
    peak_times = df[actual_col]
```

```
    # Run the new filter function
    abf_pathway = entry["abf"].abfFilePath
    result = load_and_filter_abf(abf_pathway, plot=True)

    # Save results
    filtered_data_dict[folder] = {
        "filename": abf_pathway.split('/')[-1],
        "raw_trace": result["raw"],
        "filtered": result["filtered"],
        "ground_truth_peak_times": peak_times
    }
  else:
    print(f"⚠️ '{column_name}' not found in {folder}")
```

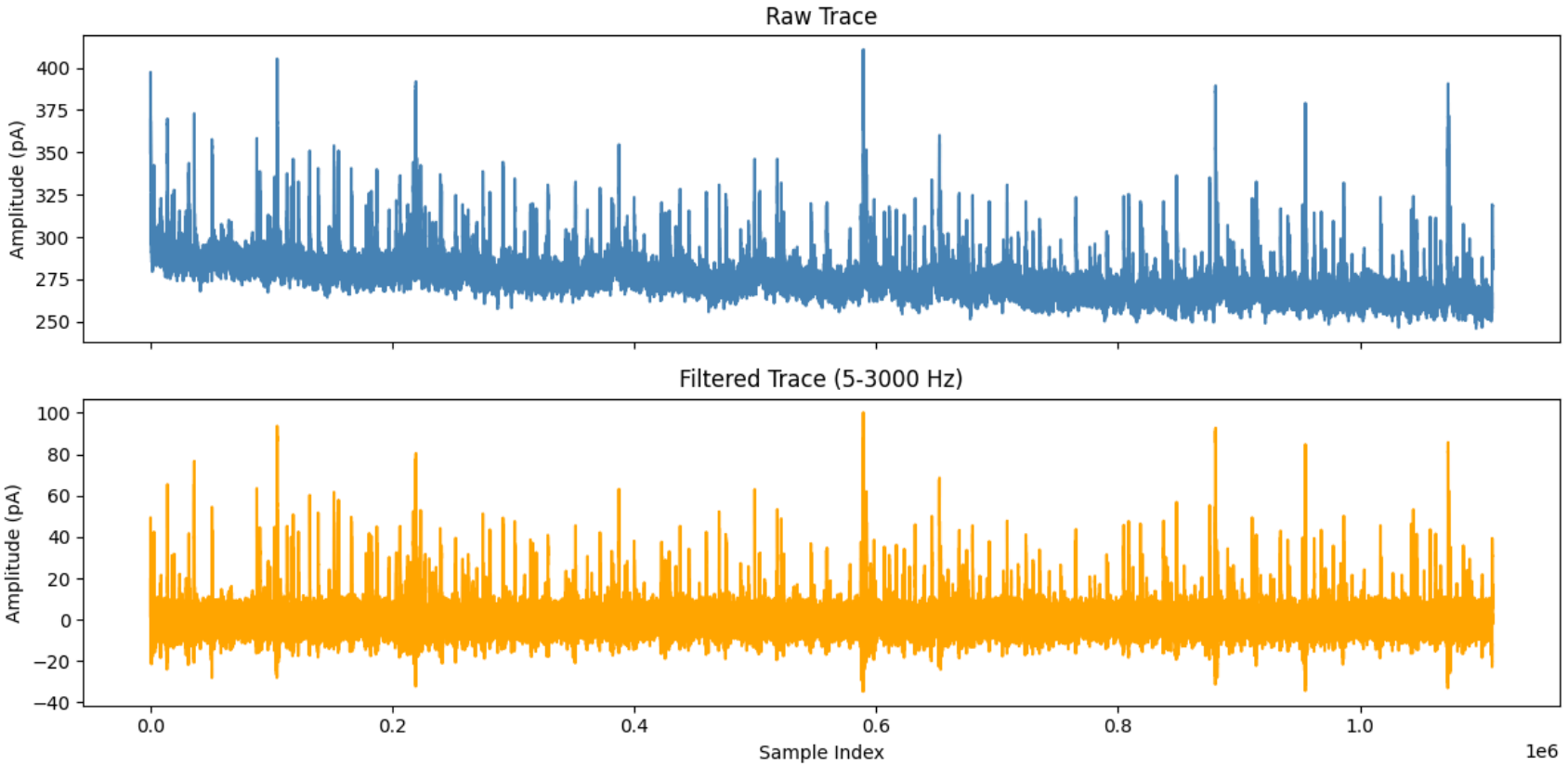
```
Data: (1, 3590940)
Y-Sweep: (3590940,)
```



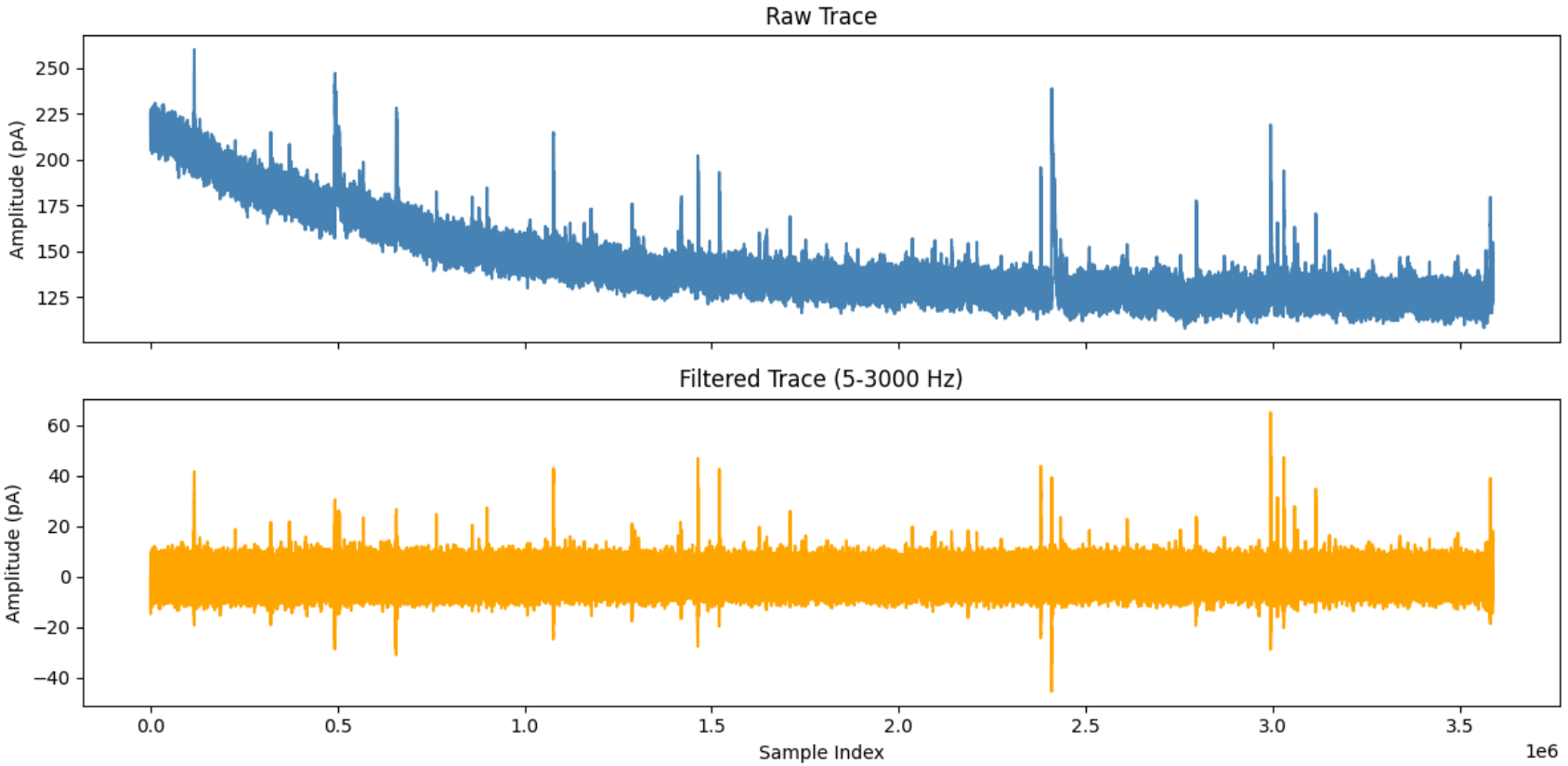
```
Data: (1, 967020)
Y-Sweep: (967020,)
```



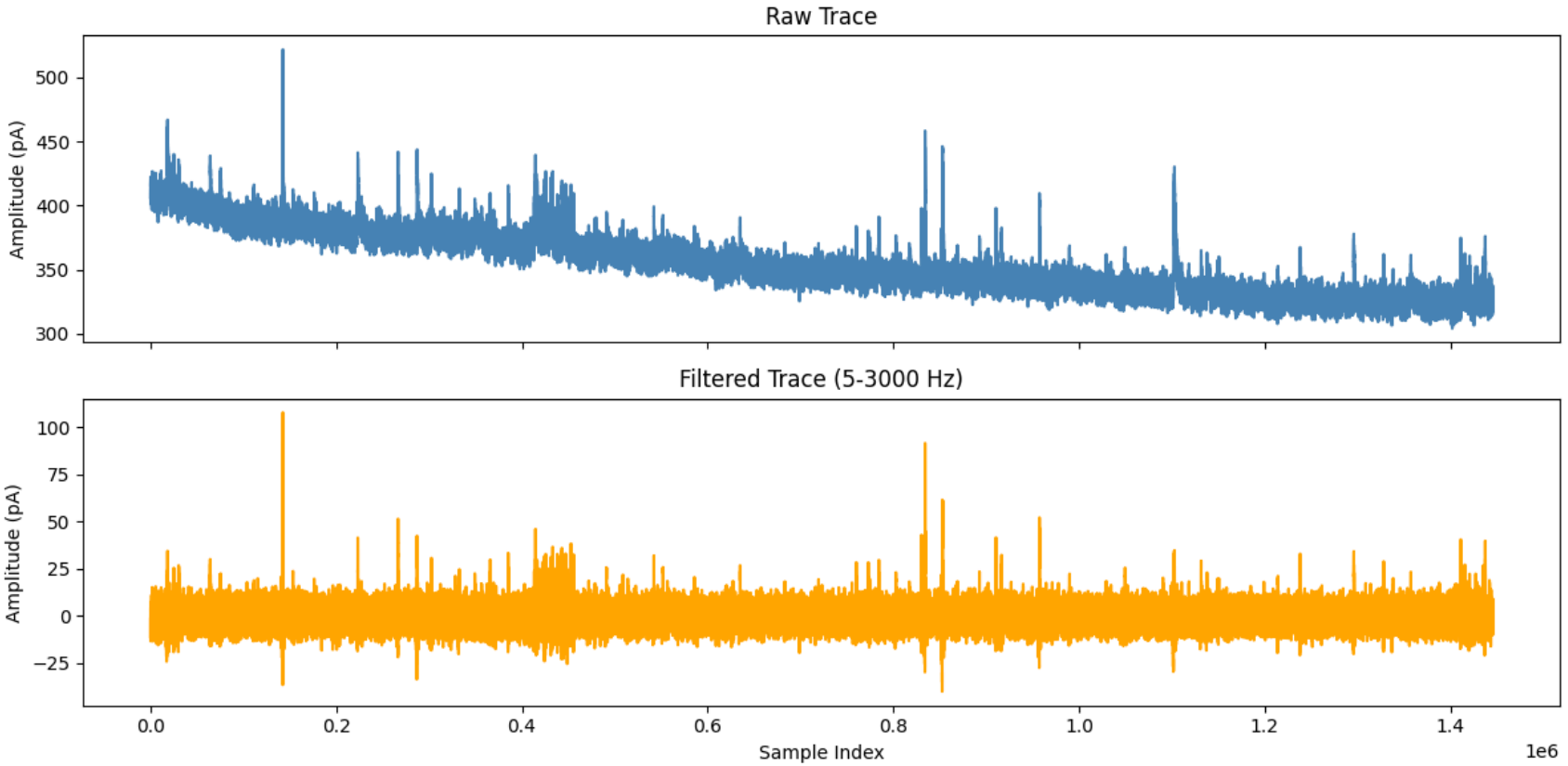

```
Data: (1, 1109680)
Y-Sweep: (1109680,)
```



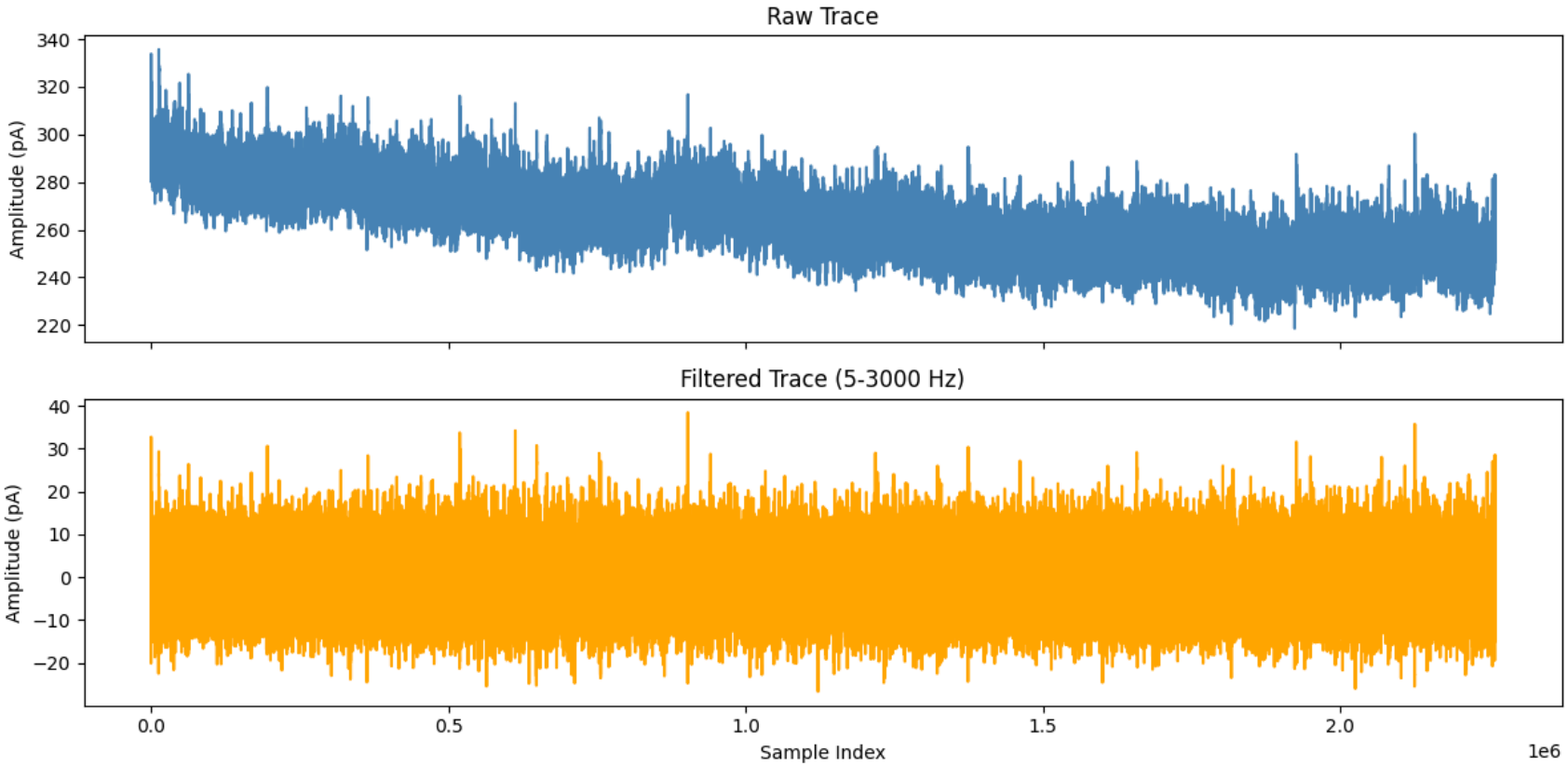

```
Data: (1, 3587640)
Y-Sweep: (3587640,)
```



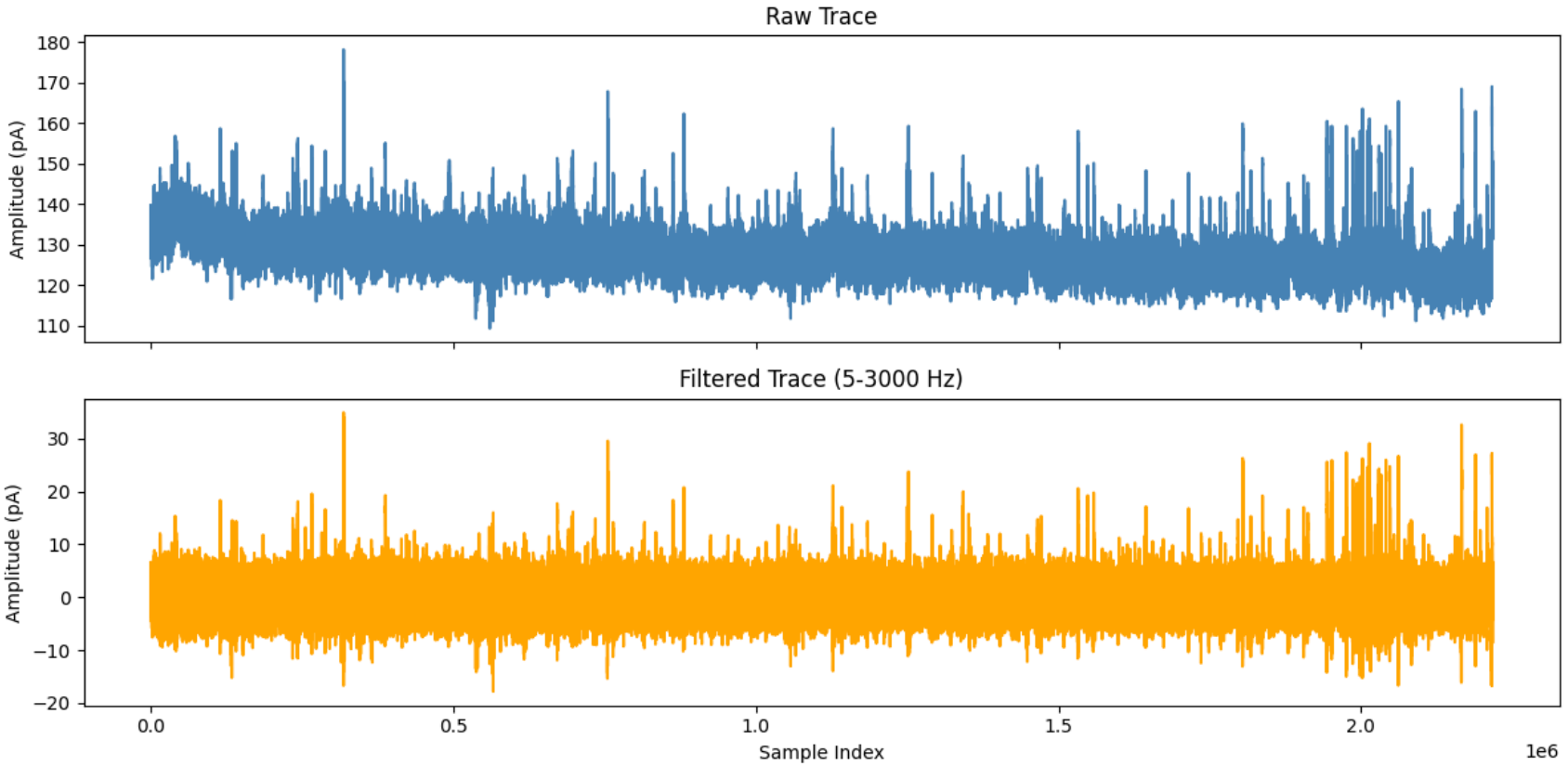

```
Data: (1, 1445560)
Y-Sweep: (1445560,)
```



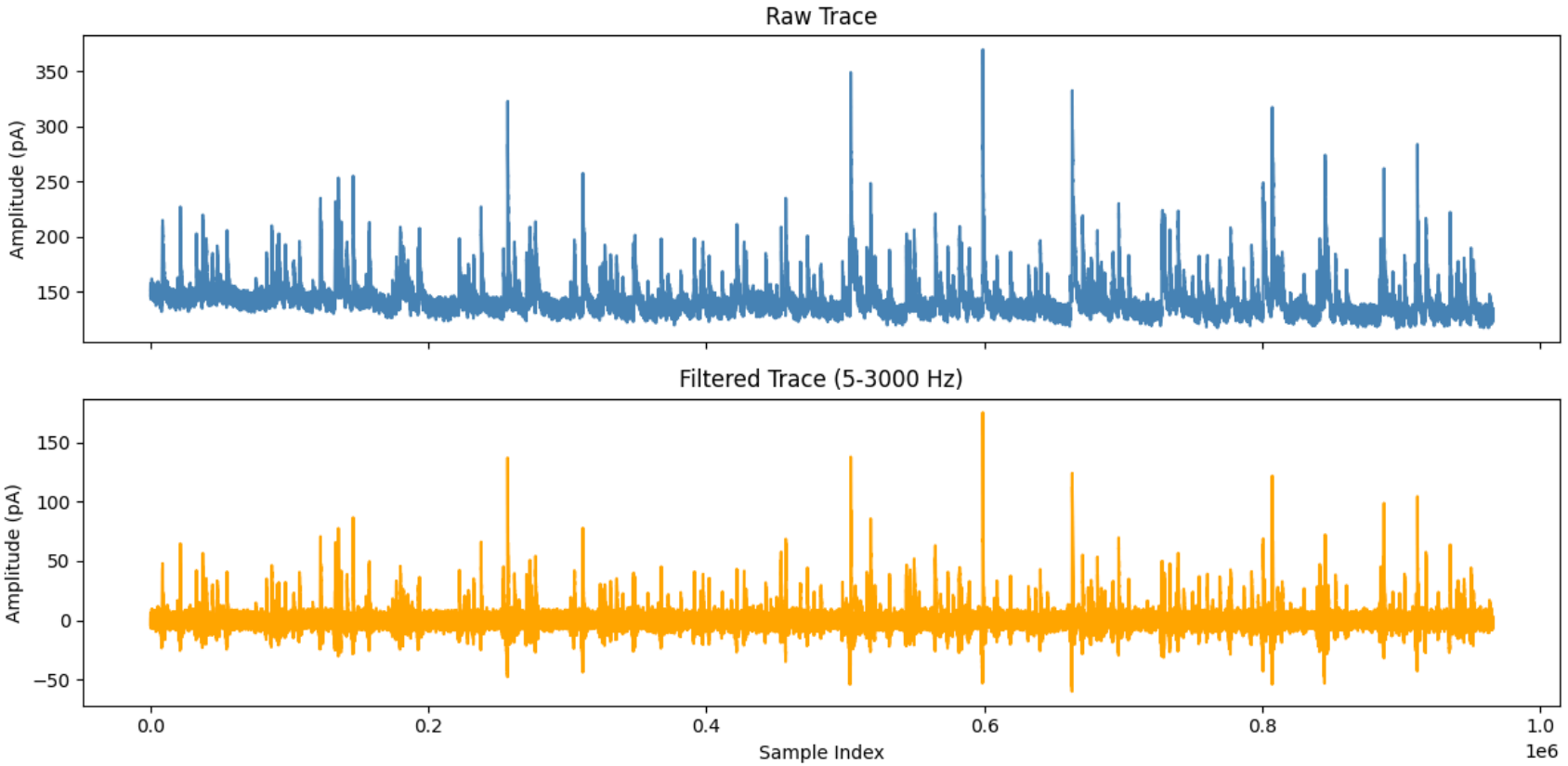

```
Data: (1, 2260220)
Y-Sweep: (2260220,)
```



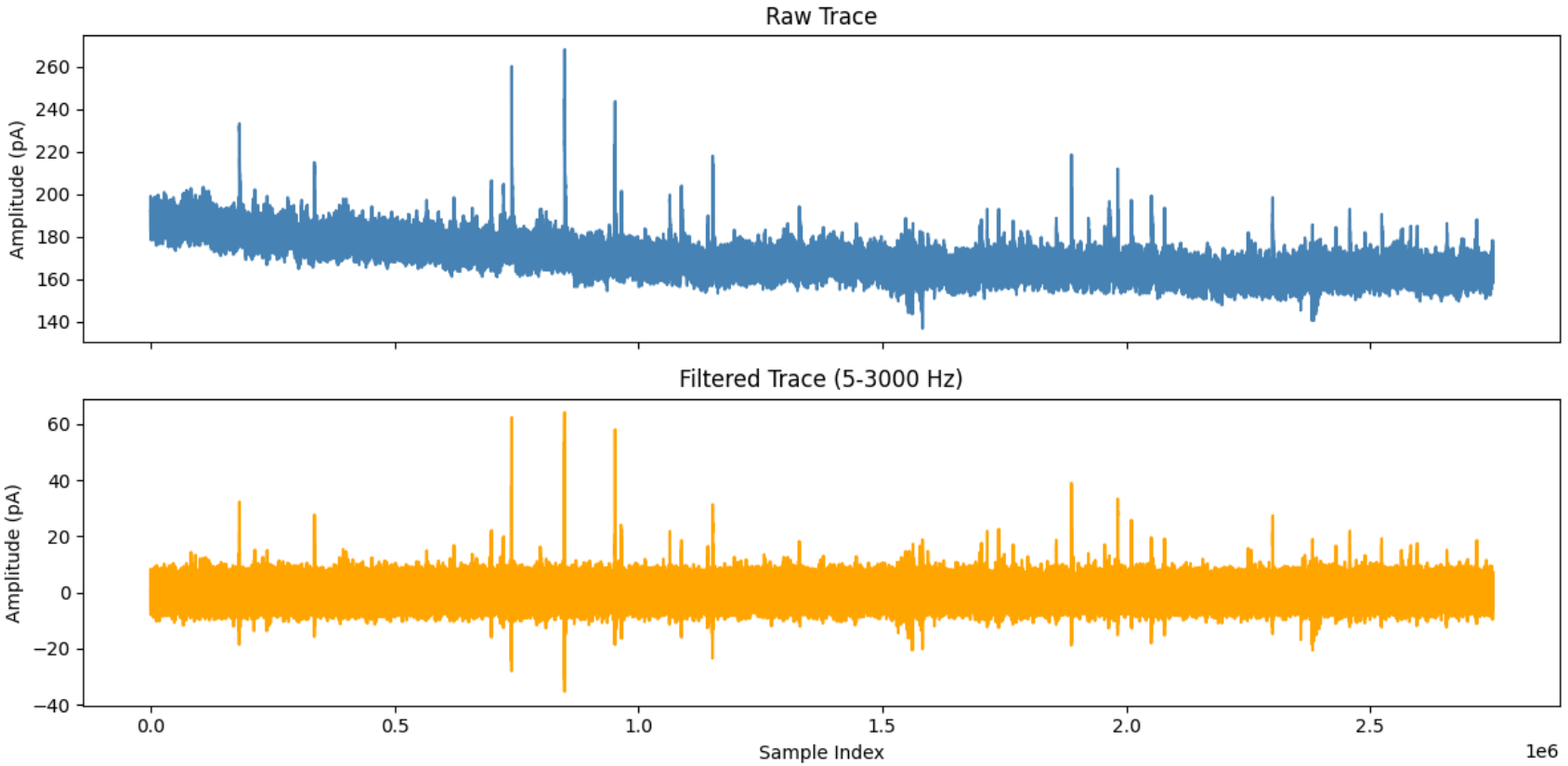

```
Data: (1, 2217500)
Y-Sweep: (2217500,)
```



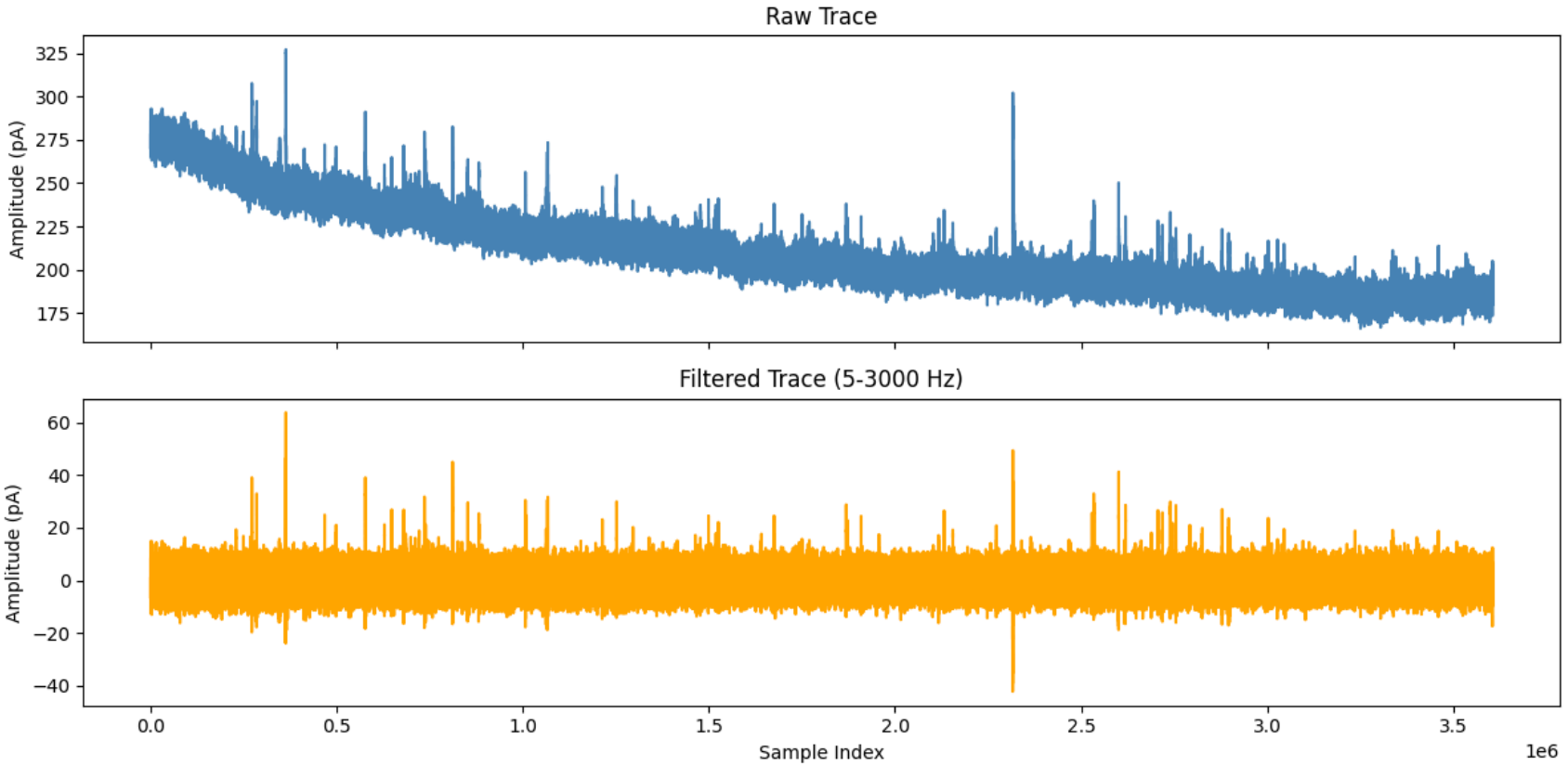

```
Data: (1, 965900)
Y-Sweep: (965900,)
```



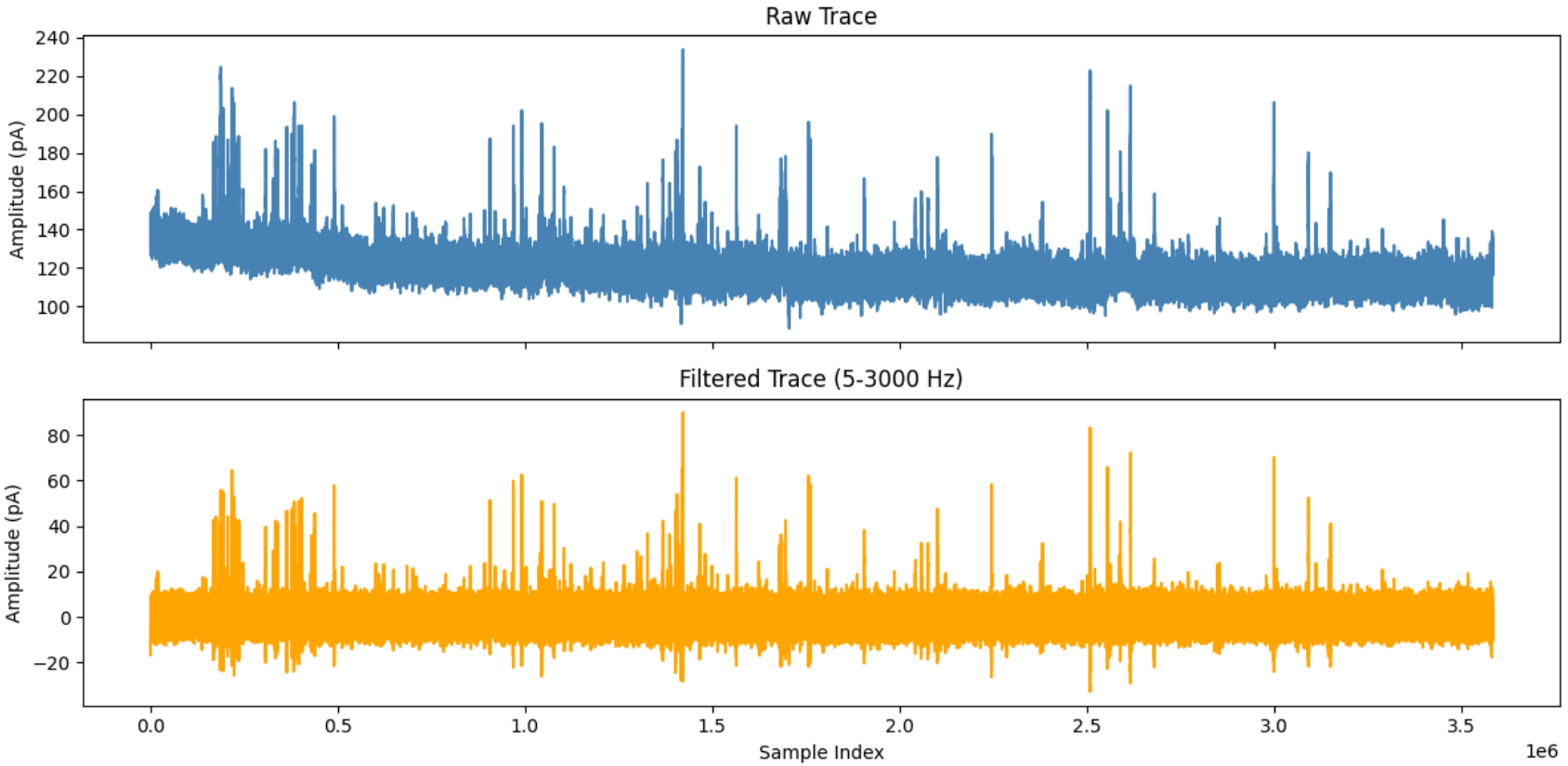

```
Data: (1, 2751320)
Y-Sweep: (2751320,)
```



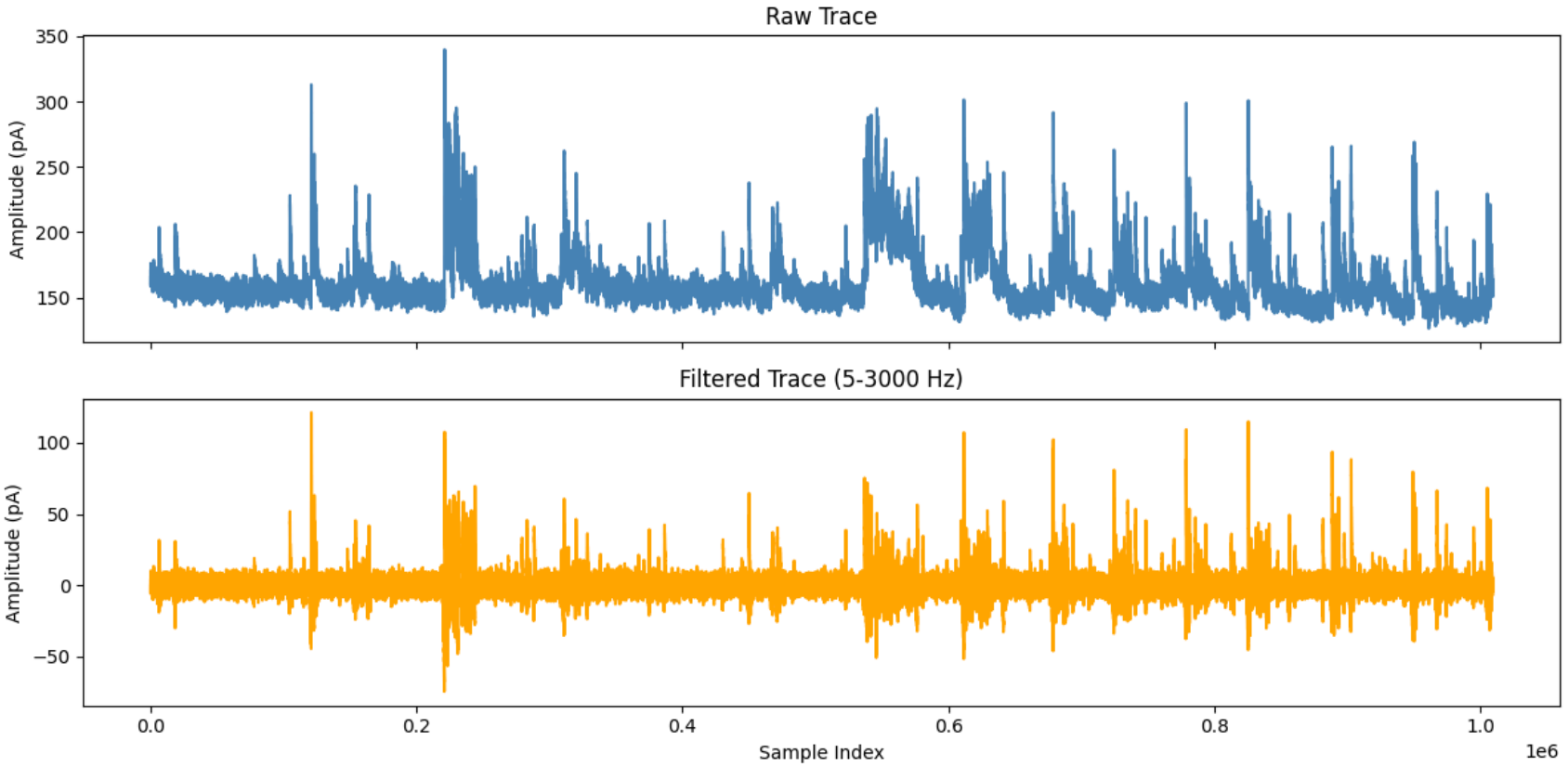

```
Data: (1, 3605680)
Y-Sweep: (3605680,)
```



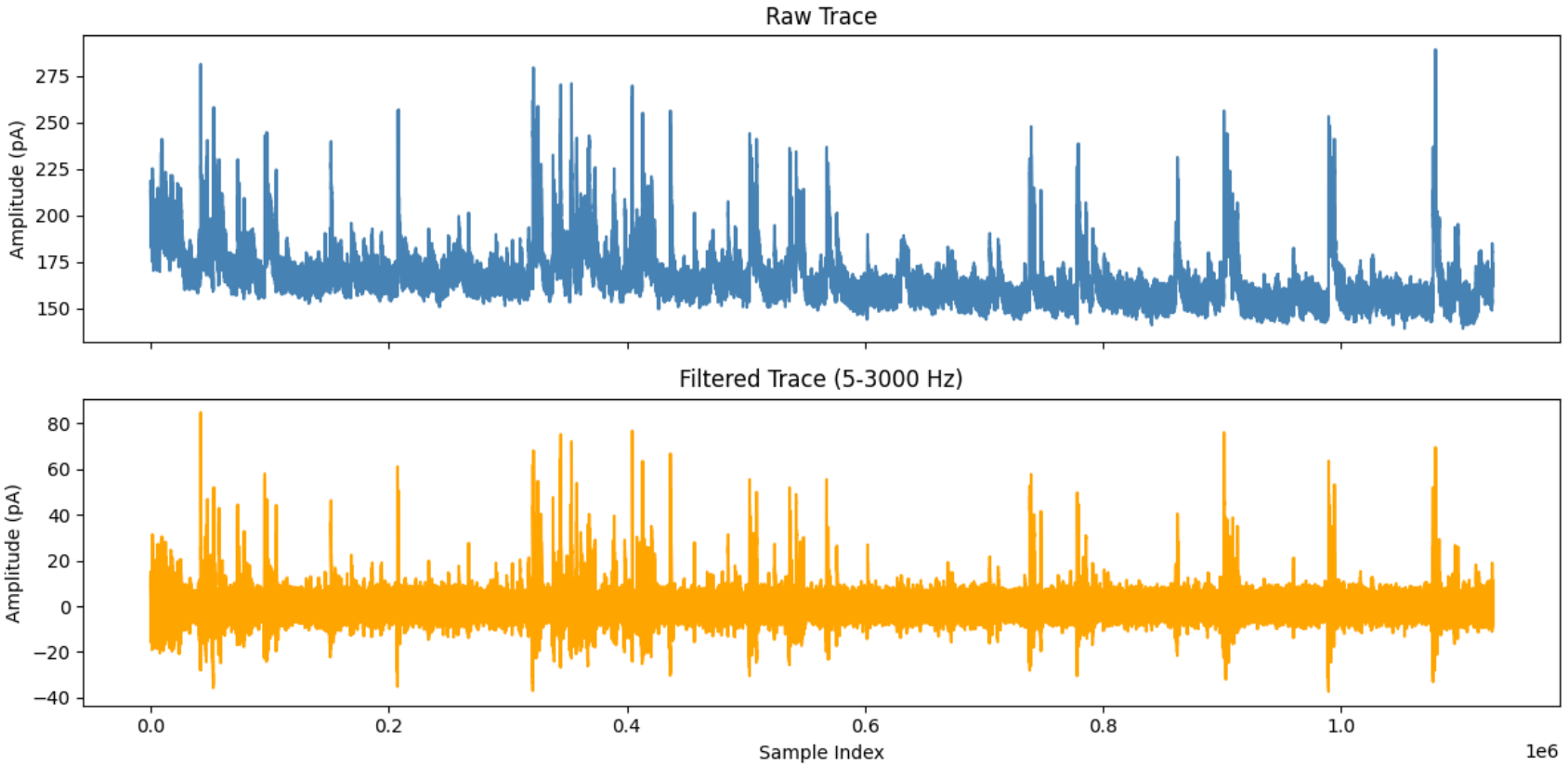

```
Data: (1, 3583620)
Y-Sweep: (3583620,)
```



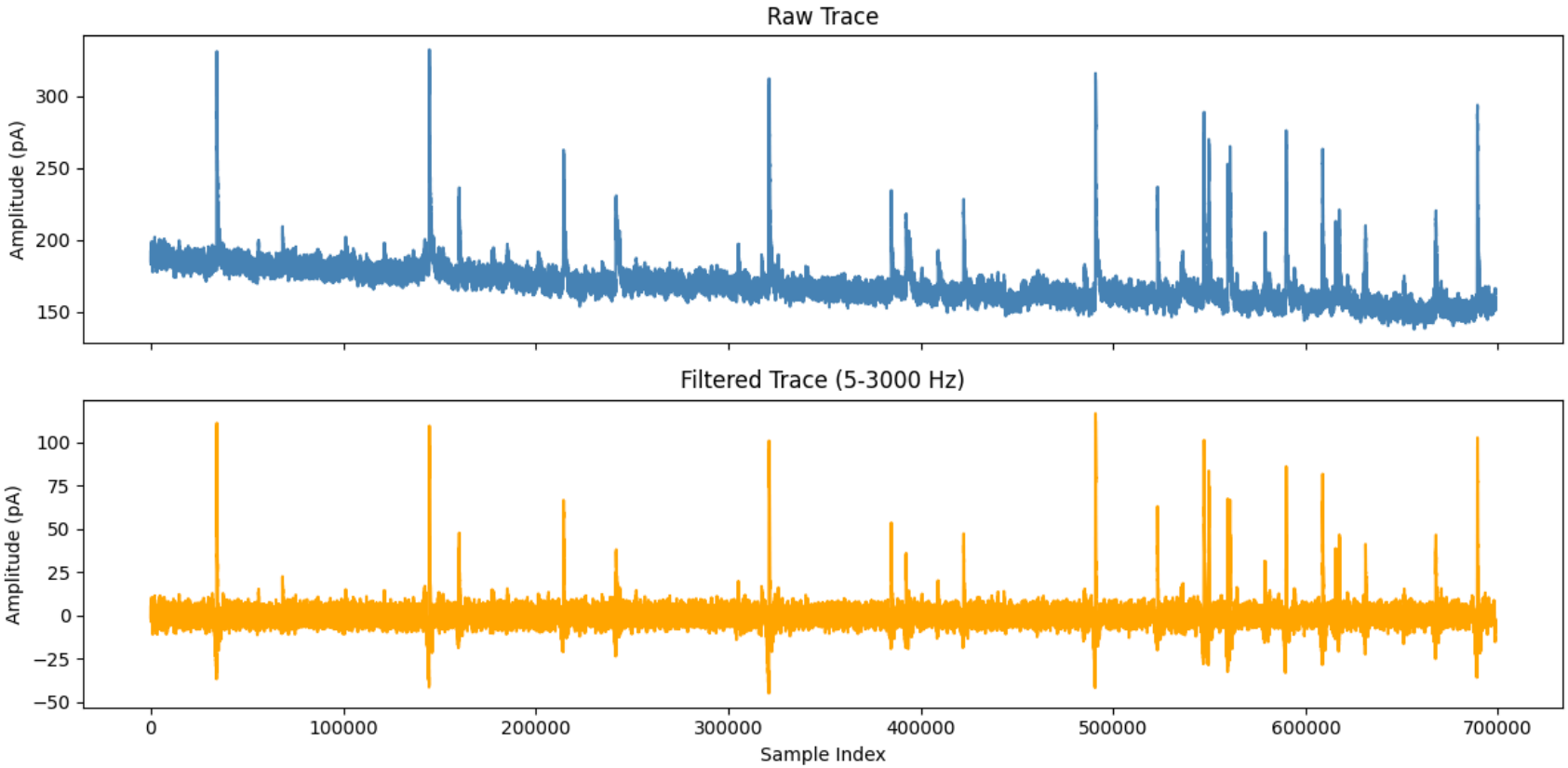

```
Data: (1, 1009240)
Y-Sweep: (1009240,)
```



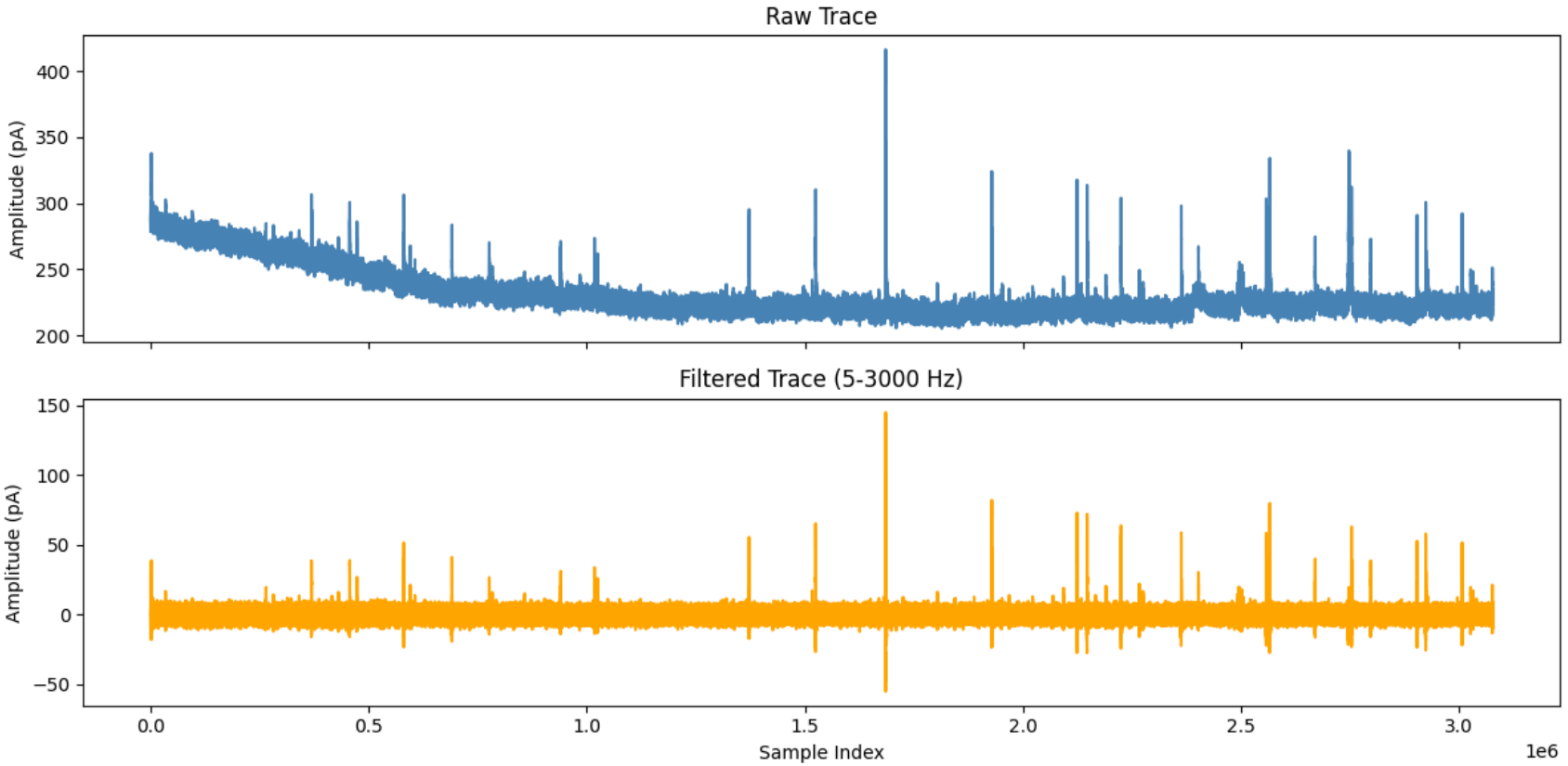

```
Data: (1, 1127240)
Y-Sweep: (1127240,)
```



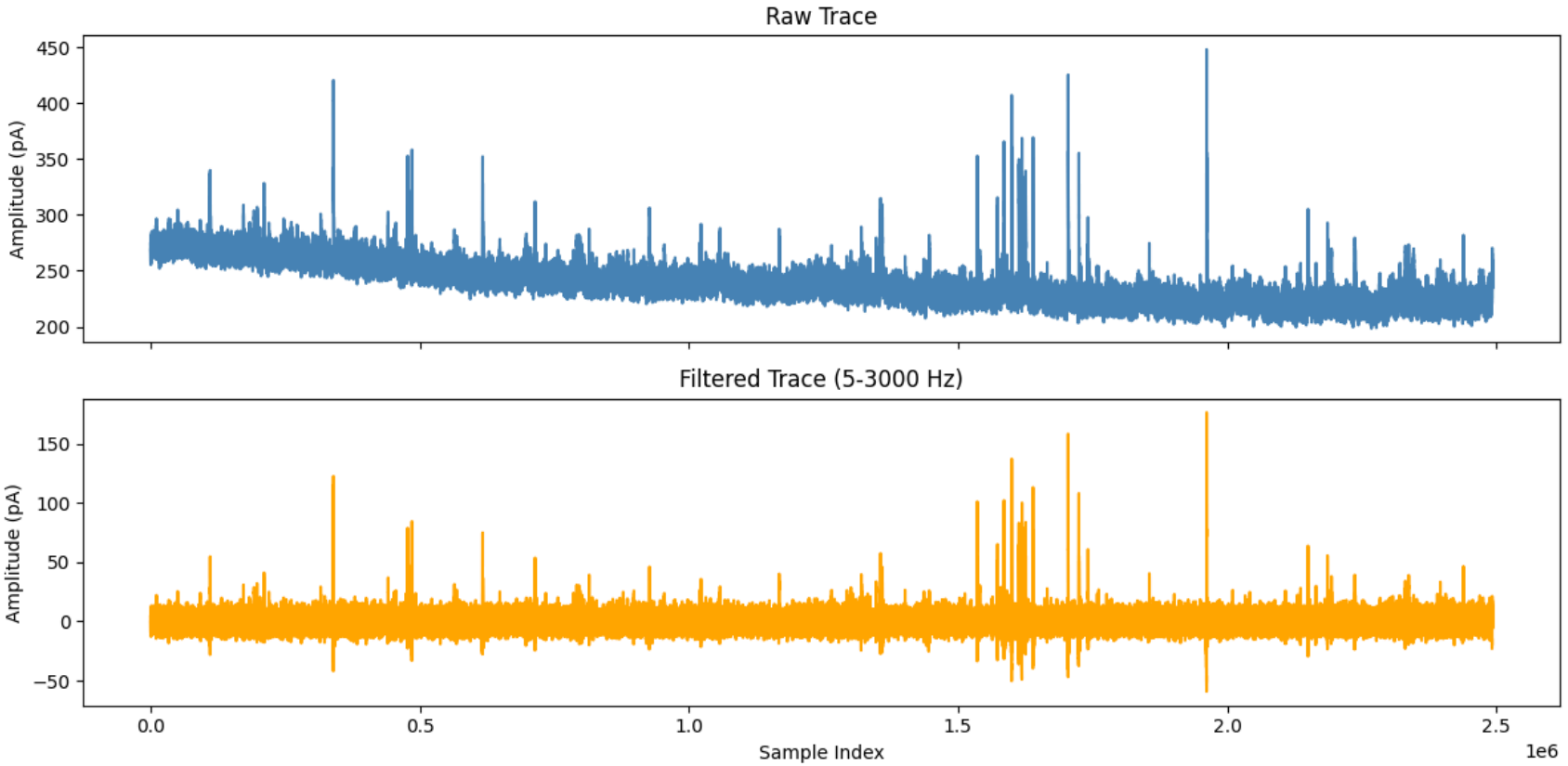

```
Data: (1, 698620)
Y-Sweep: (698620,)
```



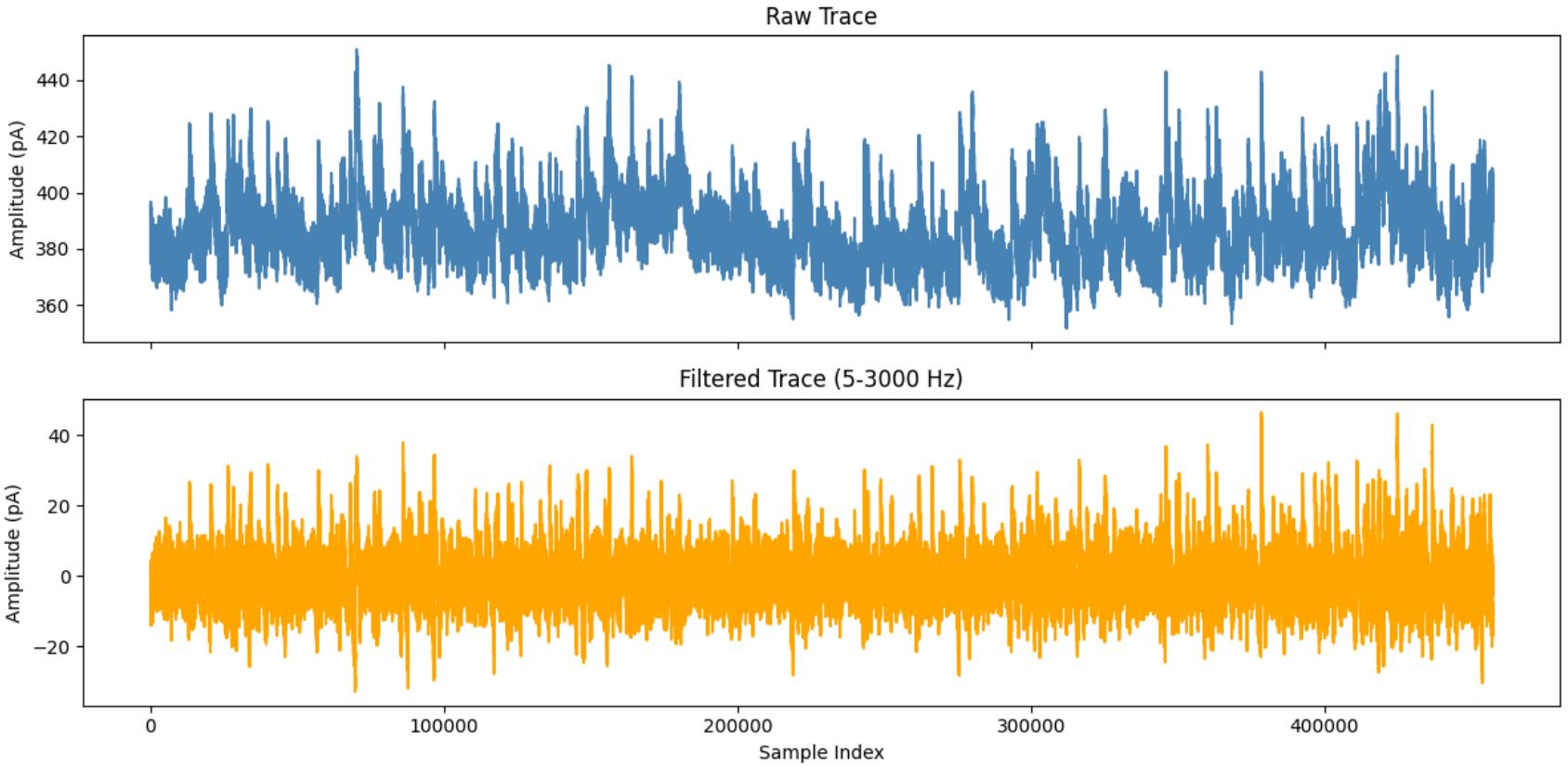

```
Data: (1, 3076800)
Y-Sweep: (3076800,)
```



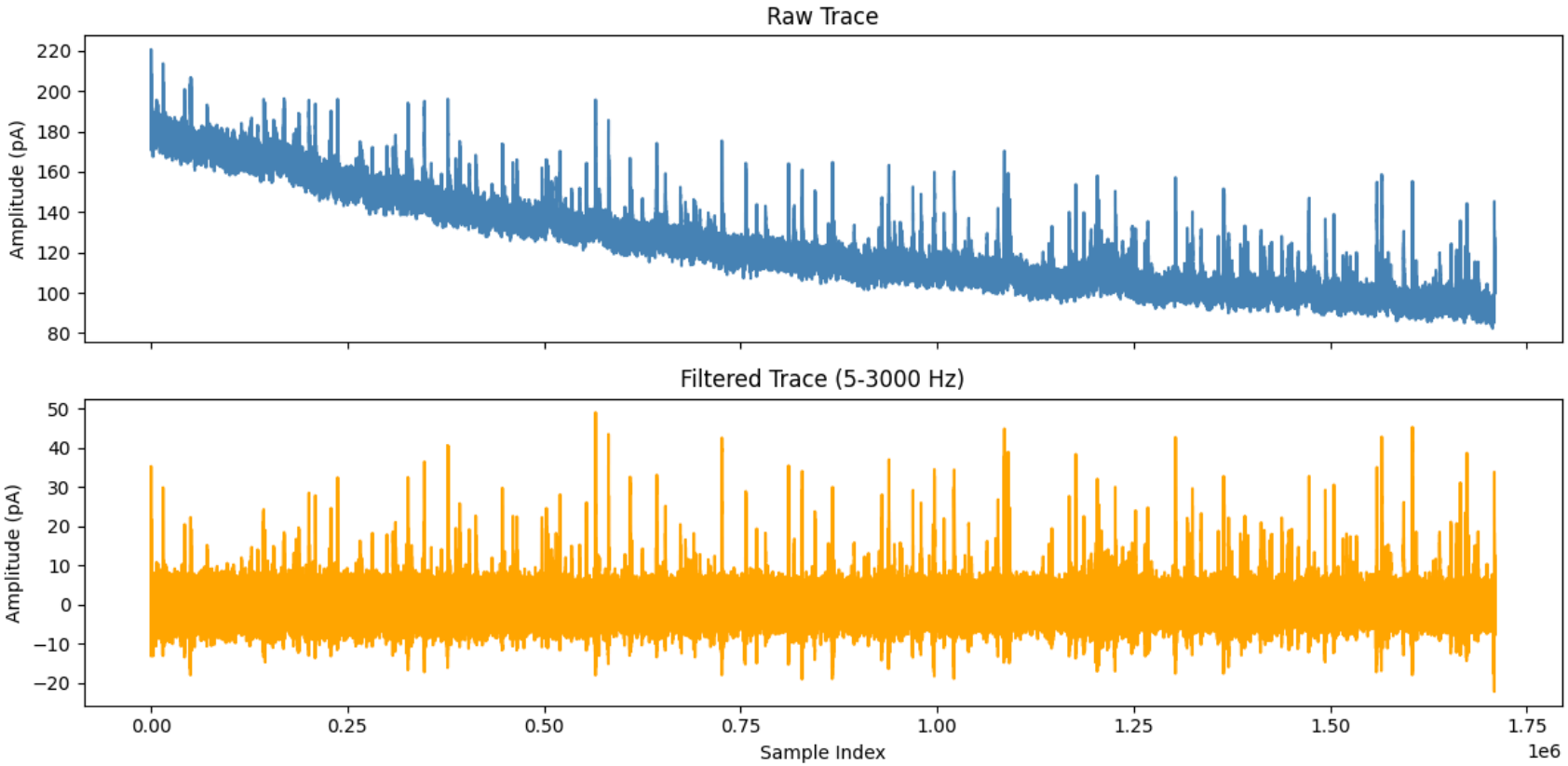

```
Data: (1, 2493120)
Y-Sweep: (2493120,)
```



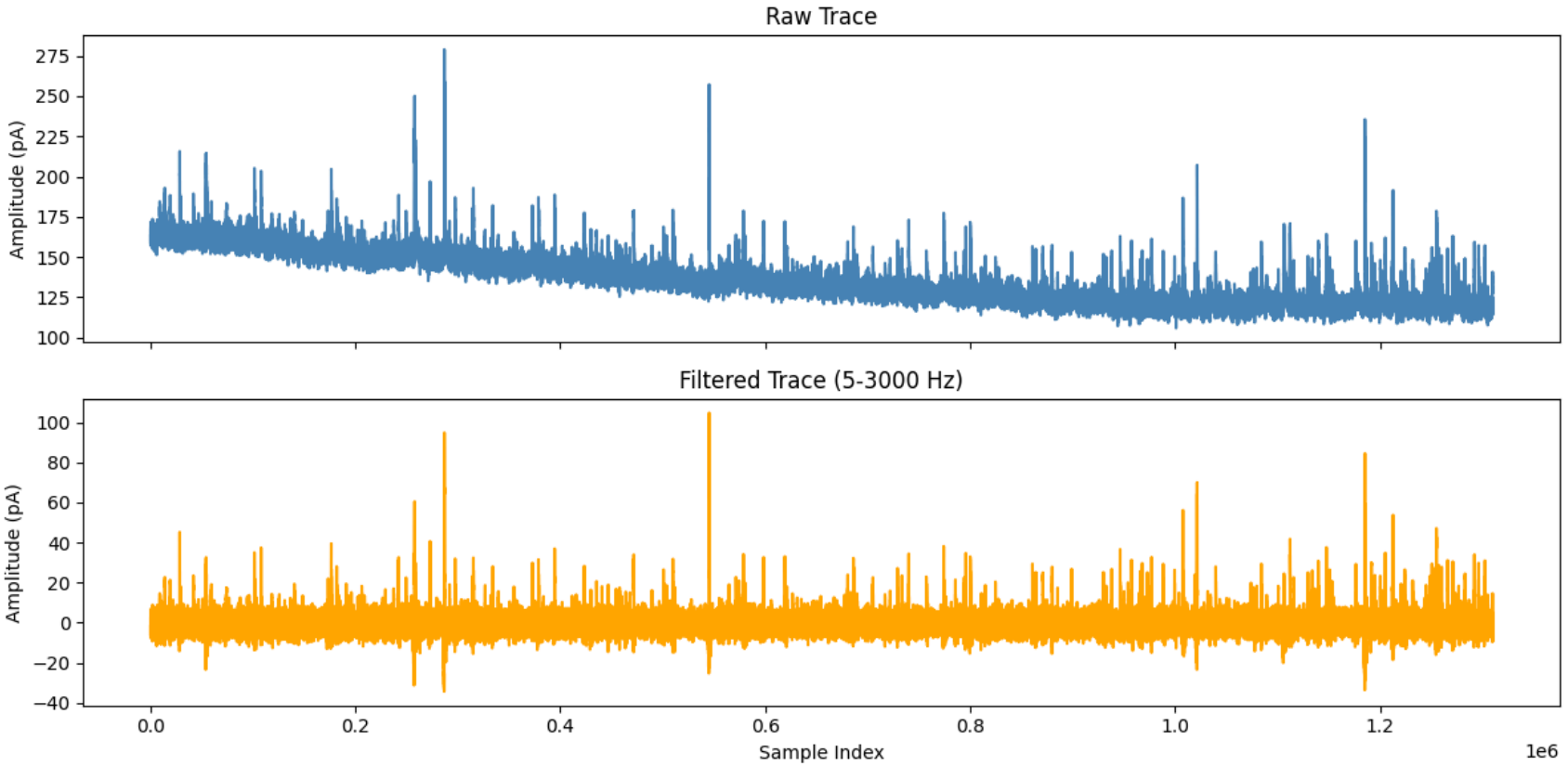

```
Data: (1, 457280)
Y-Sweep: (457280,)
```



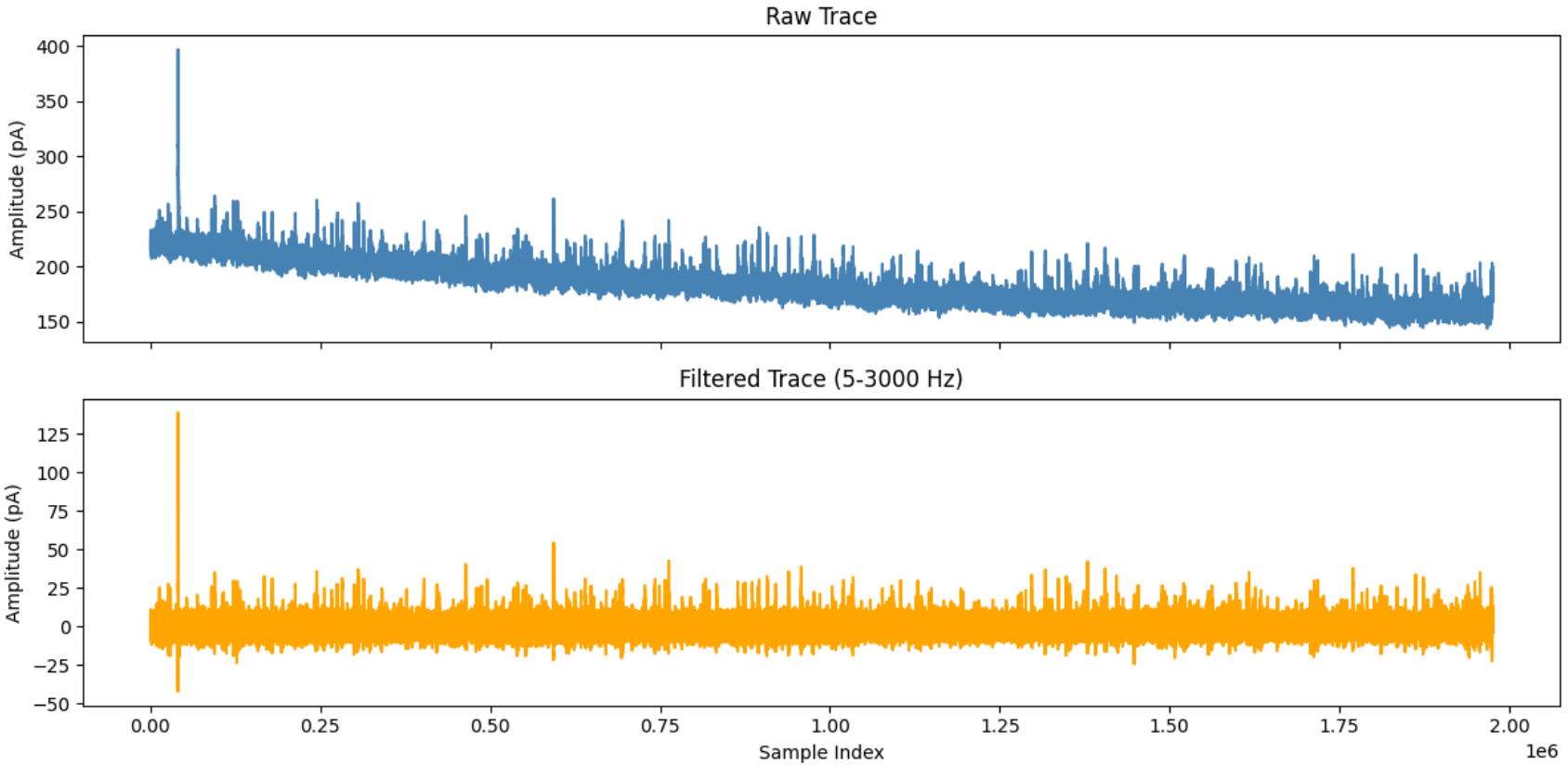

```
Data: (1, 1709660)
Y-Sweep: (1709660,)
```




```
Data: (1, 1310020)
Y-Sweep: (1310020,)
```




```
Data: (1, 1975580)
Y-Sweep: (1975580,)
```



In [ ]:
```
print("🔍 Checking filtered_data_dict contents...\n")

for folder, contents in filtered_data_dict.items():
    print(f"📁 Folder: {folder}")

    # Filename
    filename = contents.get("filename", None)
    print(f"    📄 File: {filename}")

    # Raw trace
    raw_trace = contents.get("raw_trace", None)
    print(f"    🔷 Raw Trace Length: {len(raw_trace) if raw_trace is not None else 'Missing'}")

    # Filtered trace
    filtered_trace = contents.get("filtered", None)
    print(f"    🔶 Filtered Trace Length: {len(filtered_trace) if filtered_trace is not None else 'Missing'}")
```

```
    # Ground truth peak times
    peak_times = contents.get("ground_truth_peak_times", None)
    print(f"    📍 GT Peak Times: {peak_times.shape[0] if peak_times is not None else 'Missing'} events")

    print("-" * 50)
```

```
🔍 Checking filtered_data_dict contents...

📁 Folder: T1 Set 1
   📄 File: 2024_03_06_0004_Trunc_ Cell 1 IPSC.abf
   🔷 Raw Trace Length: 3590940
   🔶 Filtered Trace Length: 3590940
   📍 GT Peak Times: 278 events
--------------------------------------------------
📁 Folder: T1 Set 2
   📄 File: 2024_03_07_0003_Trunc_ Cell 1 IPSC.abf
   🔷 Raw Trace Length: 967020
   🔶 Filtered Trace Length: 967020
   📍 GT Peak Times: 300 events
--------------------------------------------------
📁 Folder: T1 Set 3
   📄 File: 2024_03_08_0008_Trunc_ Cell 2 IPSC.abf
   🔷 Raw Trace Length: 1109680
   🔶 Filtered Trace Length: 1109680
   📍 GT Peak Times: 304 events
--------------------------------------------------
📁 Folder: T1 Set 4
   📄 File: 2024_01_25_0008_Trunc_ Cell 2 IPSC.abf
   🔷 Raw Trace Length: 3587640
   🔶 Filtered Trace Length: 3587640
   📍 GT Peak Times: 149 events
--------------------------------------------------
📁 Folder: T1 Set 5
   📄 File: 2024_03_07_0008_Trunc_ Cell 2 IPSC.abf
   🔷 Raw Trace Length: 1445560
   🔶 Filtered Trace Length: 1445560
   📍 GT Peak Times: 303 events
--------------------------------------------------
📁 Folder: T1 Set 6
   📄 File: 2024_03_08_0003_Trunc_ Cell 1 IPSC.abf
   🔷 Raw Trace Length: 2260220
   🔶 Filtered Trace Length: 2260220
   📍 GT Peak Times: 300 events
--------------------------------------------------
📁 Folder: T1 Set 7
   📄 File: 2024_03_14_0010_Trunc_ Cell 1 IPSC.abf
   🔷 Raw Trace Length: 2217500
   🔶 Filtered Trace Length: 2217500
```

📍 GT Peak Times: 301 events

📁 Folder: T1 Set 8
📄 File: 2024_03_21_0018_Trunc_ Cell 3 IPSC.abf
🔷 Raw Trace Length: 965900
🔶 Filtered Trace Length: 965900
📍 GT Peak Times: 300 events

📁 Folder: T1 Set 9
📄 File: 2024_03_22_0004_Trunc_ Cell 1 IPSC.abf
🔷 Raw Trace Length: 2751320
🔶 Filtered Trace Length: 2751320
📍 GT Peak Times: 300 events

📁 Folder: T1 Set 10
📄 File: 2024_03_22_0008_Trunc_ Cell 2 IPSC.abf
🔷 Raw Trace Length: 3605680
🔶 Filtered Trace Length: 3605680
📍 GT Peak Times: 251 events

📁 Folder: T1 Set 11
📄 File: 2024_03_19_0010_Trunc_ Cell 2 IPSC.abf
🔷 Raw Trace Length: 3583620
🔶 Filtered Trace Length: 3583620
📍 GT Peak Times: 281 events

📁 Folder: T1 Set 12
📄 File: 2024_03_21_0006_Trunc_ Cell 1 IPSC.abf
🔷 Raw Trace Length: 1009240
🔶 Filtered Trace Length: 1009240
📍 GT Peak Times: 300 events

📁 Folder: T1 Set 13
📄 File: 2024_03_21_0012_Trunc_ Cell 2 IPSC.abf
🔷 Raw Trace Length: 1127240
🔶 Filtered Trace Length: 1127240
📍 GT Peak Times: 300 events

📁 Folder: T1 Set 14
📄 File: 2024_03_14_0016_Trunc_ Cell 2 IPSC.abf
🔷 Raw Trace Length: 698620
🔶 Filtered Trace Length: 698620

```
    📍 GT Peak Times: 301 events
--------------------------------------------------
📁 Folder: T1 Set 15
    📄 File: 2024_02_08_0008_Trunc_ Cell 2 IPSC.abf
    🔷 Raw Trace Length: 3076800
    🔶 Filtered Trace Length: 3076800
    📍 GT Peak Times: 300 events
--------------------------------------------------
📁 Folder: T1 Set 16
    📄 File: 2024_03_19_0004_Trunc_ Cell 1 IPSC.abf
    🔷 Raw Trace Length: 2493120
    🔶 Filtered Trace Length: 2493120
    📍 GT Peak Times: 300 events
--------------------------------------------------
📁 Folder: T1 Set 17
    📄 File: 2024_01_23_0008_Trunc_Cell 2 IPSC.abf
    🔷 Raw Trace Length: 457280
    🔶 Filtered Trace Length: 457280
    📍 GT Peak Times: 300 events
--------------------------------------------------
📁 Folder: T1 Set 18
    📄 File: 2024_02_08_Trunc_ Cell 1 IPSC.abf
    🔷 Raw Trace Length: 1709660
    🔶 Filtered Trace Length: 1709660
    📍 GT Peak Times: 300 events
--------------------------------------------------
📁 Folder: T1 Set 19
    📄 File: 2024_01_24_0002_Trunc_ Cell 1 IPSC.abf
    🔷 Raw Trace Length: 1310020
    🔶 Filtered Trace Length: 1310020
    📍 GT Peak Times: 303 events
--------------------------------------------------
📁 Folder: T1 Set 20
    📄 File: 2024_01_22_0009_Trunc_ Cell 1 IPSC.abf
    🔷 Raw Trace Length: 1975580
    🔶 Filtered Trace Length: 1975580
    📍 GT Peak Times: 302 events
--------------------------------------------------
```

In [ ]:
```python
def extract_waveform_windows(signal, peak_indices, pre_samples=40, post_samples=80):
    """
    Extract waveform windows around each detected peak.
```

```
    """
    waveforms = []
    for peak in peak_indices:
        if peak - pre_samples >= 0 and peak + post_samples < len(signal):
            waveforms.append(signal[peak - pre_samples : peak + post_samples])
    waveforms = np.array(waveforms)
    print(f"✅ [Step 1a: extract_waveform_windows] Extracted {waveforms.shape[0]} waveforms | Window size: {waveform
    return waveforms

# === Extraction Loop ===
windowed_waveforms_dict = {}
pre_samples = 40
post_samples = 60  # ✅ was typo: post_execute → post_samples

for folder, content in filtered_data_dict.items():
    filtered = content["filtered"]
    peak_times = content["ground_truth_peak_times"]

    waveforms = extract_waveform_windows(filtered, peak_indices, pre_samples, post_samples)

    windowed_waveforms_dict[folder] = {
        "waveforms": waveforms,
        "peak_indices": peak_indices,
        "filename": content["filename"]
    }

print(f"✅ Extracted waveform windows for {len(windowed_waveforms_dict)} files.")
```

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ [Step 1a: extract_waveform_windows] Extracted 304 waveforms | Window size: 100 samples

✅ Extracted waveform windows for 20 files.

In [ ]:

```
# ------------------------------------
# 1b. Normalize Waveforms
# ------------------------------------
def normalize_waveforms(waveforms):
    """
    Normalize each waveform to unit peak amplitude.
    """
    normalized = np.array([
        w / np.max(np.abs(w)) if np.max(np.abs(w)) != 0 else w for w in waveforms
    ])
    print(f"✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: {normalized.shape}\n")
    return normalized

# ------------------------------------
# 2. Wavelet Decomposition
# ------------------------------------
def wavelet_decompose_waveforms(waveforms, wavelet='db4', level=2):
    """
    Decompose each waveform using DWT and flatten coefficients.
    """
    coeff_matrix = []
    for i, waveform in enumerate(waveforms):
        coeffs = pywt.wavedec(waveform, wavelet=wavelet, level=level)
        coeff_vector = np.concatenate(coeffs)
        coeff_matrix.append(coeff_vector)
        if i == 0:
            level_shapes = ", ".join([f"Level {j}: {c.shape}" for j, c in enumerate(coeffs)])
            print(f"🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into {len(coeffs)} levels:")
            print(f"    ↪ {level_shapes} | Total coeff length: {len(coeff_vector)}")
    coeff_matrix = np.array(coeff_matrix)
    print(f"✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: {coeff_matrix.shape}\n")
    return coeff_matrix


####
wavelet_features_dict = {}

for folder, entry in windowed_waveforms_dict.items():
    waveforms = entry["waveforms"]

    # Step 1b: Normalize
```

```
    normalized = normalize_waveforms(waveforms)

    # Step 2: Wavelet decomposition
    coeffs = wavelet_decompose_waveforms(normalized, wavelet='db4', level=2)

    wavelet_features_dict[folder] = {
        "waveforms": waveforms,
        "normalized": normalized,
        "wavelet_features": coeffs,
        "filename": entry["filename"]
    }
```

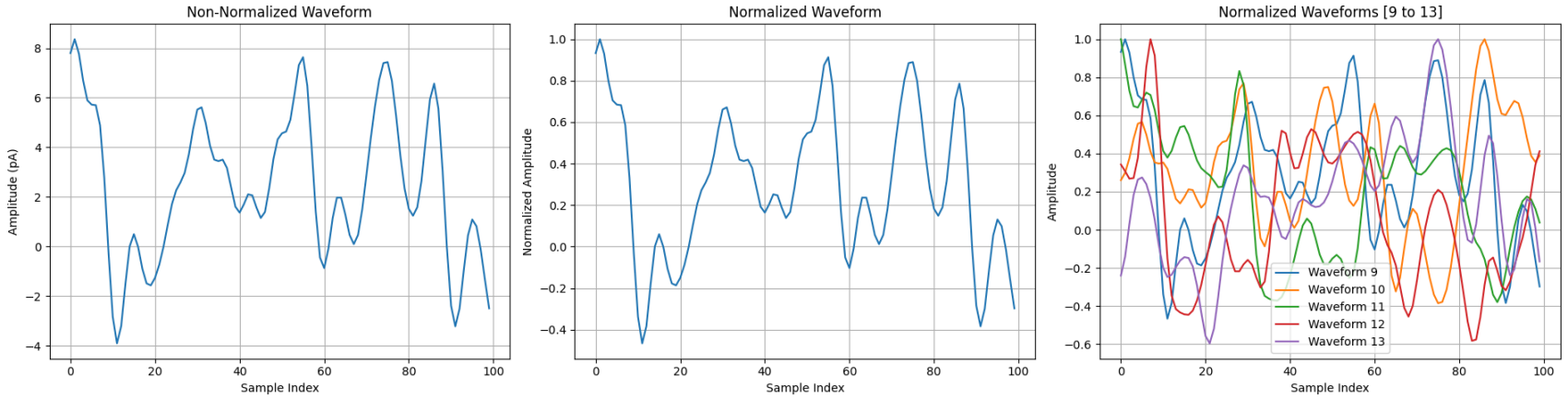
```
✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)
```

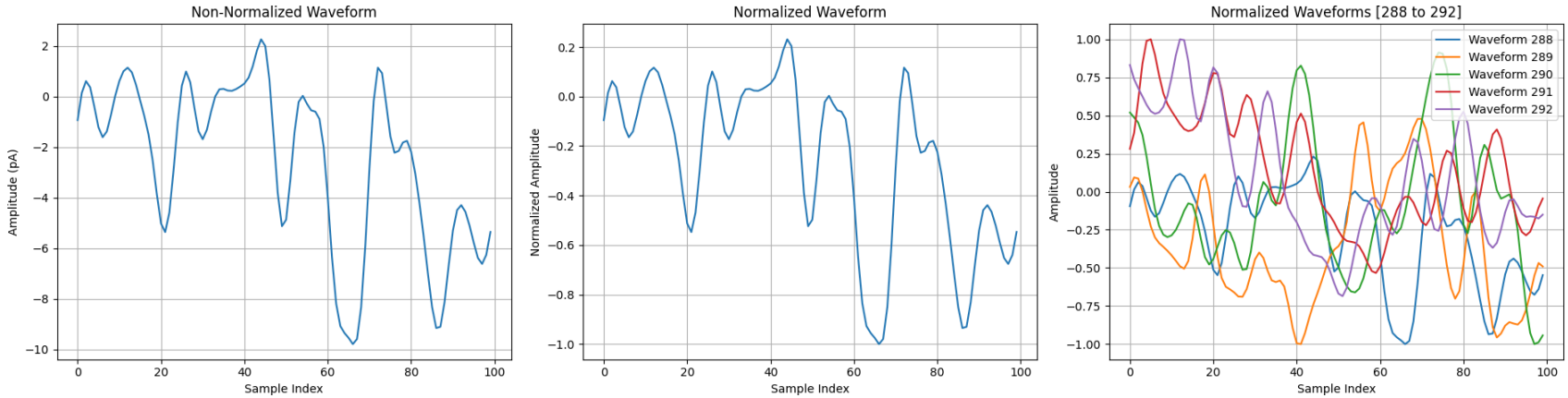
```
✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)
```

```
✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (304, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 3 levels:
   ↪ Level 0: (30,), Level 1: (30,), Level 2: (53,) | Total coeff length: 113
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (304, 113)
```

In [ ]:
```python
folder = "T1 Set 2"
entry = wavelet_features_dict[folder]
waveforms = entry["waveforms"]
normalized_waveforms = entry["normalized"]
```

```
plot_waveform_inspection(
    waveforms=waveforms,
    normalized_waveforms=normalized_waveforms,
    index_wave=9 ,     # pick any valid index
    n_examples=5      # how many to group-plot in subplot 3
)


idx = np.random.randint(0, len(waveforms))

plot_waveform_inspection(
    waveforms=waveforms,
    normalized_waveforms=normalized_waveforms,
    index_wave=idx,
    n_examples=5
)
```

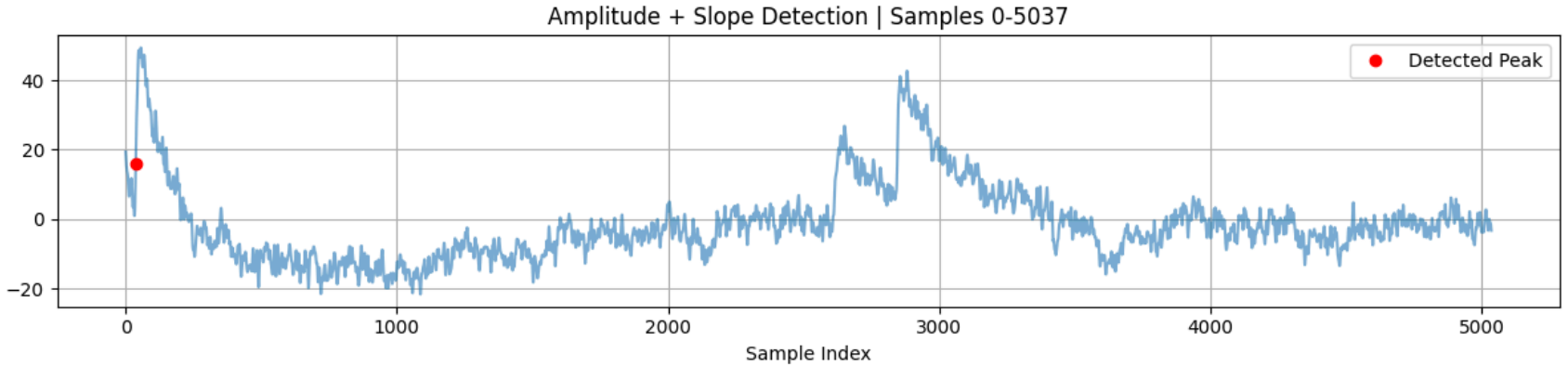


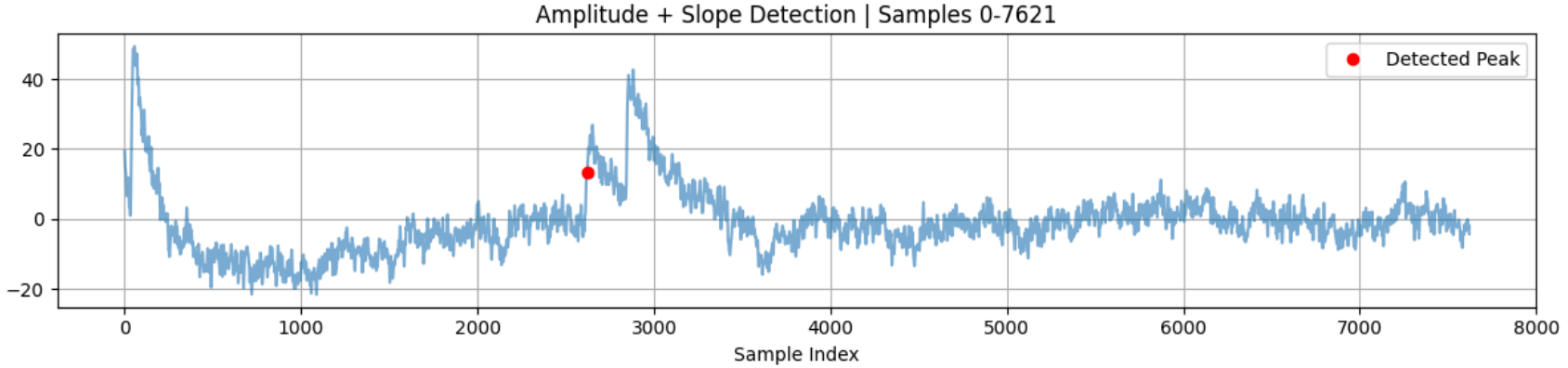


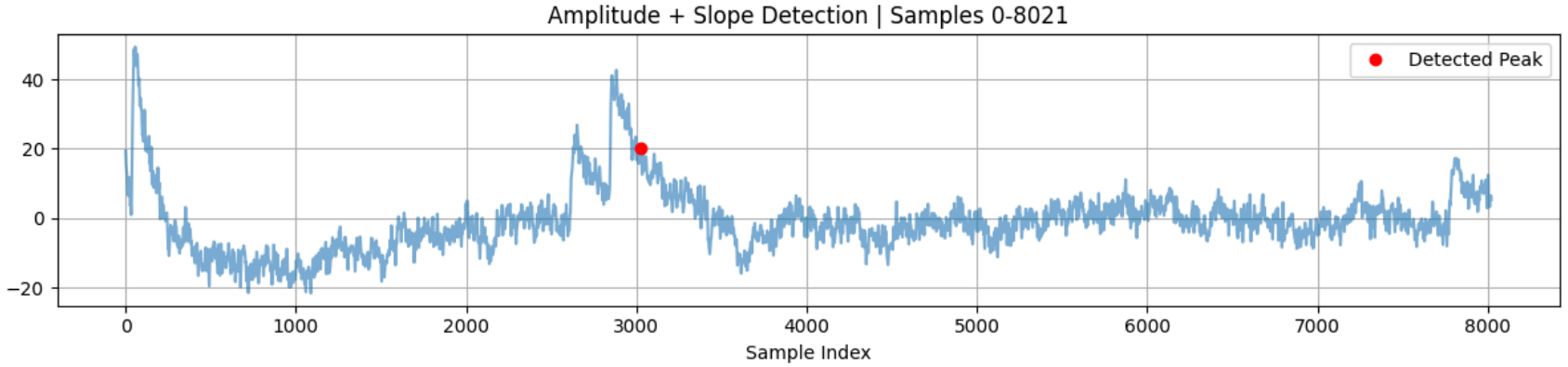
In [ ]:
```
# This is iPSC especific which is the first priority with developing this pipeline. I will eventually Like to make ea
# Step 1: Detect peaks with optional visualization
initial_peak_output = detect_peaksv2(
    signal=filtered_data,
    delta=13,
    slope_thresh=0.35,
    distance=400,
    stop=600_000,
    step=10_000,
    visualize_all=False  # Show just first 10 peaks
) # this has ["peaks"] and ['params'] which returns the parameters used

params = initial_peak_output['params']
initial_peak_output = initial_peak_output['peak_indices']


# Step 2: Refine to true apex (center of event)
refined_peaks = refine_peaks_to_apex(
    signal=filtered_data,
    peak_indices=initial_peak_output,
    search_radius=80
)

# Step 3: Visualize a few
visualize_refined_peaks(
    signal=filtered_data,
    refined_peaks=refined_peaks,
    num_peaks=10,
```

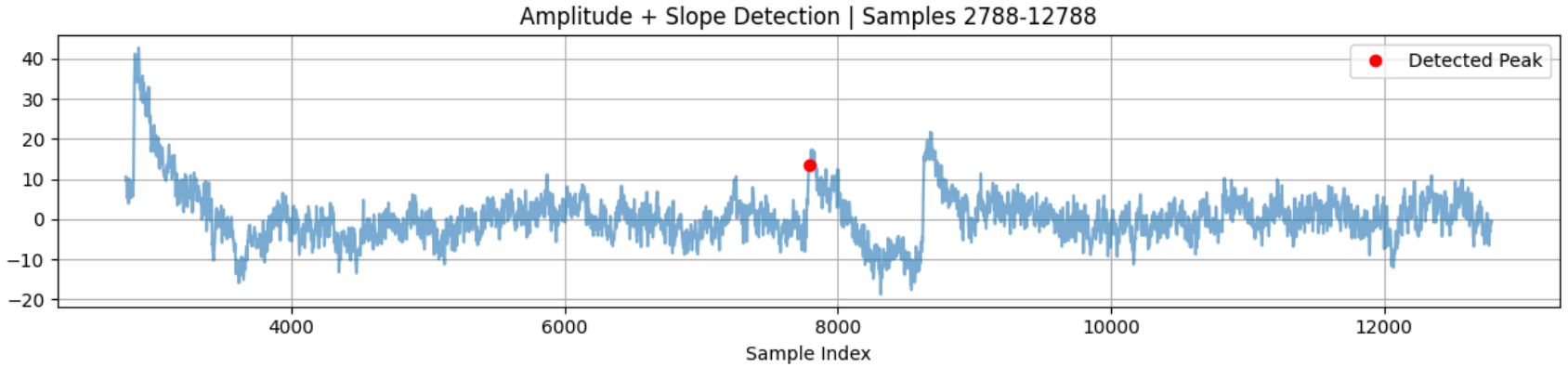
```
    window=5_500
)
'''
# EPSC Specific (commented out)
epsc_output = detect_minima_v2(
    signal=filtered_data,
    delta=13,
    slope_thresh=0.35,
    distance=400,
    stop=600_000,
    step=10_000,
    visualize_all=False  # Show just first 10 peaks
)

params = epsc_output['params']
initial_peak_output = epsc_output['peak_indices']

refined_peaks = refine_minima_to_apex( # local minimum within a window
    signal=filtered_data,
    minima_indices=initial_peak_output,
    search_radius=80
)

visualize_refined_minima(
    signal=filtered_data,
    refined_minima = refined_peaks,
    num_minima = 10,
    window=5_500
)
'''

'''
evaluate_threshold_performance(
    refined_peaks=refined_peaks,
    ground_truth_indices=ground_truth_indices,
    filtered_data=filtered_data,
    tolerance=TOLERANCE,
    zoom=(0, 2_000_000)
)
'''
```

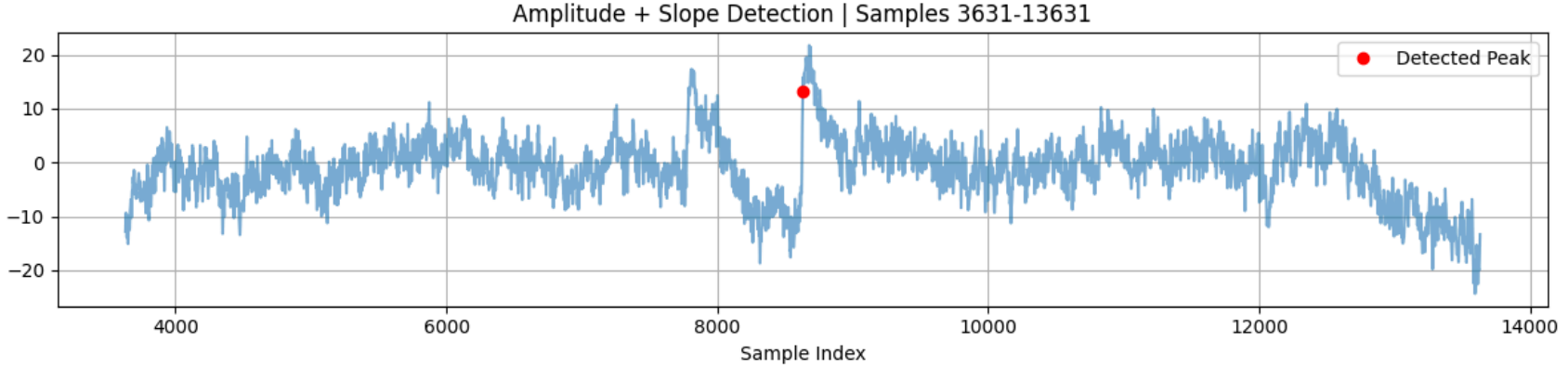
🔎 Total detected peaks: 310

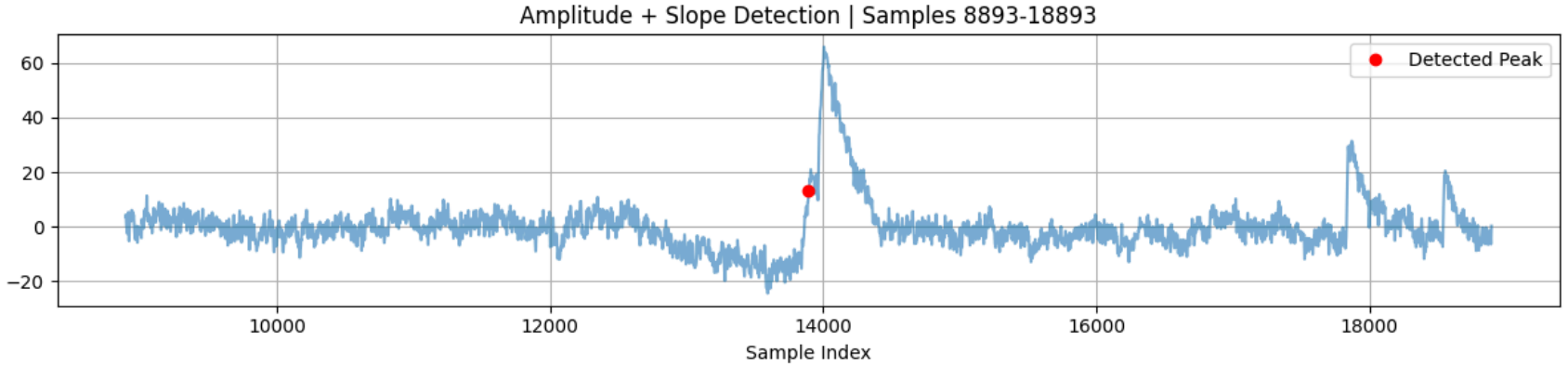

Amplitude + Slope Detection | Samples 0-5037
Detected Peak
40
20
0
−20
0
1000
2000
3000
4000
5000
Sample Index

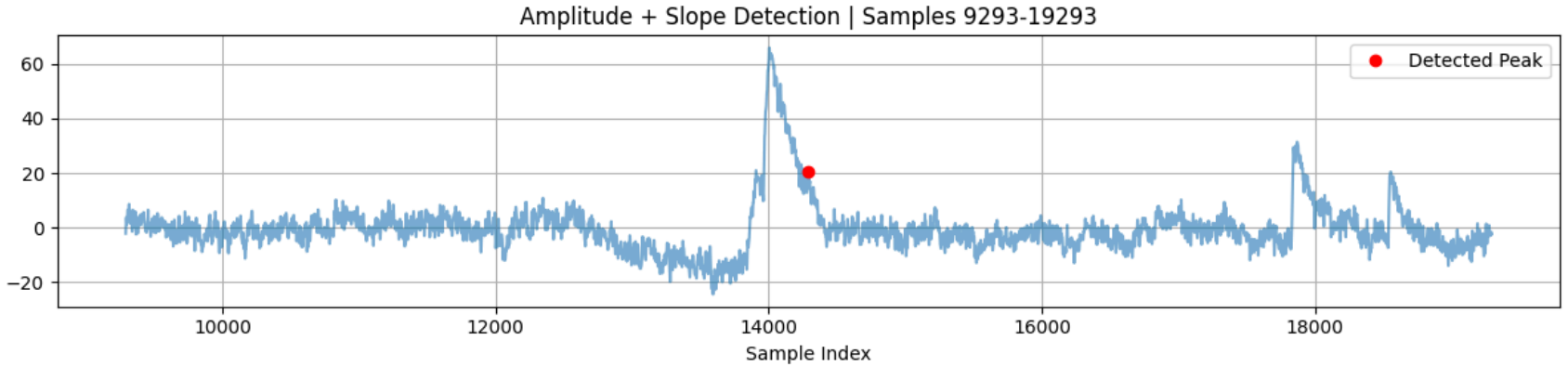

Amplitude + Slope Detection | Samples 0-7621
Detected Peak
40
20
0
−20
0
1000
2000
3000
4000
5000
6000
7000
8000
Sample Index

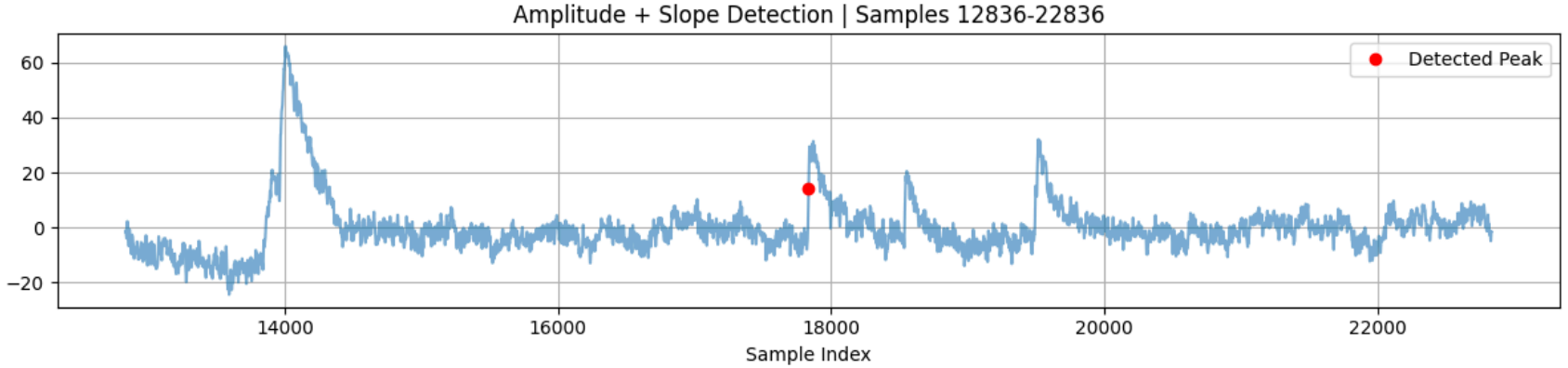

Amplitude + Slope Detection | Samples 0-8021
Detected Peak
40
20
0
−20
0
1000
2000
3000
4000
5000
6000
7000
8000
Sample Index

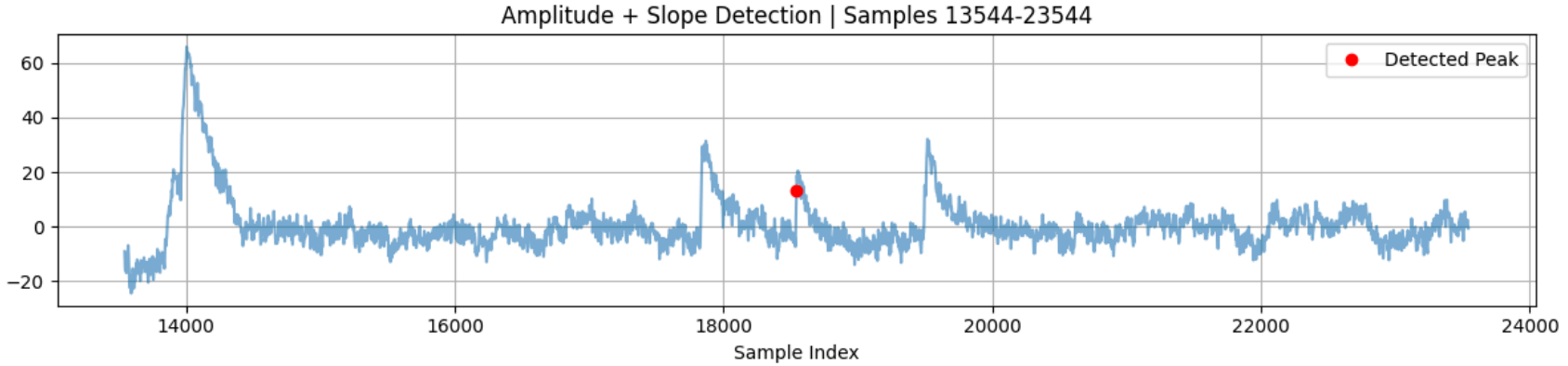

Amplitude + Slope Detection | Samples 2788-12788
Detected Peak
40
30
20
10
0
−10
−20
4000
6000
8000
10000
12000
Sample Index

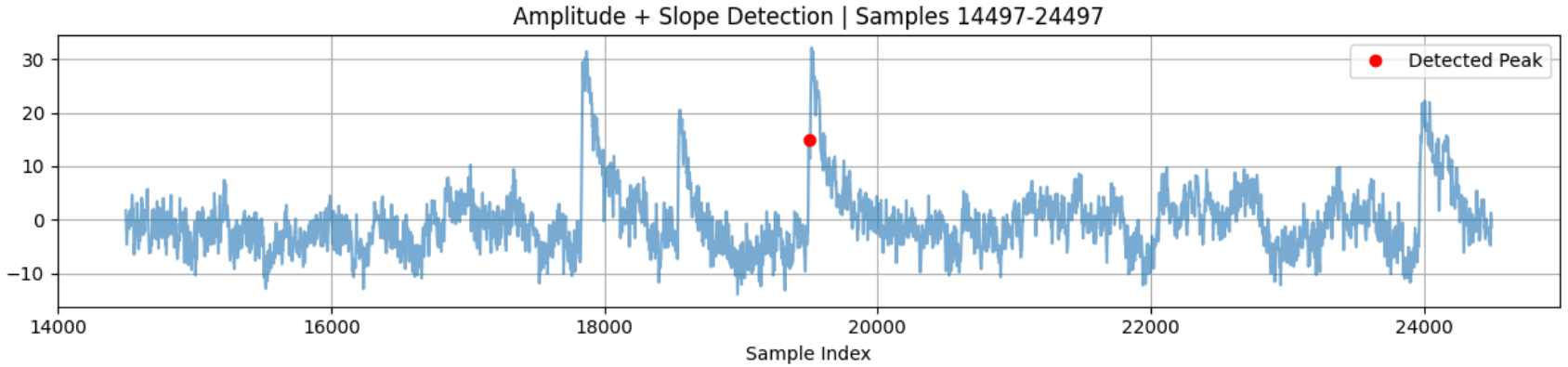

Amplitude + Slope Detection | Samples 3631-13631
Detected Peak
20
10
0
−10
−20
4000
6000
8000
10000
12000
14000
Sample Index

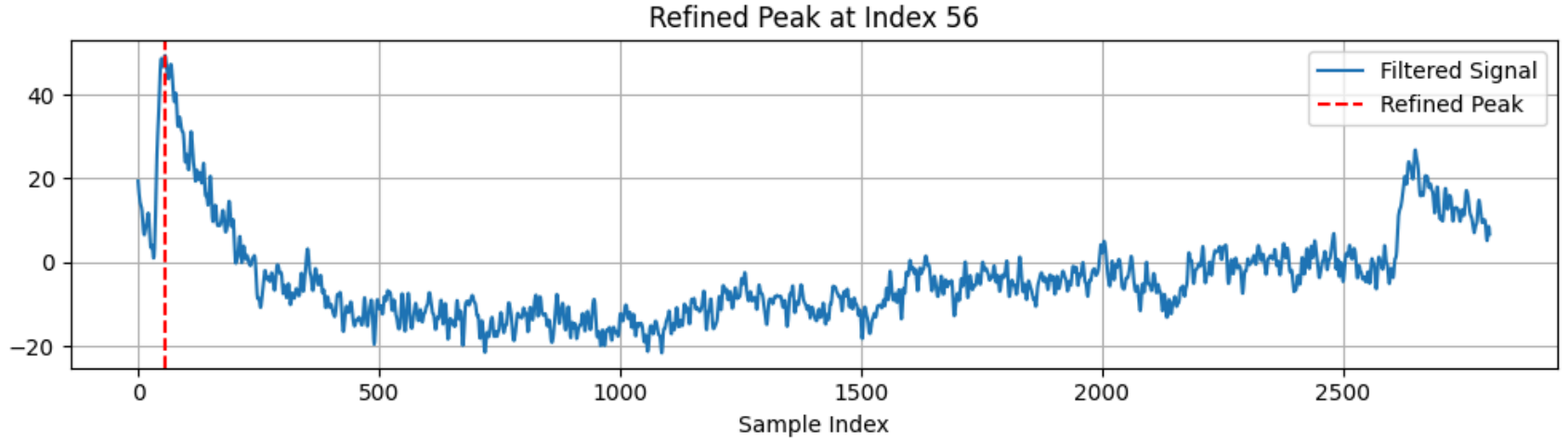

Amplitude + Slope Detection | Samples 8893-18893
Detected Peak
60
40
20
0
−20
10000
12000
14000
16000
18000
Sample Index

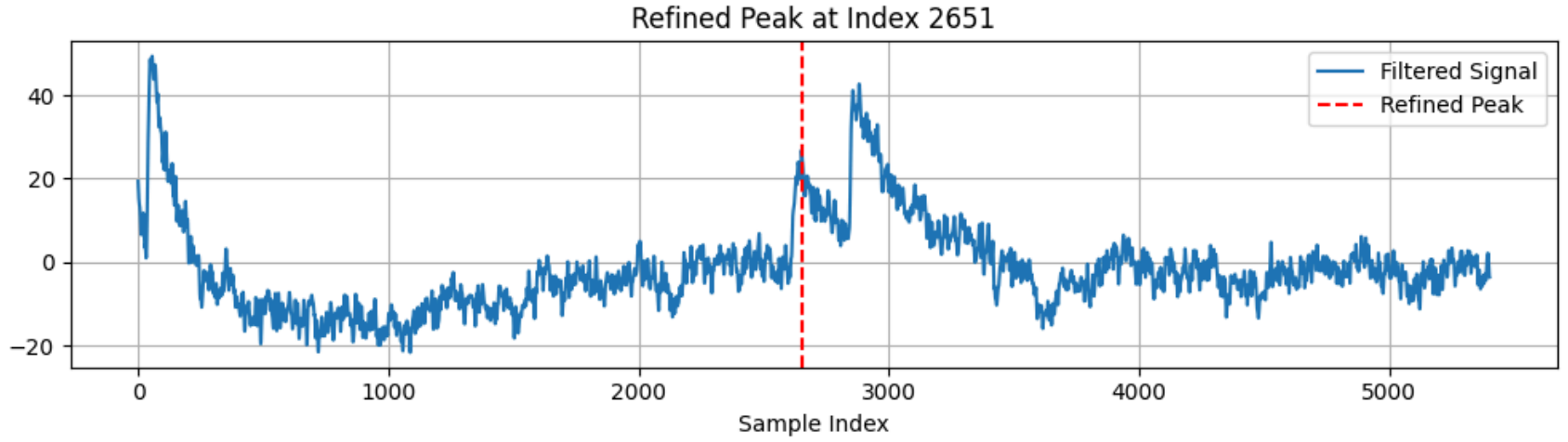

Amplitude + Slope Detection | Samples 9293-19293
Detected Peak
60
40
20
0
−20
10000
12000
14000
16000
18000
Sample Index

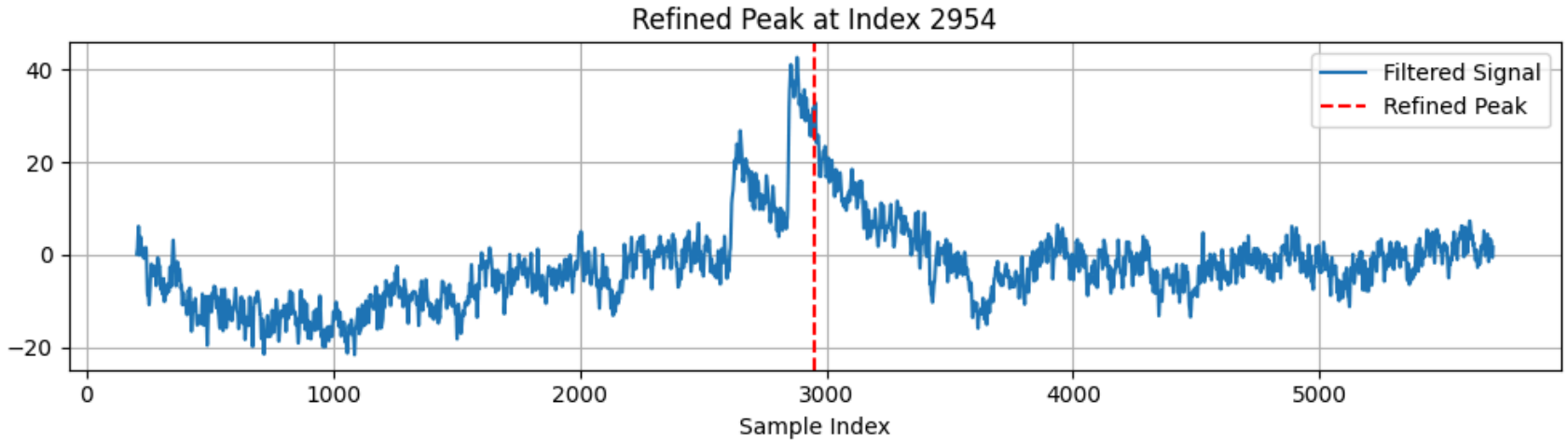

Amplitude + Slope Detection | Samples 12836-22836
Detected Peak
60
40
20
0
−20
14000
16000
18000
20000
22000
Sample Index

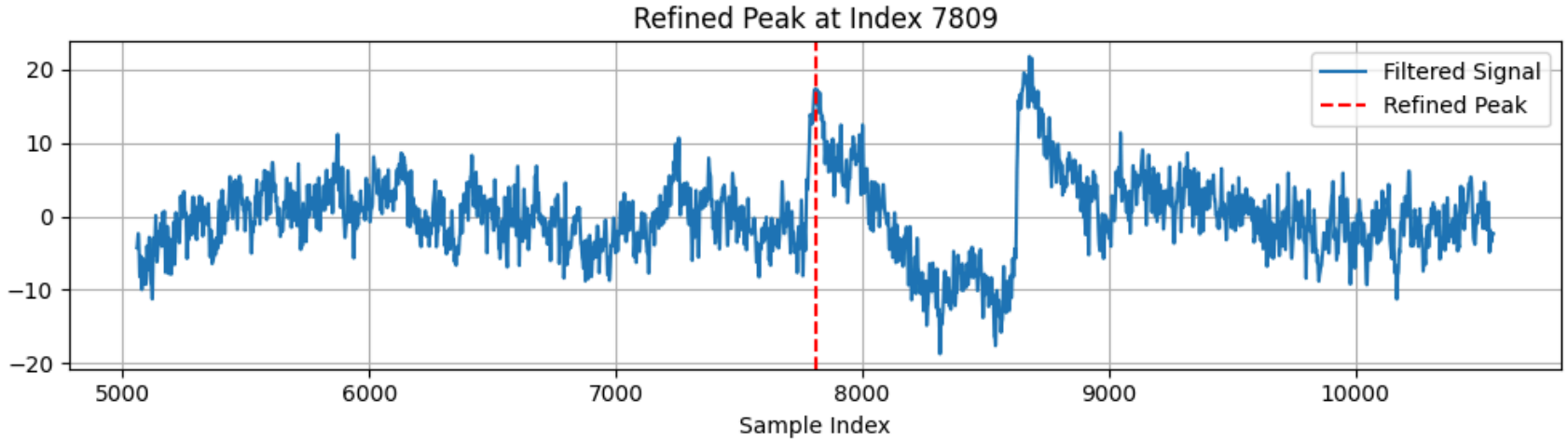

Amplitude + Slope Detection | Samples 13544-23544
Detected Peak
60
40
20
0
−20
14000
16000
18000
20000
22000
24000
Sample Index

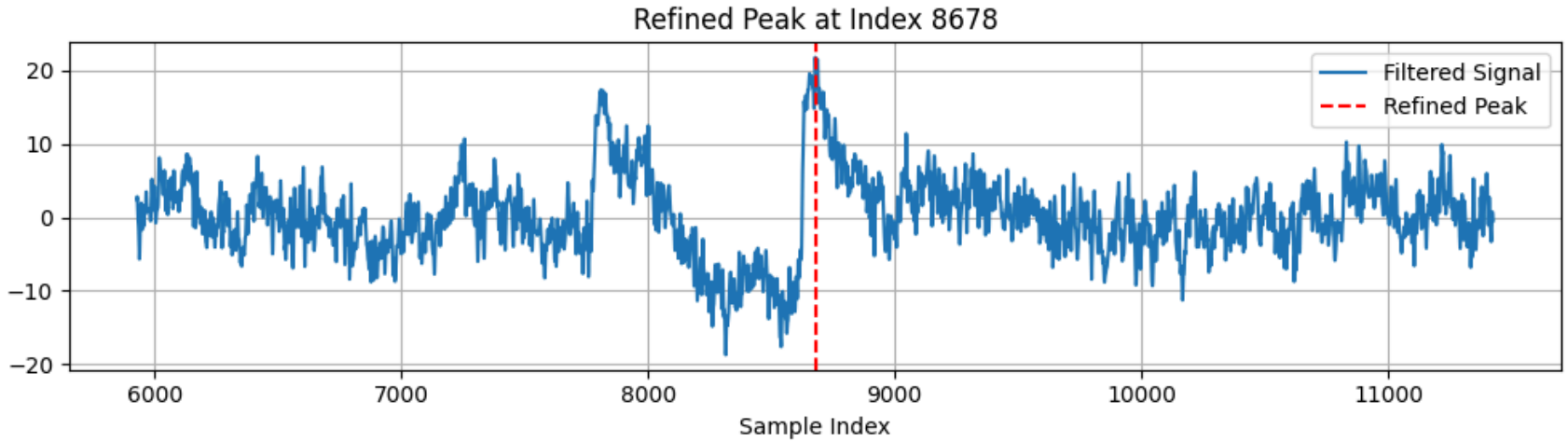

Amplitude + Slope Detection | Samples 14497-24497
Detected Peak
30
20
10
0
−10
14000
16000
18000
20000
22000
24000
Sample Index

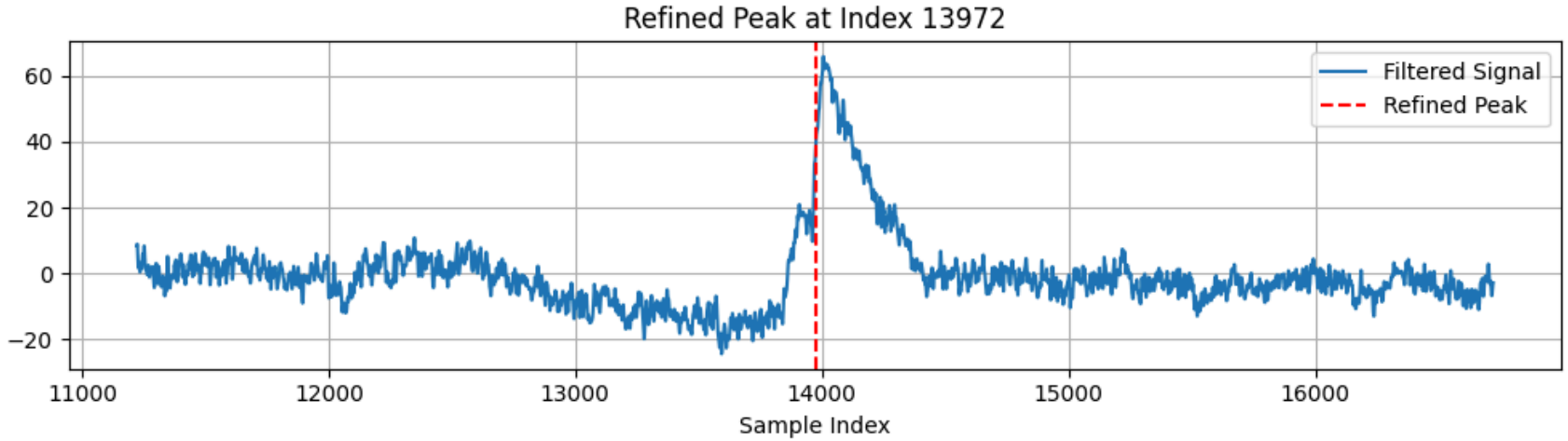

Refined Peak at Index 56
Filtered Signal
Refined Peak
40
20
0
−20
0
500
1000
1500
2000
2500
Sample Index

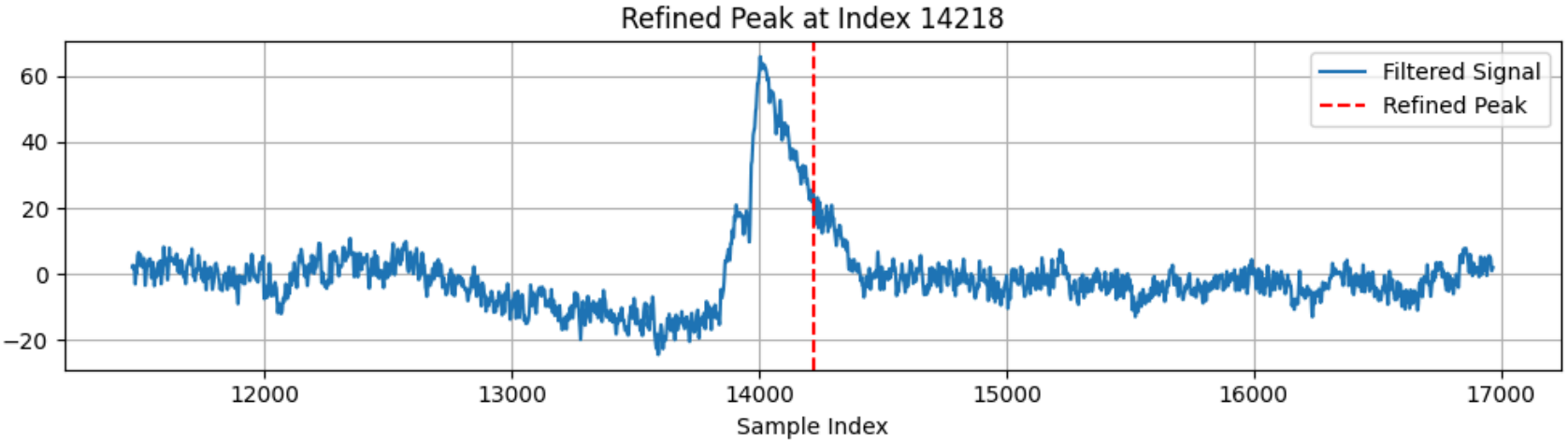

Refined Peak at Index 2651
Filtered Signal
Refined Peak
40
20
0
−20
0
1000
2000
3000
4000
5000
Sample Index

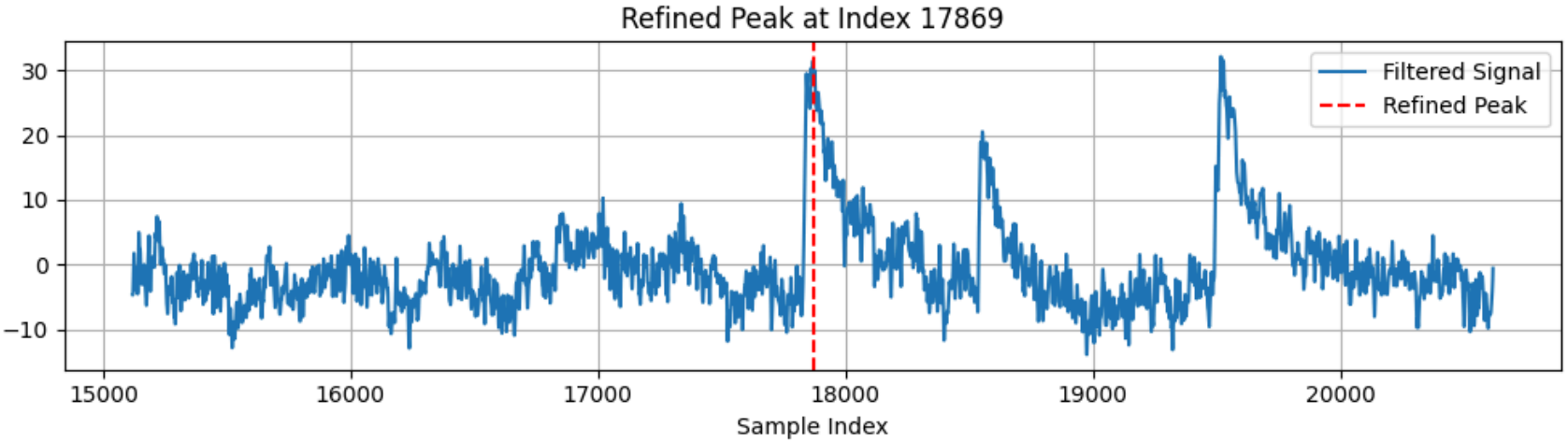

Refined Peak at Index 2954
Filtered Signal
Refined Peak
40
20
0
−20
0
1000
2000
3000
4000
5000
Sample Index

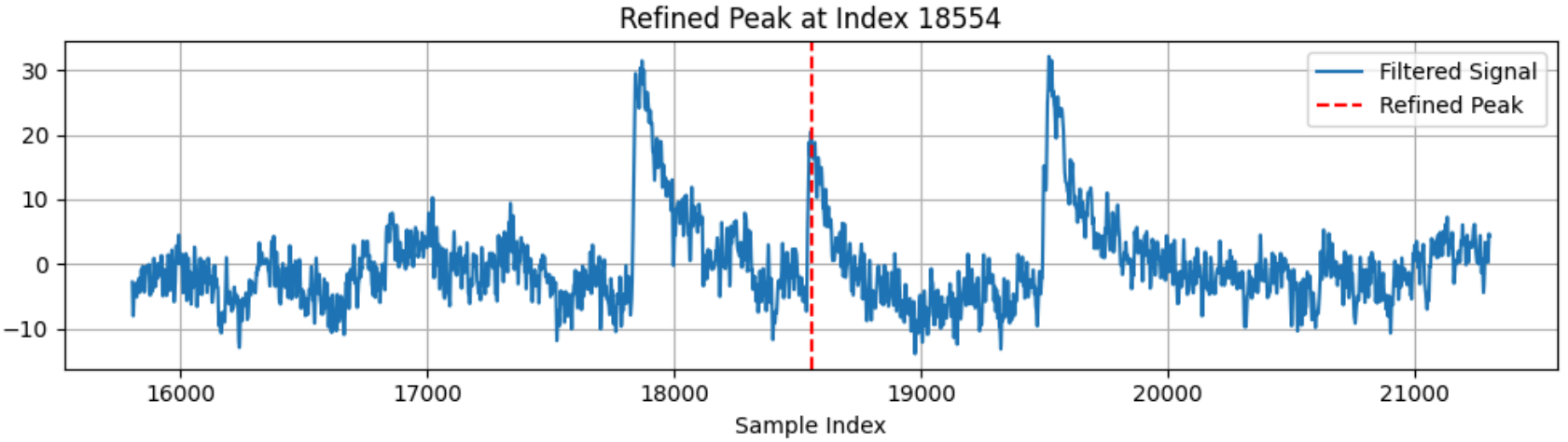

Refined Peak at Index 7809
Filtered Signal
Refined Peak
20
10
0
−10
−20
5000
6000
7000
8000
9000
10000
Sample Index

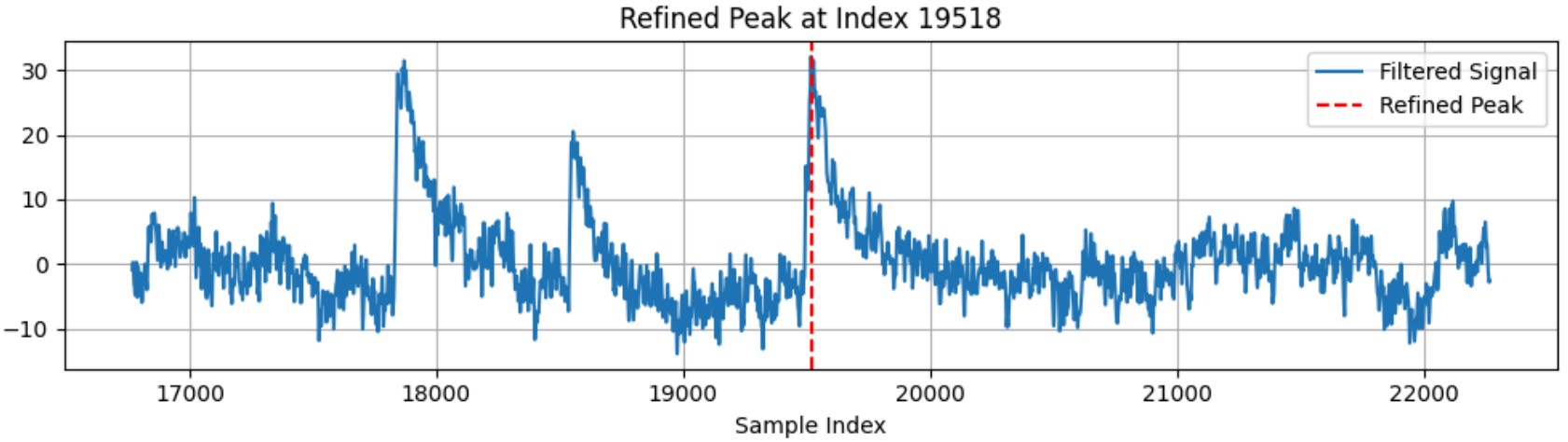

Refined Peak at Index 8678
Filtered Signal
Refined Peak
20
10
0
−10
−20
6000
7000
8000
9000
10000
11000
Sample Index


Refined Peak at Index 13972
Filtered Signal
Refined Peak
60
40
20
0
−20
11000
12000
13000
14000
15000
16000
Sample Index


Refined Peak at Index 14218
Filtered Signal
Refined Peak
60
40
20
0
−20
12000
13000
14000
15000
16000
17000
Sample Index


Refined Peak at Index 17869
Filtered Signal
Refined Peak
30
20
10
0
−10
15000
16000
17000
18000
19000
20000
Sample Index





Out[ ]: '\nevaluate_threshold_performance(\n refined_peaks=refined_peaks,\n ground_truth_indices=ground_truth_indices,\n filtered_data=filtered_data,\n tolerance=TOLERANCE,\n zoom=(0, 2_000_000)\n)\n'

In [ ]:
```
# === MODULAR FUNCTIONS FOR PIPELINE ===

# -------------------------------------
# 1a. Window Extraction
# -------------------------------------
def extract_waveform_windows(signal, peak_indices, pre_samples=40, post_samples=80):
    """
    Extract waveform windows around each detected peak.
    """
```

```
    waveforms = []
    for peak in peak_indices:
        if peak - pre_samples >= 0 and peak + post_samples < len(signal):
            waveforms.append(signal[peak - pre_samples : peak + post_samples])
    waveforms = np.array(waveforms)
    print(f"✅ [Step 1a: extract_waveform_windows] Extracted {waveforms.shape[0]} waveforms | Window size: {waveform
    return waveforms

# -------------------------------------
# 1b. Normalize Waveforms
# -------------------------------------
def normalize_waveforms(waveforms):
    """
    Normalize each waveform to unit peak amplitude.
    """
    normalized = np.array([
        w / np.max(np.abs(w)) if np.max(np.abs(w)) != 0 else w for w in waveforms
    ])
    print(f"✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: {normalized.shape}\n")
    return normalized

# -------------------------------------
# 2. Wavelet Decomposition
# -------------------------------------
def wavelet_decompose_waveforms(waveforms, wavelet='db4', level=2):
    """
    Decompose each waveform using DWT and flatten coefficients.
    """
    coeff_matrix = []
    for i, waveform in enumerate(waveforms):
        coeffs = pywt.wavedec(waveform, wavelet=wavelet, level=level)
        coeff_vector = np.concatenate(coeffs)
        coeff_matrix.append(coeff_vector)
        if i == 0:
            level_shapes = ", ".join([f"Level {j}: {c.shape}" for j, c in enumerate(coeffs)])
            print(f"🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into {len(coeffs)} levels:")
            print(f"    ↪ {level_shapes} | Total coeff length: {len(coeff_vector)}")
    coeff_matrix = np.array(coeff_matrix)
    print(f"✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: {coeff_matrix.shape}\n")
    return coeff_matrix

# -------------------------------------
```

```python
# 3. KS-Test Feature Selection
# ------------------------------------
def select_wavelet_features_via_ks(coeff_matrix, alpha=0.05):
    """
    Keep features that are non-Gaussian using KS-test.
    """
    selected_indices = []
    for i in range(coeff_matrix.shape[1]):
        _, pval = kstest(coeff_matrix[:, i], 'norm')
        if pval < alpha:
            selected_indices.append(i)
    selected = coeff_matrix[:, selected_indices]
    print(f"📊 [Step 3: select_wavelet_features_via_ks] Selected {selected.shape[1]} / {coeff_matrix.shape[1]} featu
    return selected

# ------------------------------------
# 4. PCA Reduction
# ------------------------------------
def apply_pca(features, n_components=2):
    """
    Reduce dimensionality of features using PCA.
    """
    pca = PCA(n_components=n_components)
    X_pca = pca.fit_transform(features)
    explained = np.round(pca.explained_variance_ratio_ * 100, 2)
    print(f"🔻 [Step 4: apply_pca]")
    print(f"    ↪ Reduced to {n_components} components")
    print(f"    ↪ Explained variance ratio: {explained.tolist()}%")
    print(f"    ↪ Cumulative explained variance: {np.sum(explained)}%\n")
    return X_pca, pca

# ------------------------------------
# 5. Run Clustering Algorithms
# ------------------------------------
def run_all_clusterings(X):
    """
    Run KMeans, Spectral Clustering, and HDBSCAN.
    """
    results = {}

    # KMeans
    kmeans = KMeans(n_clusters=2, random_state=42)
```

```
    results['kmeans'] = kmeans.fit_predict(X)
    print(f" 🔹 [KMeans] Clusters: {np.unique(results['kmeans'])} | Counts: {np.bincount(results['kmeans'])}")

    # Spectral Clustering
    #spectral = SpectralClustering(n_clusters=2, affinity='nearest_neighbors', assign_labels='kmeans', random_state=4
    #results['spectral'] = spectral.fit_predict(X)
    #print(f" 🔸 [SpectralClustering] Clusters: {np.unique(results['spectral'])} | Counts: {np.bincount(results['spec

    # HDBSCAN
    hdb = hdbscan.HDBSCAN(min_cluster_size=10)
    labels = hdb.fit_predict(X)
    results['hdbscan'] = labels
    print(f" 🔻 [HDBSCAN] Clusters (incl. noise -1): {np.unique(labels)} | Counts: {np.bincount(labels + 1)}\n")

    return results

# ------------------------------------
# 6. Pipeline Wrapper
# ------------------------------------
def run_waveform_clustering_pipeline(
    signal,
    peak_indices,
    pre_samples=40,
    post_samples=60,
    wavelet_level=2,
    pca_components=2
):
    """
    Full pipeline: window → normalize → wavelet → KS-test → PCA → clustering
    """
    # get windows
    waveforms = extract_waveform_windows(signal, peak_indices, pre_samples, post_samples)
    # normalize amplitudes of signals for every windowed wave
    normalized_waveforms = normalize_waveforms(waveforms)
    #
    coeffs = wavelet_decompose_waveforms(normalized_waveforms, wavelet='haar' ,level=wavelet_level)

    selected = select_wavelet_features_via_ks(coeffs)

    X_pca, pca_model = apply_pca(selected, n_components=pca_components)

    clustering_results = run_all_clusterings(X_pca)
```

```
    # replace the spectral result by overwriting it
    spectral = SpectralClustering(n_clusters=2, affinity='nearest_neighbors', random_state=42)
    clustering_results['spectral'] = spectral.fit_predict(selected)


    print(f"📈 [Pipeline] Total Explained Variance by PCA: {np.round(np.sum(pca_model.explained_variance_ratio_) * 1

    return {
        "waveforms": waveforms,
        "normalized_waveforms": normalized_waveforms,
        "coeffs": coeffs,
        "selected": selected,
        "X_pca": X_pca,
        "pca_model": pca_model,
        "clustering_results": clustering_results,
        "params": {
            "Signal Length": len(signal),
            "Number of Peaks": len(peak_indices),
            "Pre-samples": pre_samples,
            "Post-samples": post_samples,
            "Wavelet Level": wavelet_level,
            "PCA Components": pca_components
        }
    }


print("Starting pipeline...\n")
# ------------------------------------
# 7. Run Pipeline Wrapper + Plot HERE <----
# ------------------------------------
results = run_waveform_clustering_pipeline(
    signal=filtered_data, #
    peak_indices=refined_peaks,
    pre_samples=40,
    post_samples=60,
    wavelet_level=3,
    pca_components=2
)
```

```
# -------------------------------------



# Summary printout
print("Pipeline complete.\n")
print(f"\n# of refined peaks: {len(refined_peaks)}\n")
for method, labels in results["clustering_results"].items():
    unique, counts = np.unique(labels, return_counts=True)
    print(f"{method.upper()} - Labels: {unique} | Counts: {counts}")

'''
# Plot cumulative PCA variance
explained_var = np.cumsum(results["pca_model"].explained_variance_ratio_)
px.area(
    x=range(1, len(explained_var) + 1),
    y=explained_var,
    labels={"x": "# Components", "y": "Explained Variance"},
    title="Cumulative Explained Variance by PCA"
)
'''
```

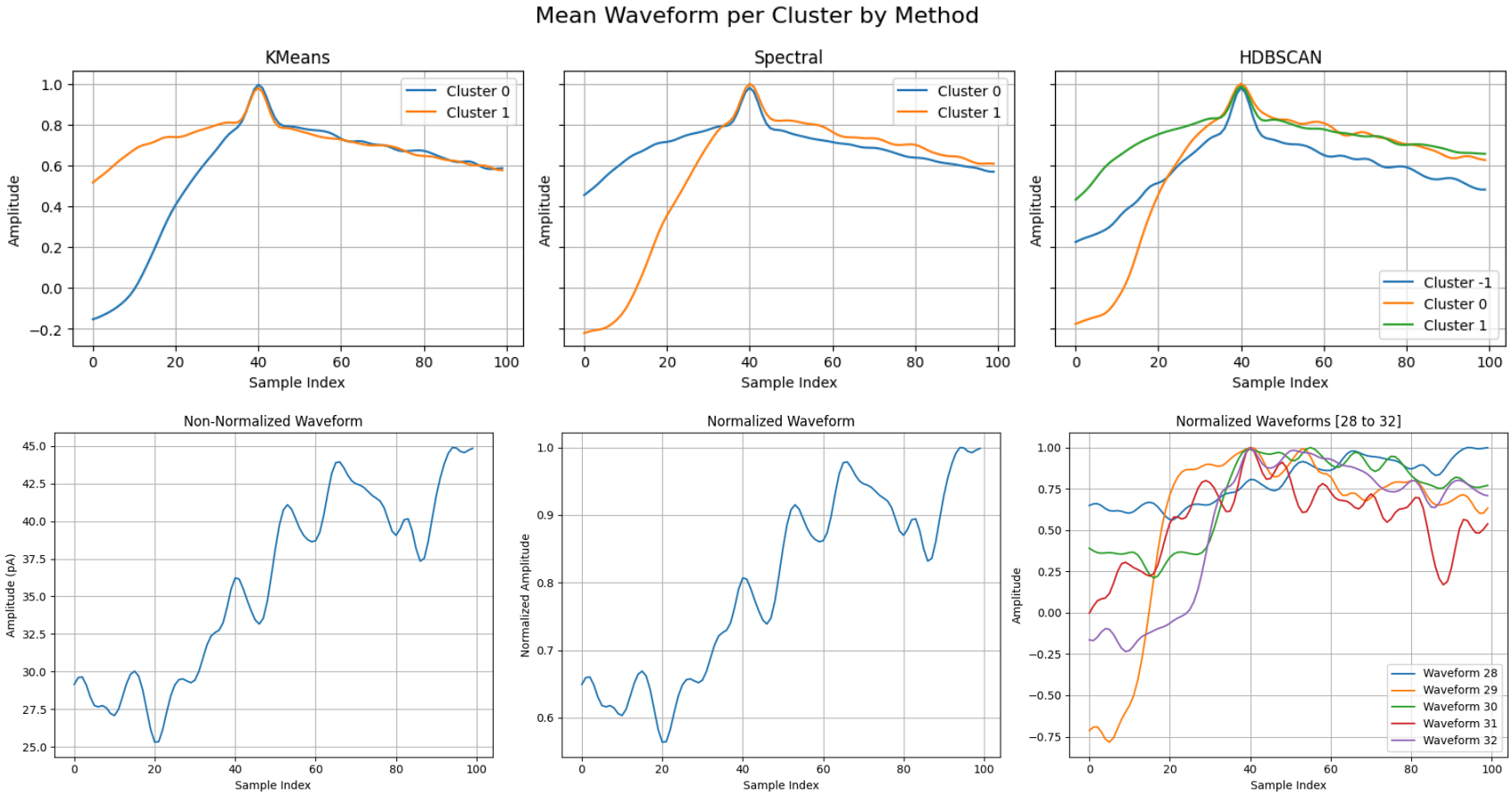
```
Starting pipeline...

✅ [Step 1a: extract_waveform_windows] Extracted 310 waveforms | Window size: 100 samples

✅ [Step 1b: normalize_waveforms] Normalized waveforms | Shape: (310, 100)

🌀 [Step 2: wavelet_decompose_waveforms] First waveform decomposed into 4 levels:
   ↪ Level 0: (13,), Level 1: (13,), Level 2: (25,), Level 3: (50,) | Total coeff length: 101
✅ [Step 2: wavelet_decompose_waveforms] Coefficient matrix shape: (310, 101)

📊 [Step 3: select_wavelet_features_via_ks] Selected 101 / 101 features (non-Gaussian)

🔻 [Step 4: apply_pca]
   ↪ Reduced to 2 components
   ↪ Explained variance ratio: [47.06, 27.24]%
   ↪ Cumulative explained variance: 74.3%

🔹 [KMeans] Clusters: [0 1] | Counts: [115 195]
🔻 [HDBSCAN] Clusters (incl. noise -1): [-1  0  1] | Counts: [129  40 141]

📈 [Pipeline] Total Explained Variance by PCA: 74.3%

Pipeline complete.


# of refined peaks: 310

KMEANS - Labels: [0 1] | Counts: [115 195]
HDBSCAN - Labels: [-1  0  1] | Counts: [129  40 141]
SPECTRAL - Labels: [0 1] | Counts: [224  86]
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renam
ed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/utils/deprecation.py:151: FutureWarning: 'force_all_finite' was renam
ed to 'ensure_all_finite' in 1.6 and will be removed in 1.8.
  warnings.warn(
```

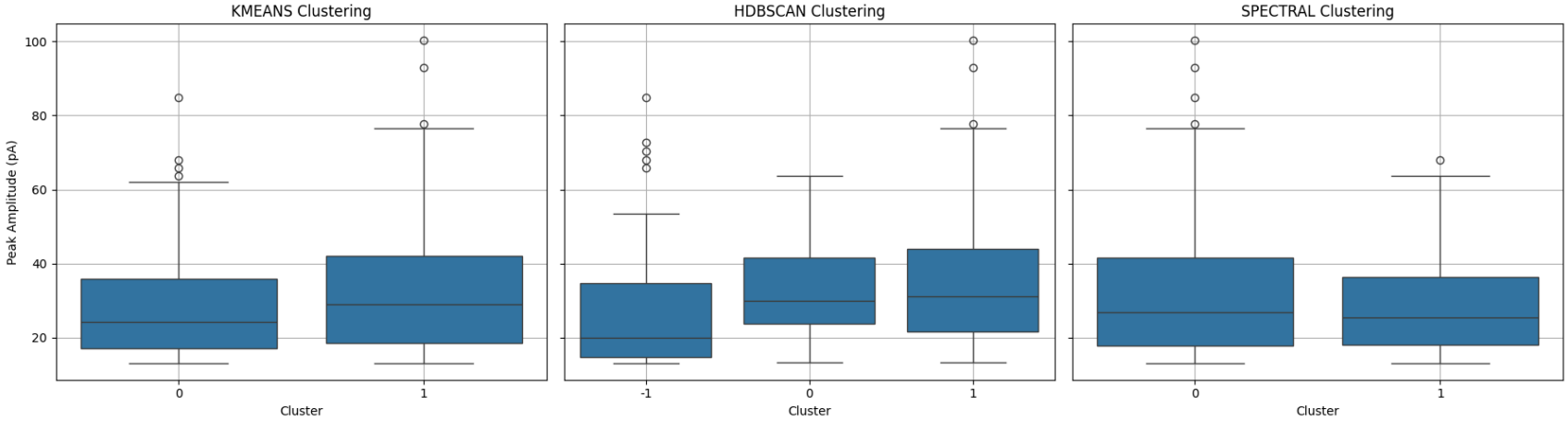
Out[ ]:
```
'\n# Plot cumulative PCA variance\nexplained_var = np.cumsum(results["pca_model"].explained_variance_ratio_)\npx.are
a(\n    x=range(1, len(explained_var) + 1),\n    y=explained_var,\n    labels={"x": "# Components", "y": "Explained
Variance"},\n    title="Cumulative Explained Variance by PCA"\n)\n'
```

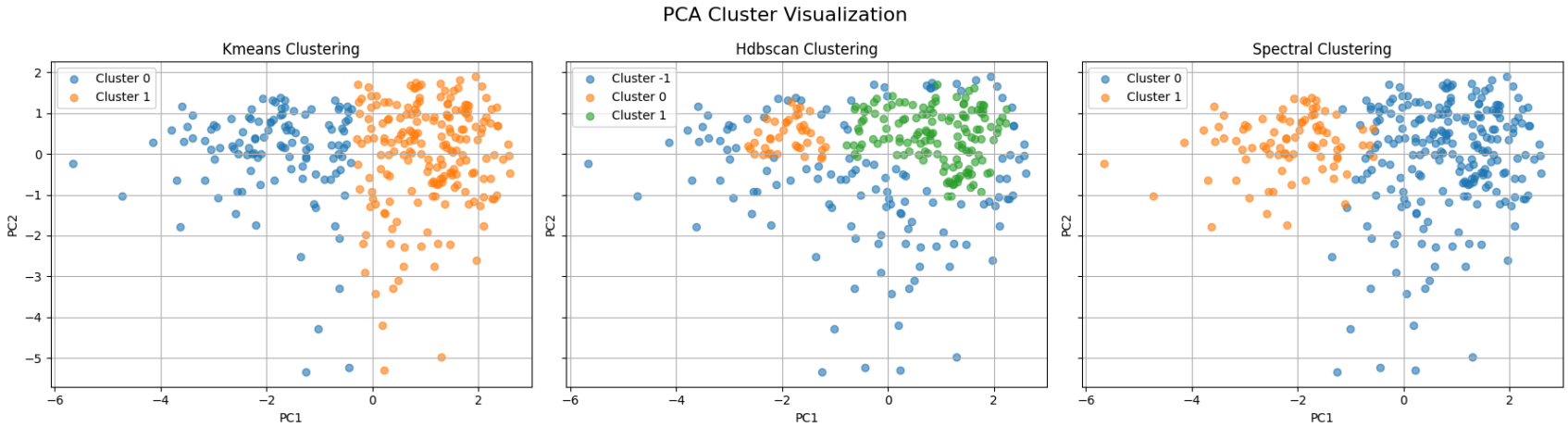
In [ ]:

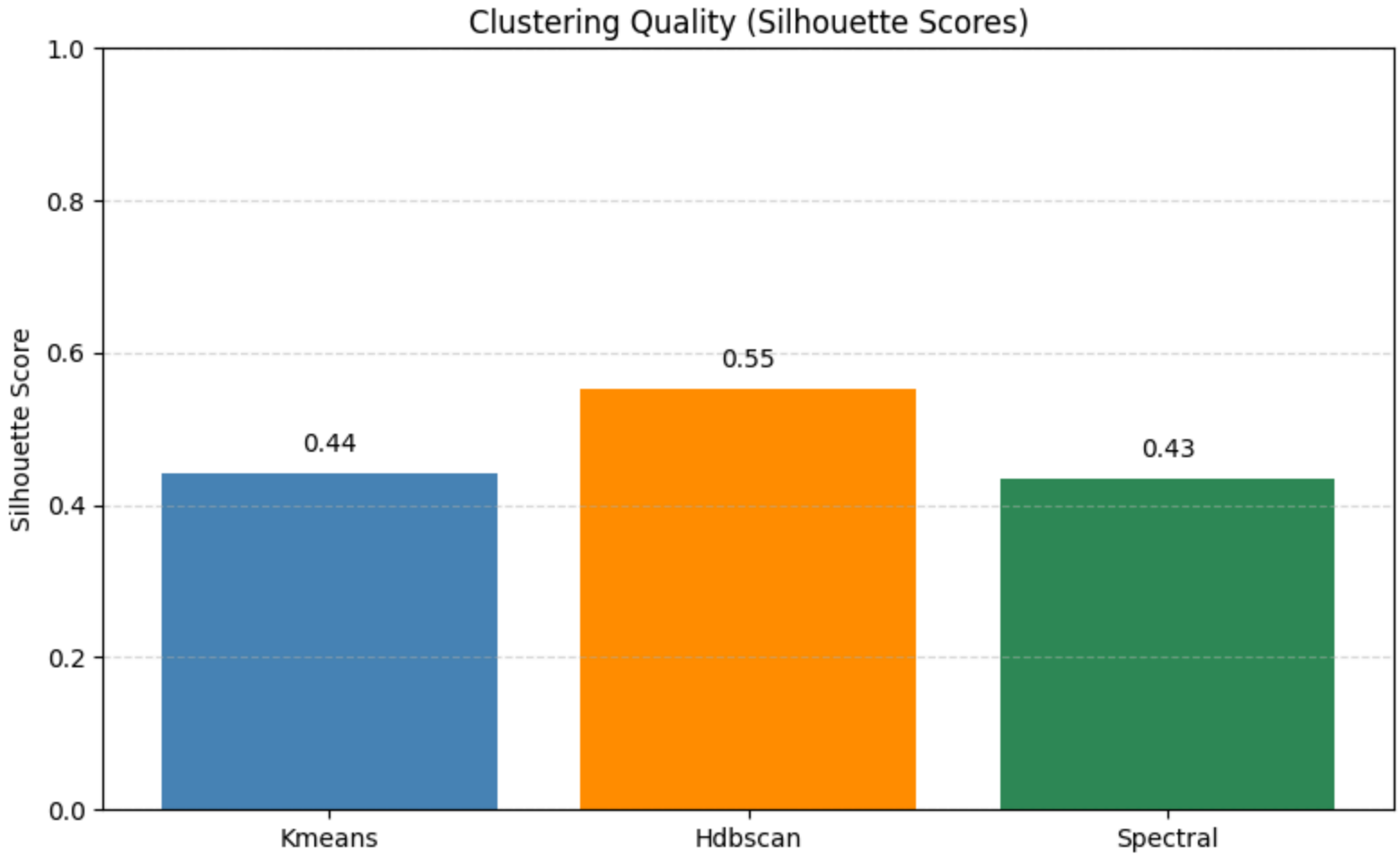
```
# ================================
# CLUSTER VISUALIZATION & INSPECTION
# ================================

# -- Visualize cluster-average waveforms
plot_cluster_means_from_dict({
    "KMeans": (results["normalized_waveforms"], results['clustering_results']["kmeans"]),
    "Spectral": (results["normalized_waveforms"], results['clustering_results']["spectral"]),
    "HDBSCAN": (results["normalized_waveforms"], results['clustering_results']["hdbscan"])  # raw for amplitude
}, normalize=False)

# For further use
waveforms = results["waveforms"]
normalized_waveforms = results["normalized_waveforms"]
wavelet_coeffs = results['coeffs']
selected_wavelet_coeffs = results['selected']
X_pca = results["X_pca"]
pca_model = results["pca_model"]
clustering_results = results["clustering_results"]

# -- Run waveform inspection for sanity check
plot_waveform_inspection(waveforms, normalized_waveforms, index_wave=28, n_examples=5)

# -- Plot amplitude distributions across clusters
plot_amplitude_distribution_across_clusters(waveforms, clustering_results)

# ================================
# 📉 PCA CLUSTER PLOTS
# ================================


# -- Plot PCA-reduced data with cluster labels
plot_pca_clusters(X_pca, clustering_results)


# ================================
# 📊 SILHOUETTE SCORES
# ================================


# -- Compute silhouette scores
```

```
scores = compute_silhouette_scores(X_pca, clustering_results)

# -- Plot silhouette scores
plot_silhouette_scores(scores)

# -- Print clustering quality summary
print_silhouette_summary(scores, clustering_results)
```




KMEANS Clustering
HDBSCAN Clustering
SPECTRAL Clustering
Peak Amplitude (pA)
Cluster


PCA Cluster Visualization
Kmeans Clustering
Hdbscan Clustering
Spectral Clustering
Cluster 0
Cluster 1
Cluster -1
PC1
PC2



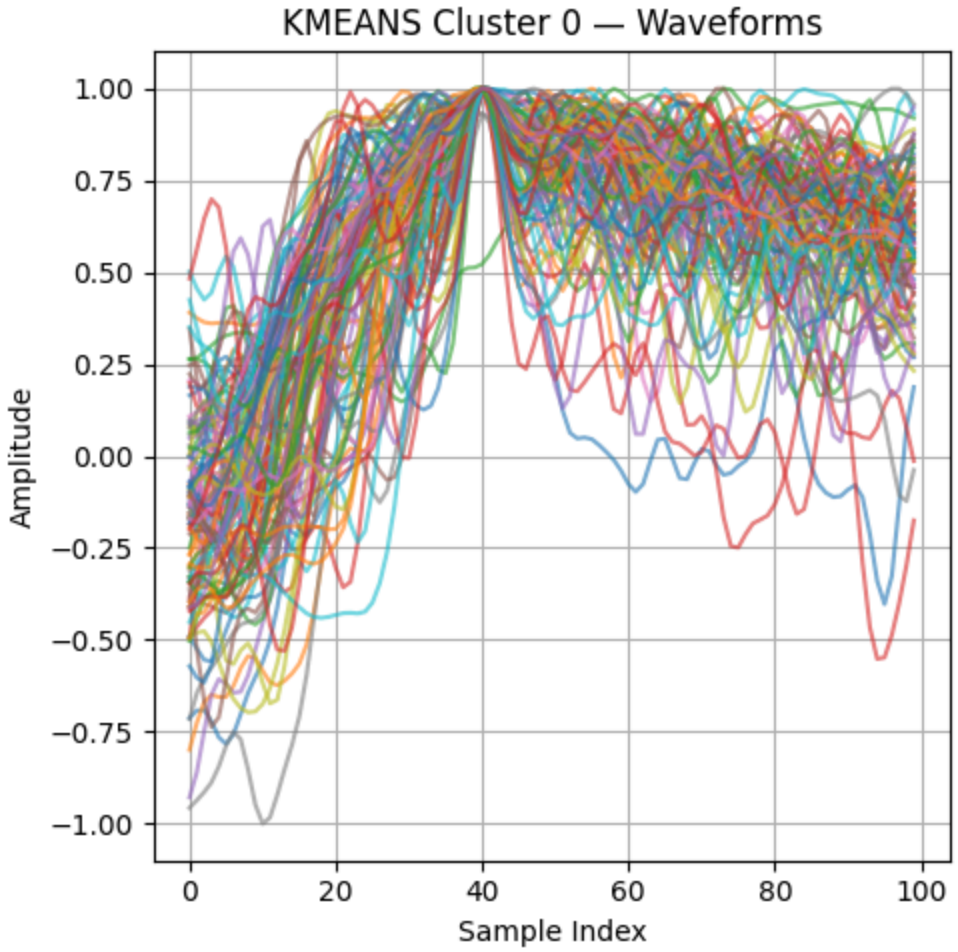
🧠 Silhouette Score Summary:

- Kmeans: 0.443 with 2 cluster(s) → ⚠️ Weak
- Hdbscan: 0.553 with 2 cluster(s) → ✅ Good
- Spectral: 0.435 with 2 cluster(s) → ⚠️ Weak

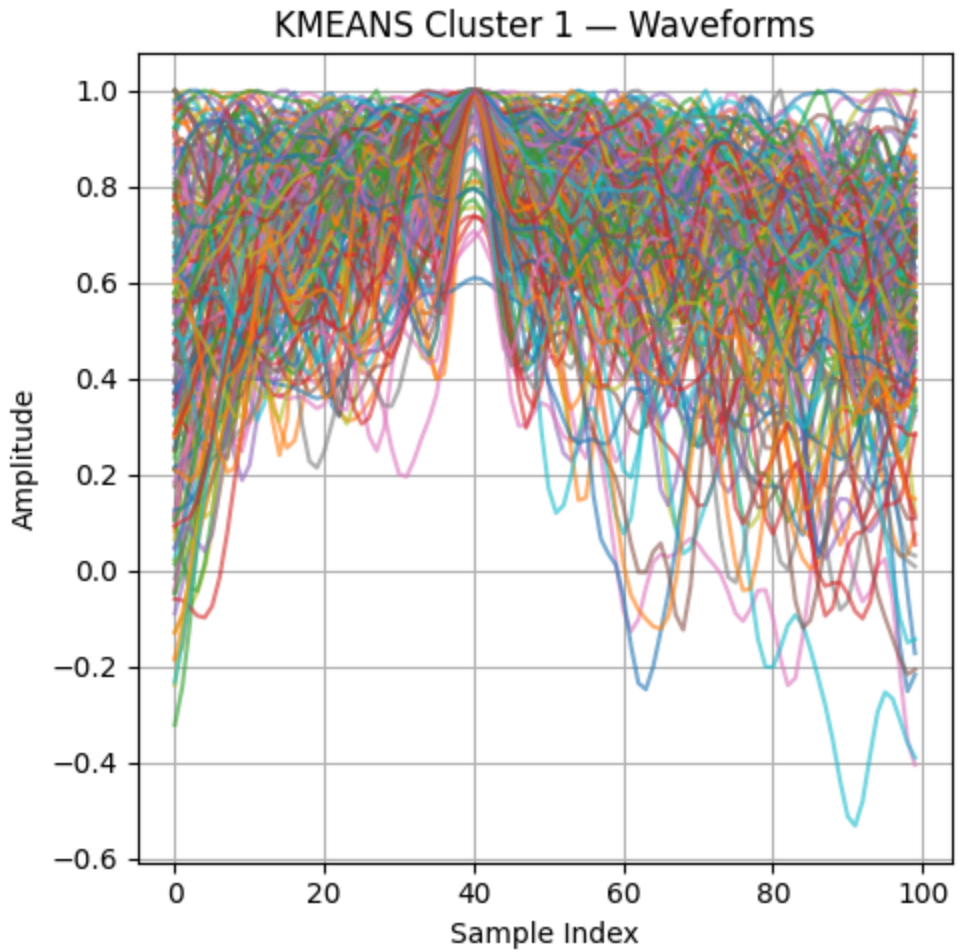
In [ ]:
```
# ================================
# 🔍 INSPECT KMEANS CLUSTERING OUTPUT
# ================================

# -- 1. Raw labels from clustering method
use_clustering_method = "kmeans"
```

```
labels_for_each_window = clustering_results[use_clustering_method]  # ndarray of shape (n_waveforms,)
print(f" 🔹 KMeans assigned labels for {len(labels_for_each_window)} waveform windows.")
print("Labels:", labels_for_each_window)

# -- 2. Identify waveform indices per cluster
cluster_0_window_indices = np.where(labels_for_each_window == 0)[0]
cluster_1_window_indices = np.where(labels_for_each_window == 1)[0]
print(f"🟦 Cluster 0: {len(cluster_0_window_indices)} windows → Indices: {cluster_0_window_indices}")
print(f"🟥 Cluster 1: {len(cluster_1_window_indices)} windows → Indices: {cluster_1_window_indices}")

# ================================
# 📈 VISUALIZE INDIVIDUAL CLUSTERS
# ================================


# -- 3. Plot each KMeans cluster (non-normalized for amplitude comparison)
cluster_visualize_overlayed_waveforms(
    waveforms=normalized_waveforms,
    clustering_results=clustering_results,
    method= use_clustering_method,
    cluster_number=0, # Adjust
    normalize=False
)

cluster_visualize_overlayed_waveforms(
    waveforms=normalized_waveforms,
    clustering_results=clustering_results,
    method=use_clustering_method,
    cluster_number=1, # Adjust
    normalize=False
)

# ================================
# 🧮 EXTRACT WAVEFORMS PER CLUSTER
# ================================

# -- 4. Retrieve waveform arrays from each cluster
cluster_0_waveforms = normalized_waveforms[clustering_results[use_clustering_method] == 0]
cluster_1_waveforms = normalized_waveforms[clustering_results[use_clustering_method] == 1]

print(f"🟦 Cluster 0 Waveforms → Shape: {cluster_0_waveforms.shape}")
print(f"🟥 Cluster 1 Waveforms → Shape: {cluster_1_waveforms.shape}")
```

🔹 KMeans assigned labels for 310 waveform windows.
Labels: [0 1 1 ... 0 1 0]
🟦 Cluster 0: 115 windows → Indices: [0   3   5 ... 305 307 309]
🟥 Cluster 1: 195 windows → Indices: [1   2   4 ... 304 306 308]
📊 [KMEANS] Overlaying 115 waveforms from Cluster 0



📊 [KMEANS] Overlaying 195 waveforms from Cluster 1

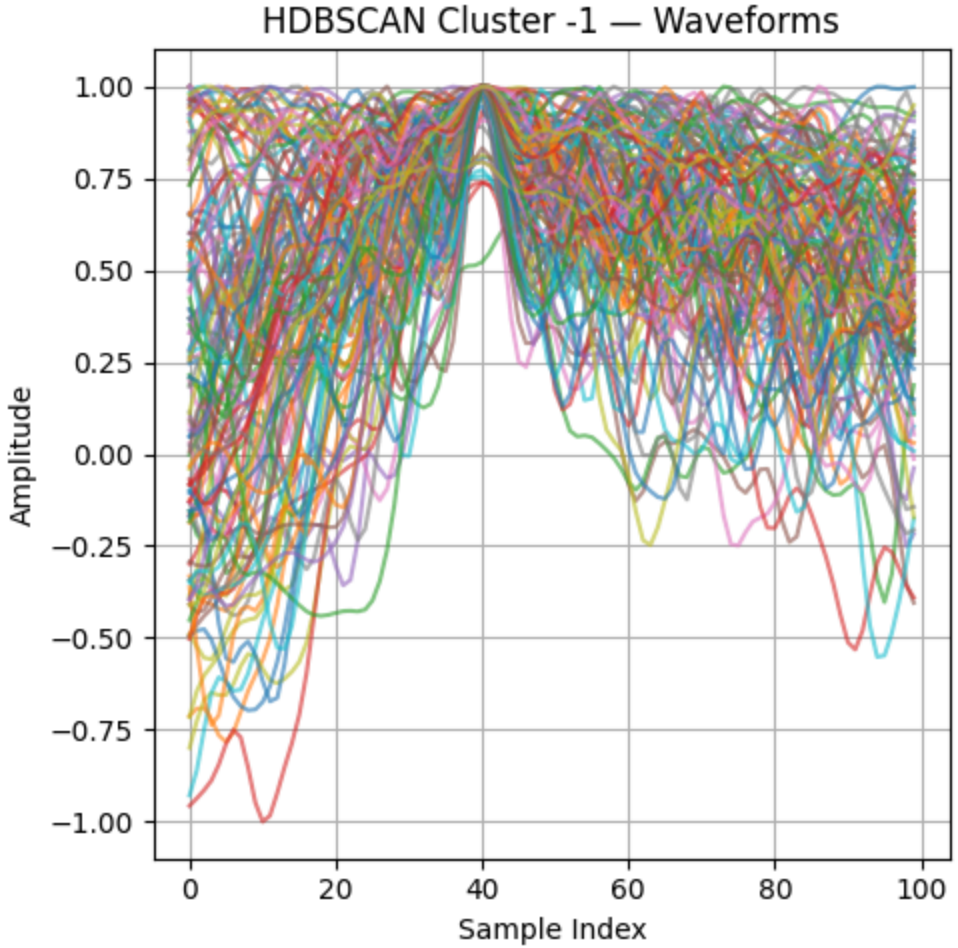


🟦 Cluster 0 Waveforms → Shape: (115, 100)
🟥 Cluster 1 Waveforms → Shape: (195, 100)

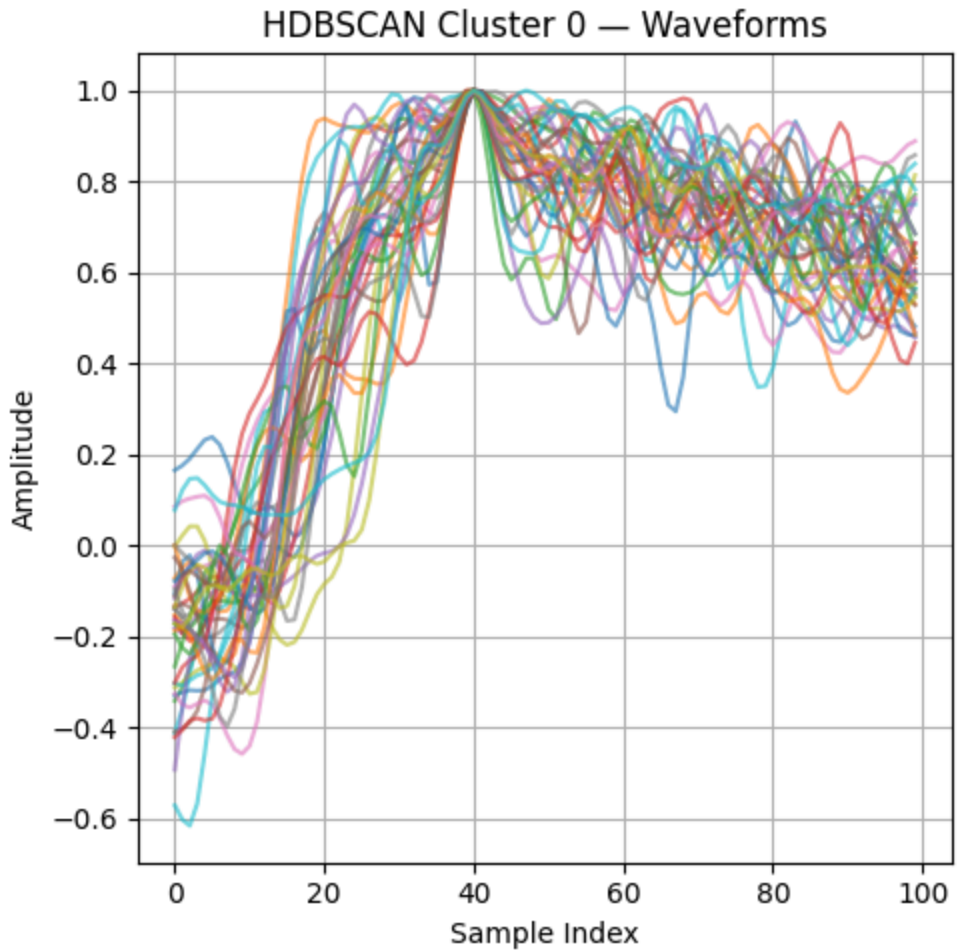
In [ ]:
```
# ================================
# 🔍 INSPECT HDBSCAN CLUSTERING OUTPUT
# ================================

use_clustering_method = "hdbscan"
labels_for_each_window = clustering_results[use_clustering_method]  # array of shape (n_waveforms,)

print(f"🔹 HDBSCAN assigned labels to {len(labels_for_each_window)} waveform windows.")
print(f"📦 Unique cluster labels: {np.unique(labels_for_each_window)} (includes -1 for noise if present)")
print("🔢 All labels:", labels_for_each_window)
```

```python
# ================================
# 📍 INDEX WAVEFORMS PER CLUSTER
# ================================

unique_clusters = np.unique(labels_for_each_window)
cluster_indices = {label: np.where(labels_for_each_window == label)[0] for label in unique_clusters}

for label, indices in cluster_indices.items():
    if label == -1:
        print(f"🟥 Noise (Cluster -1): {len(indices)} windows → Indices: {indices}")
    else:
        print(f"🟩 Cluster {label}: {len(indices)} windows → Indices: {indices}")

# ================================
# 📈 VISUALIZE ALL INDIVIDUAL CLUSTERS (INCLUDING NOISE)
# ================================

for label in unique_clusters:
    cluster_visualize_overlayed_waveforms(
        waveforms=normalized_waveforms,
        clustering_results=clustering_results,
        method=use_clustering_method,
        cluster_number=label,
        normalize=False
    )



# ================================
# 🧮 EXTRACT WAVEFORMS PER CLUSTER
# ================================

# Store waveform arrays by cluster in a dictionary
waveform_dict_by_cluster = {}
for label in unique_clusters:
    waveform_dict_by_cluster[label] = normalized_waveforms[labels_for_each_window == label]
    cluster_name = "Noise (-1)" if label == -1 else f"Cluster {label}"
    print(f"📦 {cluster_name} → Shape: {waveform_dict_by_cluster[label].shape}")

#waveform_dict_by_cluster
```

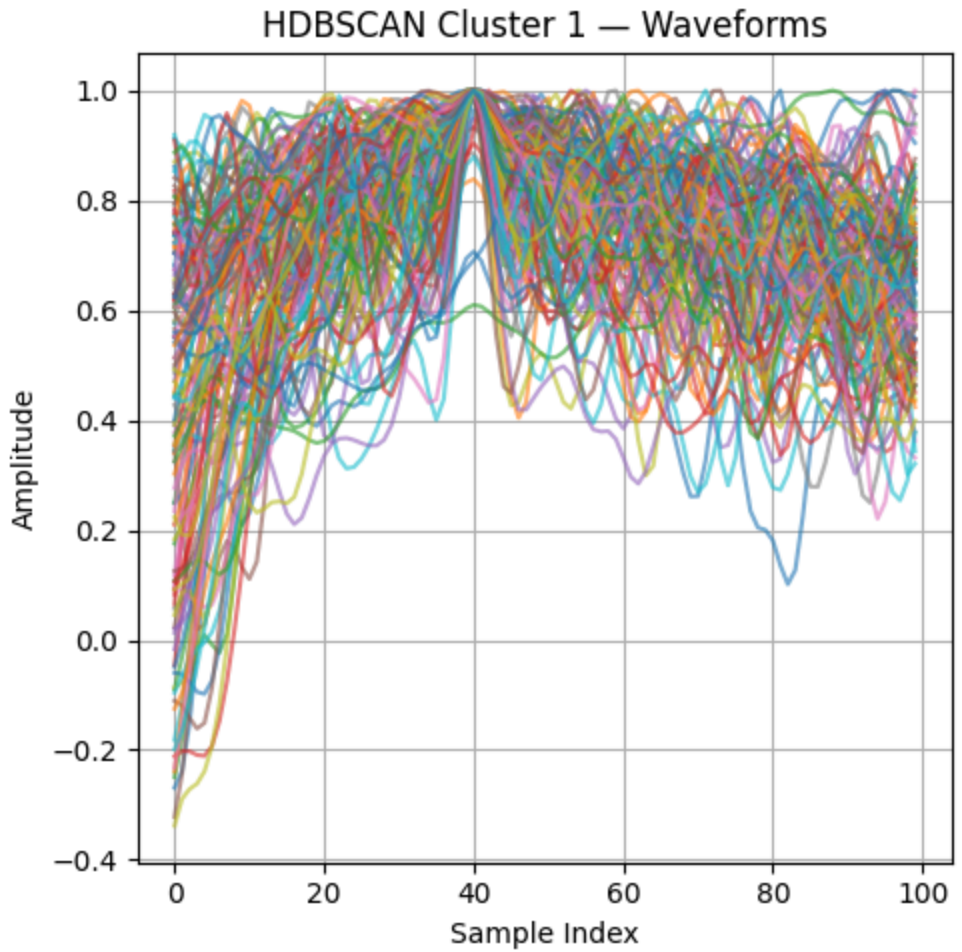
```
🔹 HDBSCAN assigned labels to 310 waveform windows.
📦 Unique cluster labels: [-1  0  1] (includes -1 for noise if present)
🔢 All labels: [ 0 -1 -1 ...  1  1 -1]
🟥 Noise (Cluster -1): 129 windows → Indices: [  1   2   5 ... 304 305 309]
🟩 Cluster 0: 40 windows → Indices: [  0   9  16 ... 290 293 294]
🟩 Cluster 1: 141 windows → Indices: [  3   4   7 ... 306 307 308]
📊 [HDBSCAN] Overlaying 129 waveforms from Cluster -1
```




```
📊 [HDBSCAN] Overlaying 40 waveforms from Cluster 0
```



📊 [HDBSCAN] Overlaying 141 waveforms from Cluster 1




```
📤 Noise (-1) → Shape: (129, 100)
📤 Cluster 0 → Shape: (40, 100)
📤 Cluster 1 → Shape: (141, 100)
```

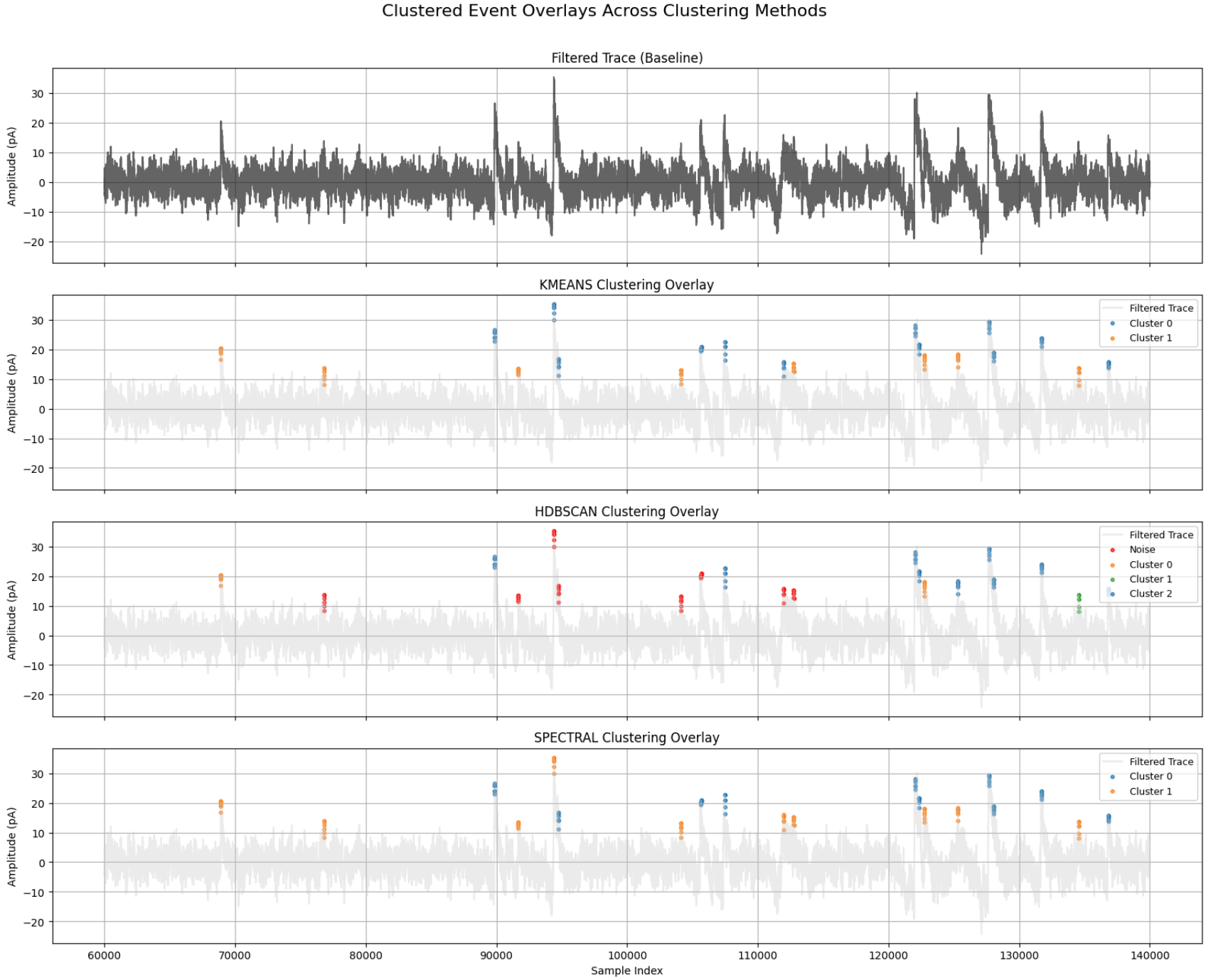
In [ ]:

In [ ]:
```
print("\n==============================")
print("📌 FEATURE PIPELINE CHECKPOINT")
print("==============================")

# Raw wavelet coefficients
print(f"🔹 Coeff Matrix (wavelet features): shape = {wavelet_coeffs.shape}")
print(f"    ↪ Example: {wavelet_coeffs[0][:10]} ...\n")
```

```
# KS-selected features
print(f"🔸 Selected Features (via KS-test): shape = {selected_wavelet_coeffs.shape}")
print(f"    ↪ Example: {selected_wavelet_coeffs[0][:10]} ...\n")

# Waveforms before and after normalization
print(f"🟦 Raw Waveforms: shape = {waveforms.shape}")
print(f"🟥 Normalized Waveforms: shape = {normalized_waveforms.shape}")

print("==============================\n")

# Line 470
plot_cluster_maps_across_methods(
    filtered_data=filtered_data,
    peak_indices=refined_peaks,
    clustering_results=clustering_results,
    start=60_000,
    end=140_000,
)
```

```
==============================
📌 FEATURE PIPELINE CHECKPOINT
==============================
🔹 Coeff Matrix (wavelet features): shape = (367, 101)
   ↪ Example: [1.4784 2.0147 1.8362 ... 1.719  1.875  1.5396] ...

🔸 Selected Features (via KS-test): shape = (367, 101)
   ↪ Example: [1.4784 2.0147 1.8362 ... 1.719  1.875  1.5396] ...

🟦 Raw Waveforms: shape = (367, 100)
🟥 Normalized Waveforms: shape = (367, 100)
==============================
```

```
<ipython-input-3-ad71f98e90b5>:573: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib
3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()`` or ``pyplo
t.get_cmap()`` instead.
  cmap = get_cmap(cmap_name)
```


Clustered Event Overlays Across Clustering Methods
Filtered Trace (Baseline)
KMEANS Clustering Overlay
HDBSCAN Clustering Overlay
SPECTRAL Clustering Overlay
Amplitude (pA)
Sample Index
Filtered Trace
Noise
Cluster 0
Cluster 1
Cluster 2

In [ ]:

```
# ======================================
# 🔍 Event Detection Evaluation Utilities
# ======================================
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.metrics import precision_score, recall_score, f1_score, confusion_matrix, ConfusionMatrixDisplay
from collections import Counter

# ------------------------------
# 🔧 Global Parameters
# ------------------------------
SAMPLING_RATE = 20_000  # Hz
TOLERANCE = 60          # samples
TRUE_CLUSTER_COLOR = "red"
FALSE_CLUSTER_COLOR = "blue"
base_filename = os.path.basename(abf_filepath1)

# ------------------------------------------------------------------------------
# 1. ✅ Load Ground Truth Events from CSV
# ------------------------------------------------------------------------------
def load_ground_truth(csv_filepath, column='Event Time (s)   ', sampling_rate=20_000):
    """
    ✅ Load and convert ground truth event times from CSV into sample indices.

    🧠 Automatically detects if the time is in milliseconds or seconds.

    📥 Parameters:
    - csv_filepath: path to the CSV file containing event annotations
    - column: name of the column with event times
    - sampling_rate: sampling rate in Hz (default 20,000)

    📤 Returns:
    - df: cleaned pandas DataFrame
    - gt_indices: numpy array of sample indices (ints)
    """

    df = pd.read_csv(csv_filepath)
    df[column] = pd.to_numeric(df[column], errors='coerce')
    df = df.dropna(subset=[column])
```

```
    # Auto-detect units: assume ms if the max is over 200 (200s)
    if df[column].max() > 200:
        print("⚠️ Detected units in milliseconds. Converting to seconds...")
        event_times_sec = df[column] / 1000
    else:
        print("✅ Detected units in seconds.")
        event_times_sec = df[column]

    gt_indices = (event_times_sec * sampling_rate).round().astype(int).to_numpy()

    print(f"📦 Loaded {len(gt_indices)} ground truth events")
    return df, gt_indices



# ------------------------------------------------------------------------
# 2. ✅ Match Detected Peaks to Ground Truth
# ------------------------------------------------------------------------
def match_detected_to_ground_truth(detected, ground_truth, tolerance=TOLERANCE):
    """
    ✅ Matches detected peaks to ground truth events using a tolerance window.

    📥 Parameters:
    - detected: array of detected event indices (e.g., from thresholding)
    - ground_truth: array of GT event indices (e.g., from CSV)
    - tolerance: range within which a match is accepted (in samples)

    📤 Returns:
    - TP: number of true positives (matched detections)
    - FP: number of false positives (detected but unmatched)
    - FN: number of false negatives (missed ground truths)
    - matched_gt: set of ground truth indices that were matched

    📌 Use these metrics to evaluate basic peak detection performance.
    """
    TP, FP = 0, 0
    matched_gt = set()
    for d in detected:
        for gt in ground_truth:
            if abs(d - gt) <= tolerance and gt not in matched_gt:
                TP += 1
                matched_gt.add(gt)
```

```
                break
        else:
            FP += 1
    FN = len(ground_truth) - TP
    return TP, FP, FN, matched_gt


# ------------------------------------------------------------------------
# 3. 📊 Evaluate Peak Detection Metrics
# ------------------------------------------------------------------------
def evaluate_threshold_performance(refined_peaks, ground_truth_indices, filtered_data, tolerance=TOLERANCE, zoom=(0,
    """
    📊 Evaluate performance of threshold-based detection and show visual overlay.

    📥 Parameters:
    - refined_peaks: array of detected peak indices
    - ground_truth_indices: GT event indices (from CSV)
    - filtered_data: 1D array of signal trace (after filtering)
    - tolerance: matching window "Is this close enough to any GT event to count as a hit (TP)?"
    - zoom: (start, end) sample range for plot

    📤 Returns:
    - None, but prints precision/recall/f1 and plots results

    📌 Use to verify how well your thresholding captured true events.
    """
    TP, FP, FN, matched_gt = match_detected_to_ground_truth(refined_peaks, ground_truth_indices, tolerance)

    precision = TP / (TP + FP) if TP + FP > 0 else 0
    recall = TP / (TP + FN) if TP + FN > 0 else 0
    f1 = 2 * (precision * recall) / (precision + recall) if precision + recall > 0 else 0

    print(f"===== METRICS FOR THRESHOLDING ONLY =====")
    print(f"    ↪ Threshold Parameters: {params}")
    print(f"\nTotal GT Events: {len(ground_truth_indices)} | Total Detected: {len(refined_peaks)}\n")

    print(f"True Positives: {TP}")
    print(f"False Positives: {FP}")
    print(f"False Negatives: {FN}")
    print(f"Precision: {precision:.2f}")
    print(f"Recall: {recall:.2f}")
    print(f"F1 Score: {f1:.2f}")
```

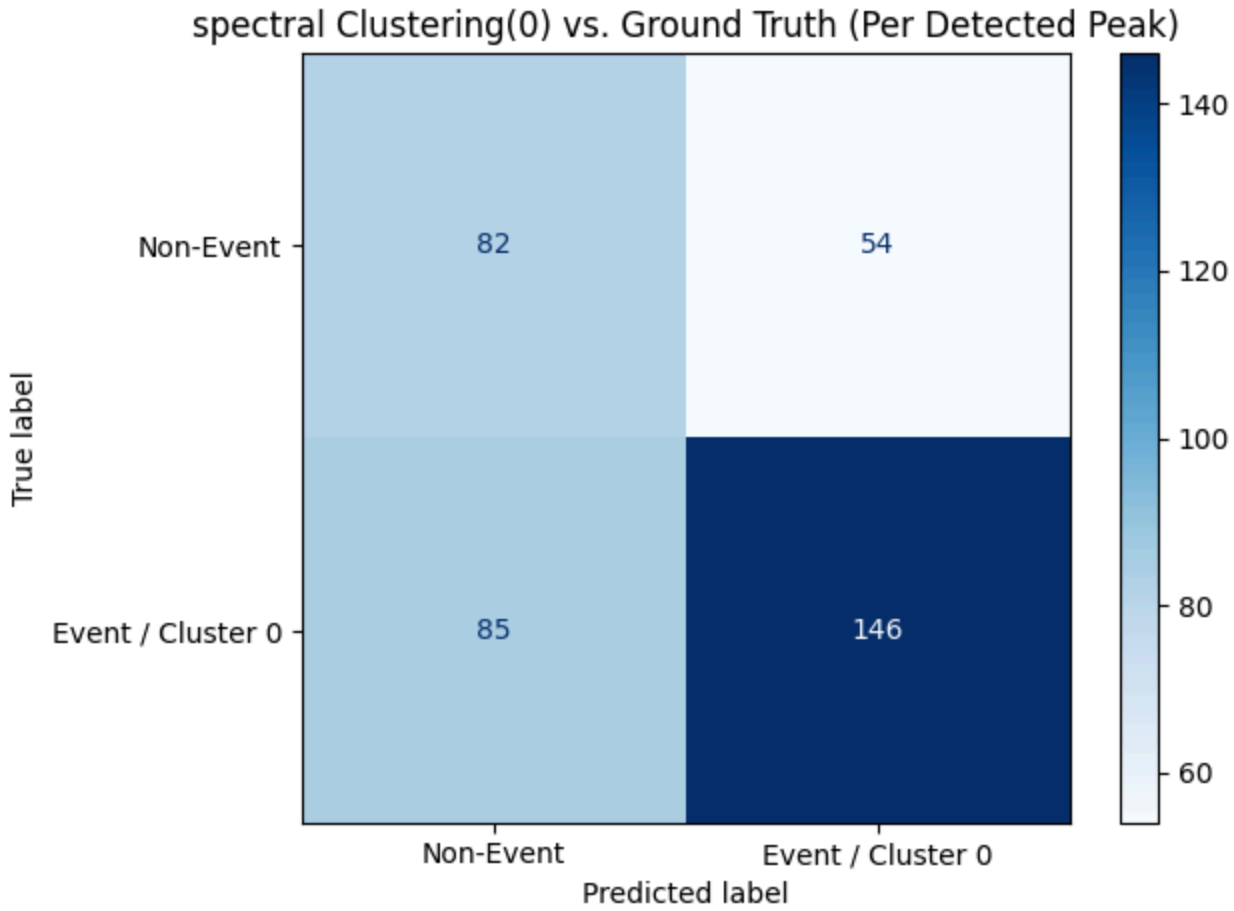
```
    print(f"\n🚫 Undetected GT Events: {len(ground_truth_indices) - TP}")
    print(f"✅ Max # of Cluster True: {len(matched_gt)}")
    print(f"📦 GT-Thresh Matching Tolerance: {TOLERANCE} samples")


    '''
    # Plot overlay
    zoom_start, zoom_end = zoom
    gt_in_view = [gt for gt in ground_truth_indices if zoom_start <= gt < zoom_end]
    det_in_view = [p for p in refined_peaks if zoom_start <= p < zoom_end]

    plt.figure(figsize=(14, 5))
    plt.plot(np.arange(zoom_start, zoom_end), filtered_data[zoom_start:zoom_end], color='black', label="Filtered Trac
    plt.scatter(gt_in_view, filtered_data[gt_in_view], color='green', s=50, label="Ground Truth")
    plt.scatter(det_in_view, filtered_data[det_in_view], color='red', s=30, alpha=0.6, label="Detected")
    plt.title("Ground Truth vs Refined Peaks (Zoomed View)")
    plt.xlabel("Sample Index")
    plt.ylabel("Amplitude (pA)")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()
    '''


# ------------------------------------------------------------------------
# 4. 🤖 Evaluate Clustering Results vs. GT
# ------------------------------------------------------------------------
def evaluate_cluster_against_true_positive(
    cluster_labels, refined_peaks, ground_truth_indices,
    filtered_data, true_cluster, tolerance,
    show_preview=True, max_preview=30
):
    """
    🤖 Evaluates how well a given cluster label aligns with true events.

    🎛️ Parameters:
    - cluster_labels: cluster assigned to each refined peak (e.g., from KMeans)
    - refined_peaks: list of detected/refined event indices
    - ground_truth_indices: GT event indices
    - filtered_data: trace data used to extract amplitudes
    - true_cluster: the label corresponding to true events (usually 0 or 1)
```

```
- tolerance: allowed window for matching GT events "Did this cluster explain any ground truth events?"
- show_preview: whether to display match summary
- max_preview: number of matches to preview

📤 Returns:
- df_preview: summary DataFrame of matched peaks (if show_preview=True)

📌 Use to determine whether clustering found the correct event group.
"""
print("=================================================")
print(f"📊 Clustering Evaluation (Cluster {true_cluster} as 'True Event')")
print(f"🧪 Cluster Method: {use_clustering_method}\n")

binary_truth = np.zeros(len(refined_peaks), dtype=int)
binary_preds = np.array([1 if lbl == true_cluster else 0 for lbl in cluster_labels])

for gt in ground_truth_indices:
    for i, peak in enumerate(refined_peaks):
        if abs(peak - gt) <= tolerance:
            binary_truth[i] = 1
            break

precision = precision_score(binary_truth, binary_preds, zero_division=0)
recall = recall_score(binary_truth, binary_preds, zero_division=0)
f1 = f1_score(binary_truth, binary_preds, zero_division=0)

print(f"• Precision: {precision:.3f}")
print(f"• Recall:    {recall:.3f}")
print(f"• F1 Score:  {f1:.3f}\n")

cm = confusion_matrix(binary_truth, binary_preds)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=["Non-Event", f"Event / Cluster {true_cluster}"
plt.figure(figsize=(6, 5))
disp.plot(cmap="Blues", values_format="d")
plt.title(f"{use_clustering_method} Clustering({true_cluster}) vs. Ground Truth (Per Detected Peak)")
plt.tight_layout()
plt.grid(False)
plt.show()

# Append hard summary match preview
total_gt = len(ground_truth_indices) # Count total number of ground truth events
total_detected = len(refined_peaks) # Count total number of detected peaks (refined from thresholding)
```

```
# Identify which ground truth events were matched by a peak from the correct cluster
# For each GT index, we check if there's any peak (with the correct cluster label)
# that falls within the allowed tolerance window
matched_gt_indices = {gt for gt in ground_truth_indices for peak, label in zip(refined_peaks, cluster_labels) # 
                      if abs(peak - gt) <= tolerance and label == true_cluster}
undetected_count = total_gt - len(matched_gt_indices)


print(f"📂 Analyzed File: {base_filename}")
#print(f"📦 Total GT Events: {total_gt} | Total Detected Peaks: {total_detected}\n")

TN = cm[0, 0]
FP = cm[0, 1]
FN = cm[1, 0]
TP = cm[1, 1]

print(f"  Confusion Matrix Breakdown:")
print(f"    - True Positives (TP): {TP}")
print(f"    - False Positives (FP): {FP}")
print(f"    - True Negatives (TN): {TN}")
print(f"    - False Negatives (FN): {FN}")
print()
print(f"  Summary:")
print(f"    ✅ Matched GT Events: {TP} (i.e., GT events detected by cluster {true_cluster})")
print(f"    🚫 Missed GT Events: {FN} (GT events not captured by this cluster)")
print(f"    📦 Matching GT-Cluster Tolerance: {tolerance} samples")



if show_preview:
    matches = []
    last_peak = None
    for gt in ground_truth_indices:
        for i, peak in enumerate(refined_peaks):
            if abs(peak - gt) <= tolerance and cluster_labels[i] == true_cluster:
                delta = abs(peak - gt)
                amp = float(filtered_data[peak])
                iei = (peak - last_peak) if last_peak is not None else np.nan
                matches.append((gt, peak, cluster_labels[i], delta, amp, iei, iei / SAMPLING_RATE if last_peak e
                last_peak = peak
                break
```

```
        df_preview = pd.DataFrame(matches, columns=[
            "GT Index", "Peak Index", "Cluster", "Δ (samples)",
            "Amplitude (pA)", "IEI (samples)", "IEI (seconds)"
        ])
        print(df_preview.head(max_preview).to_string(index=False))
        if len(df_preview) > max_preview:
            print(f"...and {len(df_preview) - max_preview} more matches.\n")
        print("================================================")
        return df_preview

    return None
#####################################################

hand_count_data, ground_truth_indices = load_ground_truth(
    csv_filepath1,
    column='time of peak',
    sampling_rate=SAMPLING_RATE
)

# -------------------------------
# 🔧 Global Parameters
# -------------------------------
TOLERANCE = 60  # Samples
use_clustering_method = "spectral"
TRUE_CLUSTER = 0       # "Correct" cluster label

# ---------------------------------------------

evaluate_threshold_performance(refined_peaks=refined_peaks,ground_truth_indices=ground_truth_indices,filtered_data=fi
    tolerance=TOLERANCE, # defines the window around each GT index in which a detection (peak) counts as a "match".
    zoom=(0, 2_000_000)
)

evaluate_cluster_against_true_positive(cluster_labels=clustering_results[use_clustering_method],refined_peaks=refined
    tolerance=TOLERANCE, # defines whether any clustered peak (with the correct label) falls within ±tolerance of a G
    show_preview=False,
    max_preview=30,
    true_cluster=TRUE_CLUSTER
)
```

```
⚠️ Detected units in milliseconds. Converting to seconds...
📦 Loaded 300 ground truth events
===== METRICS FOR THRESHOLDING ONLY =====
    ↪ Threshold Parameters: {'delta': 13, 'slope_thresh': 0.35, 'distance': 400}

Total GT Events: 300 | Total Detected: 367

True Positives: 231
False Positives: 136
False Negatives: 69
Precision: 0.63
Recall: 0.77
F1 Score: 0.69

🚫 Undetected GT Events: 69
✅ Max # of Cluster True: 231
📦 GT-Thresh Matching Tolerance: 60 samples
==================================================
📊 Clustering Evaluation (Cluster 0 as 'True Event')
🌌 Cluster Method: spectral

• Precision: 0.730
• Recall:    0.632
• F1 Score:  0.677

<Figure size 600x500 with 0 Axes>
```



📂 Analyzed File: 2024_01_22_0009_Trunc_ Cell 1 IPSC.abf
Confusion Matrix Breakdown:
- True Positives (TP): 146
- False Positives (FP): 54
- True Negatives (TN): 82
- False Negatives (FN): 85

Summary:
✅ Matched GT Events: 146 (i.e., GT events detected by cluster 0)
🚫 Missed GT Events: 85 (GT events not captured by this cluster)
📦 Matching GT-Cluster Tolerance: 60 samples

📊 Clustering Evaluation (Cluster X as 'True Event') • Precision: % of events labeled by the cluster that were correct (Low FP → High Precision) • Recall: % of real ground truth events the cluster successfully captured (Low FN → High Recall) • F1 Score: Harmonic

average of Precision & Recall (balanced metric)

Confusion Matrix: → Rows = Ground Truth (actual) → Columns = Cluster Labels (predicted)

```
TN: True Negative  – correct non-event (rare if only evaluating peaks)
FP: False Positive – model says “event” but it's not (bad if high)
FN: False Negative – missed true events (bad if high)
TP: True Positive  – correctly detected events
```

🚫 Undetected GT Events: Total GT events that were missed entirely (not matched by clustering at all)

In [ ]:
```python
def summarize_clustering_results(
    clustering_results_dict,
    refined_peaks,
    ground_truth_indices,
    tolerance=60,
    manual_cluster_map=None
):
    """
    Evaluates all clustering methods and returns a transposed summary DataFrame.
    If manual_cluster_map is provided, it uses the specified cluster for each method.
    Otherwise, the largest cluster is selected (excluding noise).

    Parameters:
    - clustering_results_dict: dict of method_name -> cluster_labels
    - refined_peaks: indices of detected events
    - ground_truth_indices: known event indices
    - tolerance: matching window in samples
    - manual_cluster_map: dict (optional) with method_name -> cluster_id

    Returns:
    - Transposed DataFrame comparing metrics across clustering methods
    """
    manual_cluster_map = manual_cluster_map or {}
    summary = []

    for method_name, cluster_labels in clustering_results_dict.items():
        cluster_labels = np.array(cluster_labels)

        # Count labels excluding noise (-1)
```

```
        valid_labels = cluster_labels[cluster_labels >= 0]
        unique, counts = np.unique(valid_labels, return_counts=True)

        # Use manual cluster if provided
        if method_name in manual_cluster_map:
            true_cluster = manual_cluster_map[method_name]
            cluster_count = np.sum(cluster_labels == true_cluster)
            print(f"🔧 Using manually specified cluster {true_cluster} for {method_name} "
                  f"(Count: {cluster_count})")
        else:
            if len(valid_labels) == 0:
                print(f"⚠️ {method_name} produced only noise.")
                continue

            true_cluster = unique[np.argmax(counts)]
            cluster_count = counts[np.argmax(counts)]
            print(f"📊 Auto-selected cluster {true_cluster} for {method_name} "
                  f"(Largest non-noise cluster, Count: {cluster_count})")

        # Create binary labels
        binary_truth = np.zeros(len(refined_peaks), dtype=int)
        binary_preds = np.array([1 if lbl == true_cluster else 0 for lbl in cluster_labels])
        matched_gt = set()

        for i, peak in enumerate(refined_peaks):
            for gt in ground_truth_indices:
                if abs(peak - gt) <= tolerance:
                    binary_truth[i] = 1
                    matched_gt.add(gt)
                    break

        TP = np.sum((binary_truth == 1) & (binary_preds == 1))
        FP = np.sum((binary_truth == 0) & (binary_preds == 1))
        FN = np.sum((binary_truth == 1) & (binary_preds == 0))

        summary.append({
            "Method": f"{method_name} (Clust:{true_cluster})",
            "True Positives": TP,
            "False Positives": FP,
            "False Negatives": FN,
            "Precision": round(precision_score(binary_truth, binary_preds, zero_division=0), 3),
            "Recall": round(recall_score(binary_truth, binary_preds, zero_division=0), 3),
```

```
            "F1 Score": round(f1_score(binary_truth, binary_preds, zero_division=0), 3),
            "Matched GTs": len(matched_gt),
            "Missed GTs": len(ground_truth_indices) - len(matched_gt)
        })

    df_summary = pd.DataFrame(summary)
    df_transposed = df_summary.set_index("Method").T

    display(df_transposed.style.set_caption("🔎 Clustering Method Comparison (Transposed)"))
    return df_transposed



manual_cluster_map = {
    #"kmeans": TRUE_CLUSTER,
    #"hdbscan": 1,
    #'spectral': 0
}

df_summary = summarize_clustering_results(
    clustering_results_dict=clustering_results,
    refined_peaks=refined_peaks,
    ground_truth_indices=ground_truth_indices,
    tolerance=60,
    manual_cluster_map=manual_cluster_map,  # Added to include cluster category
)
print(f"\nPCA Explained Variance: {np.round(np.sum(pca_model.explained_variance_ratio_* 100))}%") # Explained Varianc
print_pipeline_params(results["params"]) # Pipeline Params
plot_cluster_maps_across_methods(filtered_data=filtered_data,peak_indices=refined_peaks,clustering_results=clustering
```

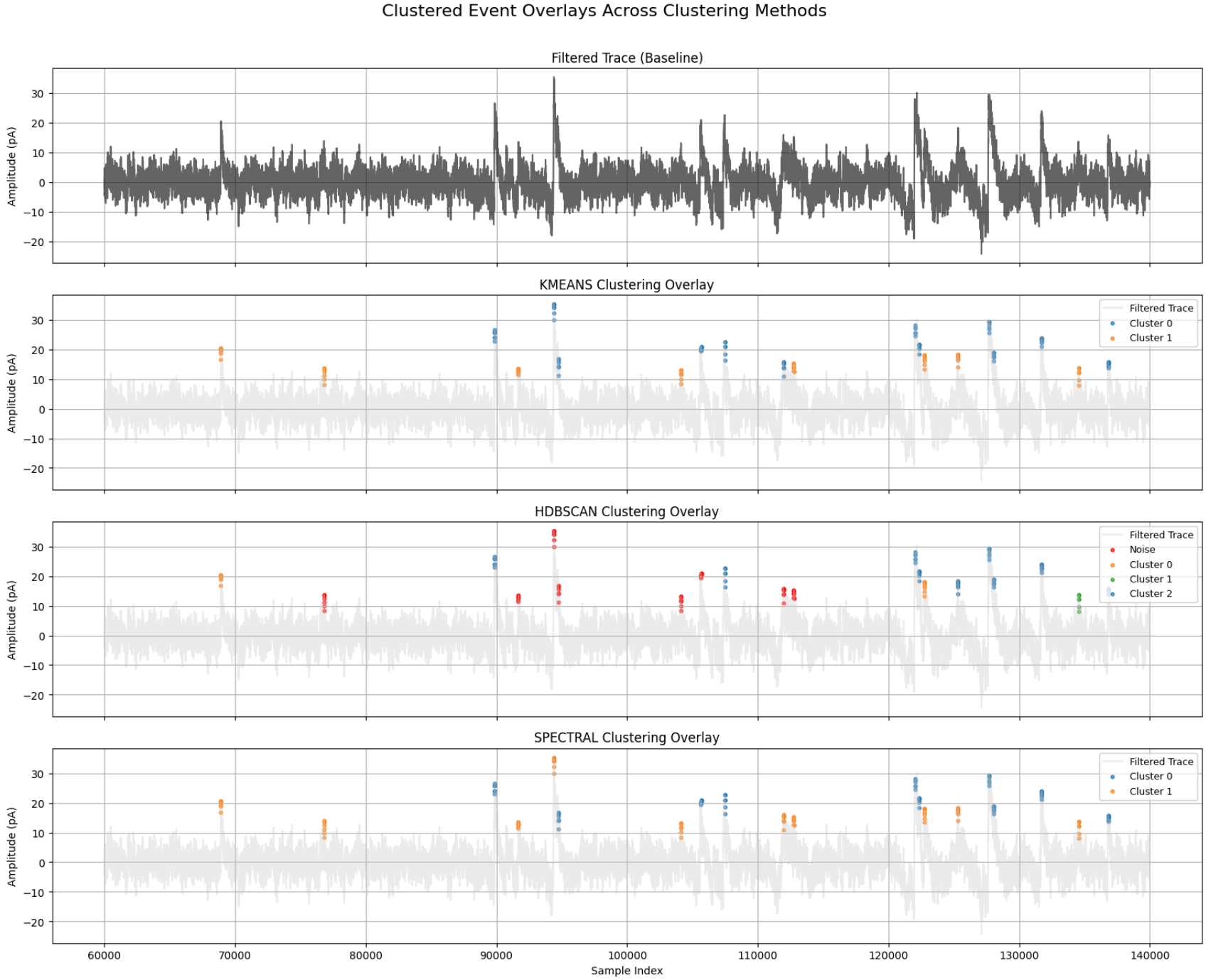
```
📊 Auto-selected cluster 0 for kmeans (Largest non-noise cluster, Count: 224)
📊 Auto-selected cluster 2 for hdbscan (Largest non-noise cluster, Count: 190)
📊 Auto-selected cluster 0 for spectral (Largest non-noise cluster, Count: 200)
```

🔎 Clustering Method Comparison (Transposed)

Method	kmeans (Clust:0)	hdbscan (Clust:2)	spectral (Clust:0)
True Positives	166.000000	134.000000	146.000000
False Positives	58.000000	56.000000	54.000000
False Negatives	65.000000	97.000000	85.000000
Precision	0.741000	0.705000	0.730000
Recall	0.719000	0.580000	0.632000
F1 Score	0.730000	0.637000	0.677000
Matched GTs	231.000000	231.000000	231.000000
Missed GTs	69.000000	69.000000	69.000000

```
PCA Explained Variance: 67.0%

====== ⚙️ Pipeline Parameter Summary ======
• Signal Length: 1975580
• Number of Peaks: 367
• Pre-samples: 40
• Post-samples: 60
• Wavelet Level: 3
• PCA Components: 2
========================================
```

```
<ipython-input-3-ad71f98e90b5>:573: MatplotlibDeprecationWarning: The get_cmap function was deprecated in Matplotlib
3.7 and will be removed in 3.11. Use ``matplotlib.colormaps[name]`` or ``matplotlib.colormaps.get_cmap()`` or ``pyplo
t.get_cmap()`` instead.
  cmap = get_cmap(cmap_name)
```


Clustered Event Overlays Across Clustering Methods
Filtered Trace (Baseline)
KMEANS Clustering Overlay
HDBSCAN Clustering Overlay
SPECTRAL Clustering Overlay
Amplitude (pA)
Sample Index
Filtered Trace
Noise
Cluster 0
Cluster 1
Cluster 2

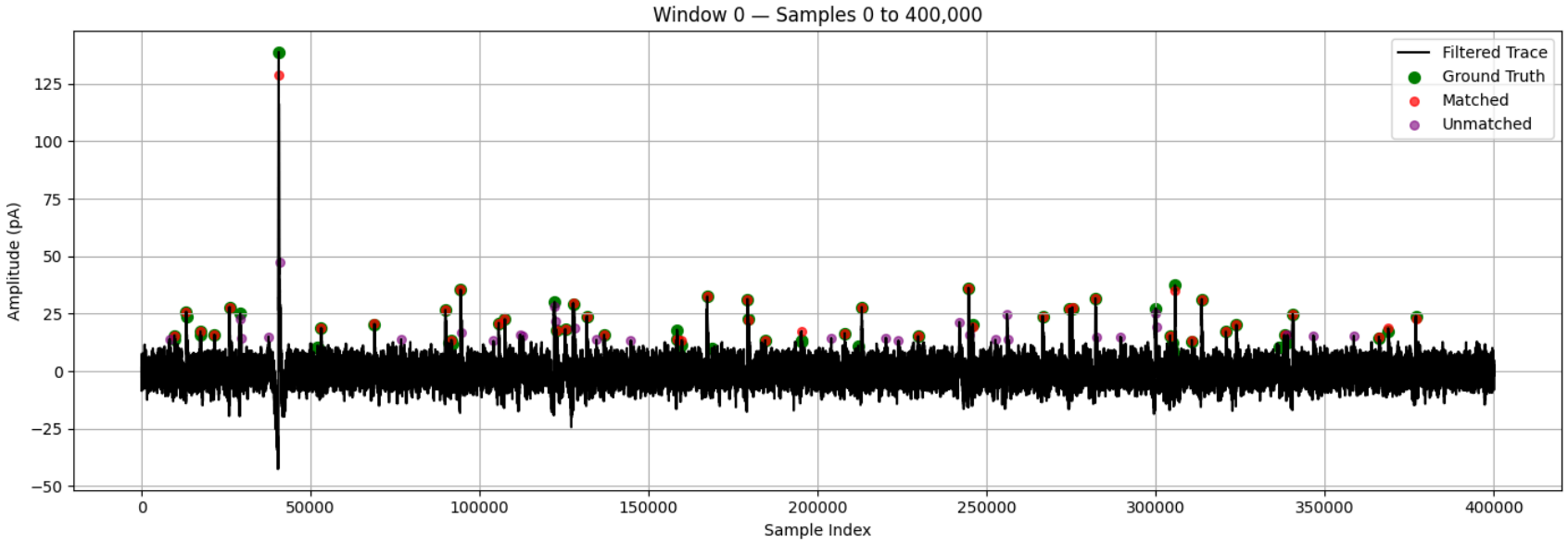
In [ ]:
```
# ================================================
# 🔎 Threshold Detector Evaluation + Visualization
# ================================================

import numpy as np
import matplotlib.pyplot as plt

# -------------------------------
# ⚙️ Config Parameters
# -------------------------------
WINDOW_SIZE = 400_000     # Samples per plot window

# -------------------------------
# 1. Match Detected to Ground Truth
# -------------------------------
def match_detected_to_ground_truth(detected, ground_truth, tolerance=TOLERANCE):
    """
    Match detected peaks to ground truth events using a tolerance window.

    Parameters:
    - detected (array): Indices of detected events
    - ground_truth (array): Indices of ground truth events
    - tolerance (int): Max allowable distance between a detected and true event

    Returns:
    - TP (int): True Positives (matched detections)
    - FP (int): False Positives (unmatched detections)
    - FN (int): False Negatives (unmatched GTs)
    - unmatched_gt (set): Set of GT events that were not matched
    """
    TP, FP = 0, 0
    matched_gt = set()

    for d in detected:
        matched = False
        for gt in ground_truth:
            if abs(d - gt) <= tolerance and gt not in matched_gt:
                matched_gt.add(gt)
                TP += 1
                matched = True
                break
```

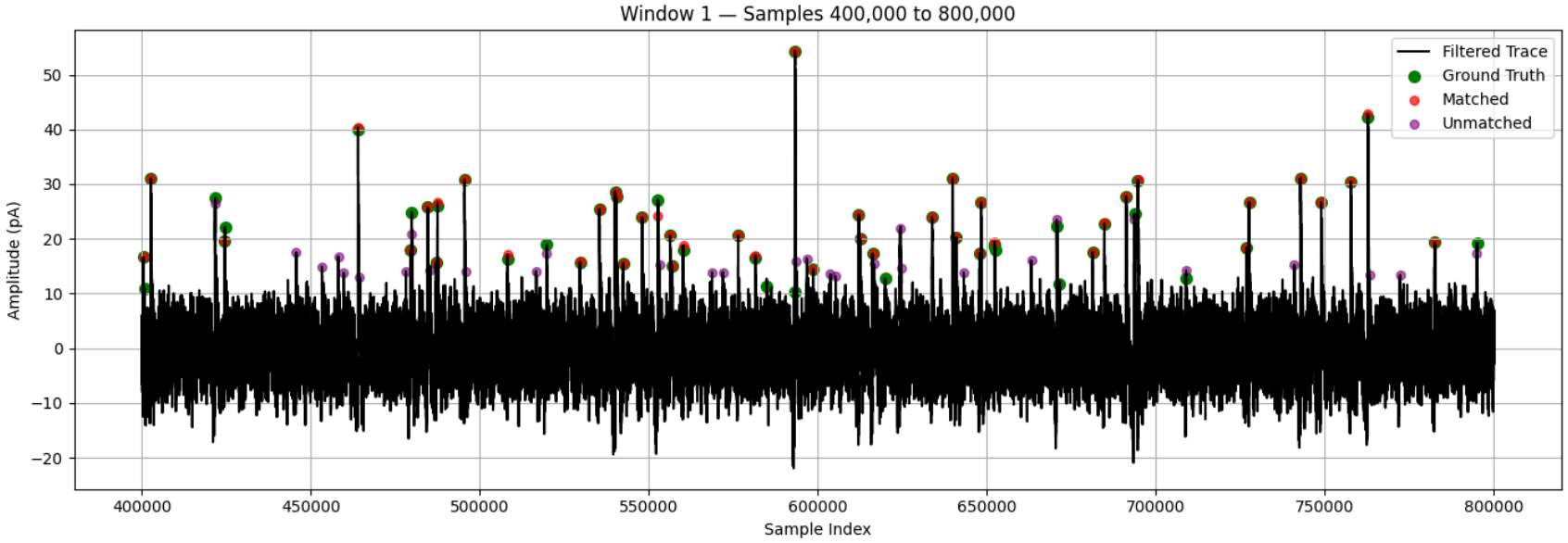
```
        if not matched:
            FP += 1

    unmatched_gt = set(ground_truth) - matched_gt
    FN = len(unmatched_gt)
    return TP, FP, FN, unmatched_gt


# --------------------------------
# 2. Run Evaluation
# --------------------------------
TP, FP, FN, unmatched_gt = match_detected_to_ground_truth(
    refined_peaks, ground_truth_indices, tolerance=TOLERANCE
)

precision = TP / (TP + FP) if (TP + FP) > 0 else 0
recall = TP / (TP + FN) if (TP + FN) > 0 else 0
f1 = 2 * (precision * recall) / (precision + recall) if (precision + recall) > 0 else 0

# --------------------------------
# 📝 Print Summary
# --------------------------------
print("===== METRICS FOR THRESHOLDING ONLY =====")
print(f"✔️  True Positives: {TP}")
print(f"❌ False Positives: {FP}")
print(f"🚫 False Negatives: {FN}")
print(f"📊 Precision: {precision:.2f}")
print(f"📈 Recall:    {recall:.2f}")
print(f"🧠 F1 Score:  {f1:.2f}")
print(f"\n📦 Total GT Events: {len(ground_truth_indices)}")
print(f"💉 Total Detected Events: {len(refined_peaks)}")
print(f"🕳️  Unmatched GT Events: {len(unmatched_gt)}")


# --------------------------------
# 3. Visualize Matched / Unmatched Over Trace
# --------------------------------
def loop_matched_vs_unmatched_windows(
    filtered_data,
    refined_peaks,
    ground_truth_indices,
    tolerance=TOLERANCE,
```

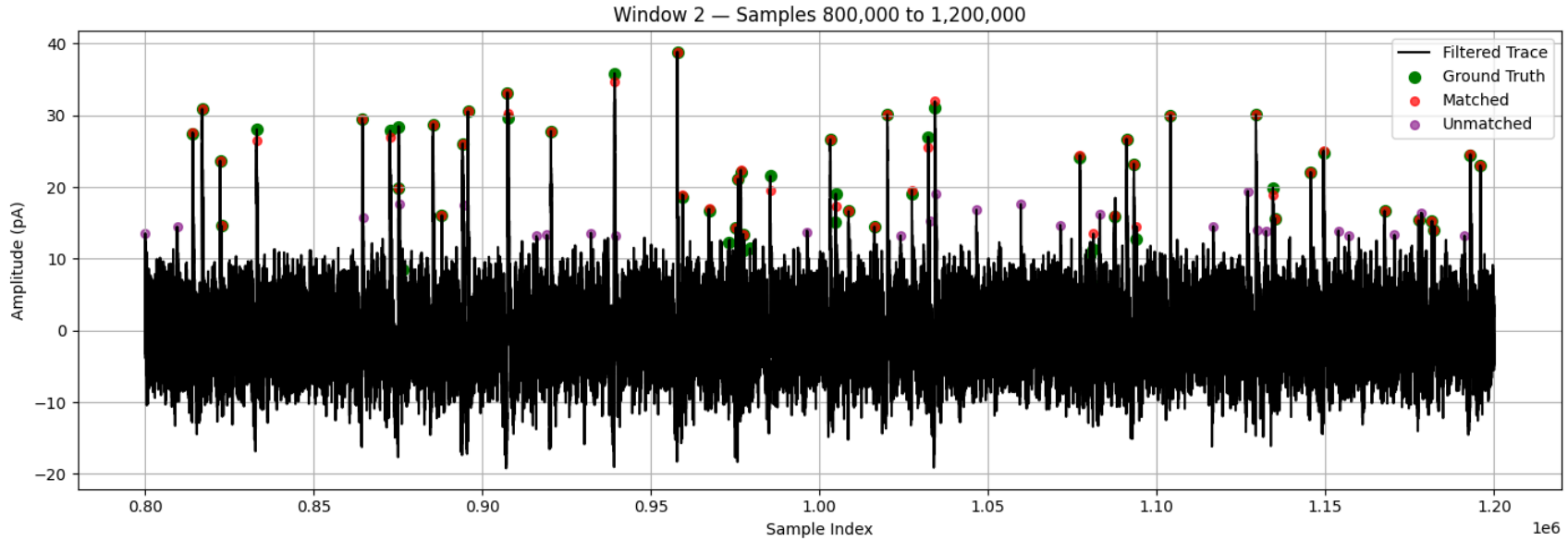
```
    window_size=WINDOW_SIZE,
    start_window=0,
    end_window=None
):
    """
    Iterate over signal in windows and visualize:
    - Ground truth events (green)
    - Matched detected peaks (red)
    - Unmatched detected peaks (purple)
    """
    signal_length = len(filtered_data)
    end_window = end_window or (signal_length // window_size)

    # Flag peaks as matched
    matched_flags = np.array([
        any(abs(p - gt) <= tolerance for gt in ground_truth_indices)
        for p in refined_peaks
    ])
    matched_peaks = refined_peaks[matched_flags]
    unmatched_peaks = refined_peaks[~matched_flags]

    for win in range(start_window, end_window):
        start = win * window_size
        end = min(start + window_size, signal_length)

        plt.figure(figsize=(14, 5))
        plt.plot(np.arange(start, end), filtered_data[start:end], color='black', label="Filtered Trace")

        # Ground Truth Events
        gt_in_view = [gt for gt in ground_truth_indices if start <= gt < end]
        plt.scatter(gt_in_view, filtered_data[gt_in_view], color='green', s=50, label="Ground Truth")

        # Matched Detections
        matched_in_view = [p for p in matched_peaks if start <= p < end]
        plt.scatter(matched_in_view, filtered_data[matched_in_view], color='red', s=30, alpha=0.7, label="Matched")

        # Unmatched Detections
        unmatched_in_view = [p for p in unmatched_peaks if start <= p < end]
        plt.scatter(unmatched_in_view, filtered_data[unmatched_in_view], color='purple', s=30, alpha=0.6, label="Unma

        plt.title(f"Window {win} — Samples {start:,} to {end:,}")
        plt.xlabel("Sample Index")
```

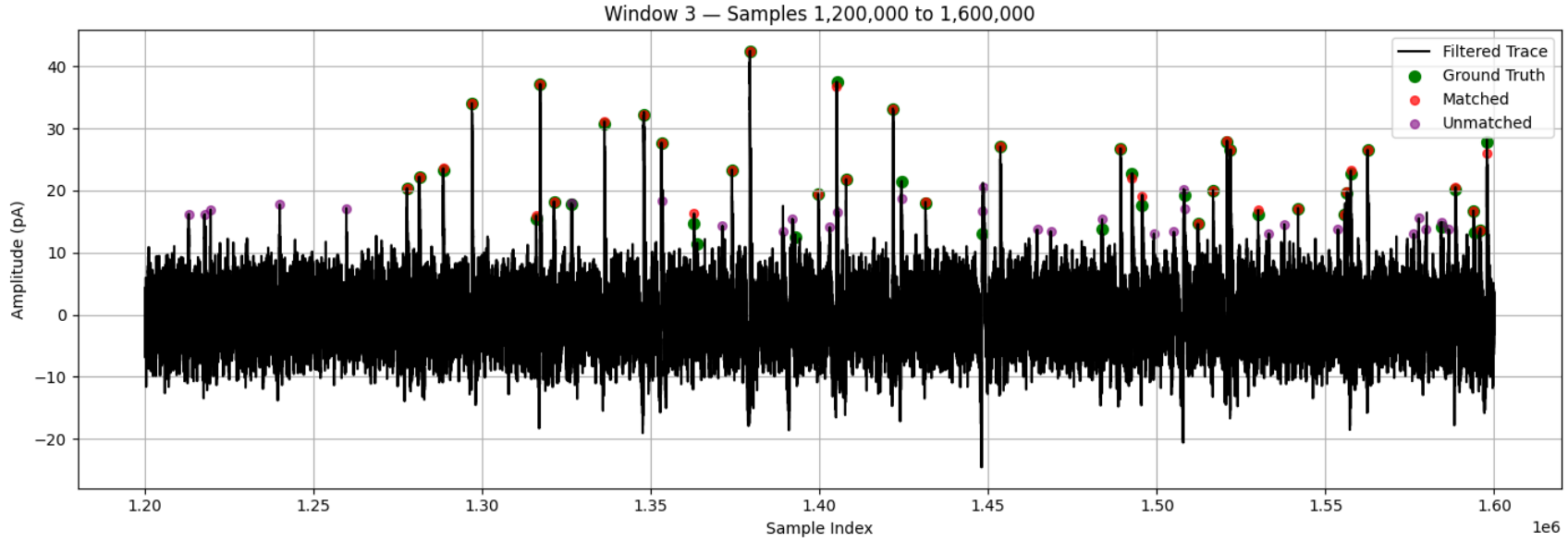
```
        plt.ylabel("Amplitude (pA)")

        handles, labels = plt.gca().get_legend_handles_labels()
        by_label = dict(zip(labels, handles))
        plt.legend(by_label.values(), by_label.keys())

        plt.grid(True)
        plt.tight_layout()
        plt.show()


# --------------------------------
# 4. Run Visualization
# --------------------------------
signal_length = len(filtered_data)
max_windows = signal_length // WINDOW_SIZE

loop_matched_vs_unmatched_windows(
    filtered_data=filtered_data,
    refined_peaks=refined_peaks,
    ground_truth_indices=ground_truth_indices,
    tolerance=TOLERANCE,
    window_size=WINDOW_SIZE,
    start_window=0,
    end_window=max_windows
)
```

```
===== METRICS FOR THRESHOLDING ONLY =====
✔️  True Positives: 231
❌ False Positives: 136
🚫 False Negatives: 69
📊 Precision: 0.63
📈 Recall:    0.77
🧠 F1 Score:  0.69

📦 Total GT Events: 300
🧪 Total Detected Events: 367
🕳️  Unmatched GT Events: 69
```


Window 0 — Samples 0 to 400,000
Filtered Trace
Ground Truth
Matched
Unmatched
Amplitude (pA)
Sample Index


Window 1 — Samples 400,000 to 800,000
Filtered Trace
Ground Truth
Matched
Unmatched
Amplitude (pA)
Sample Index





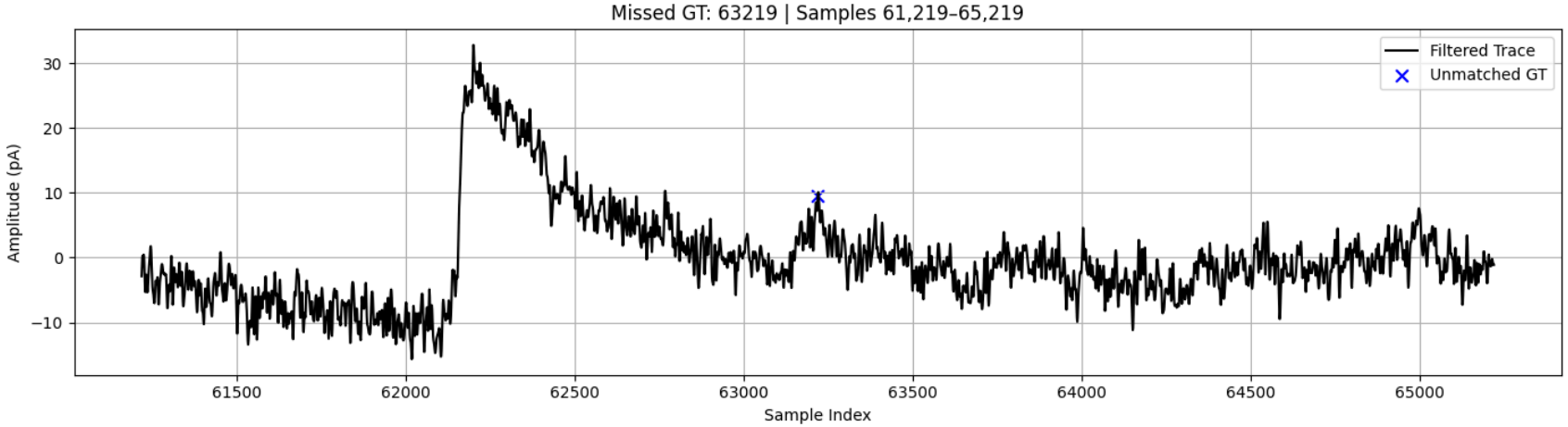
In [ ]:
```
# ========================================================================
# 🔍 Missed GT Viewer — Shows missed GT and (optional) misclassified peaks
# ========================================================================

def plot_missed_gt_events(
```

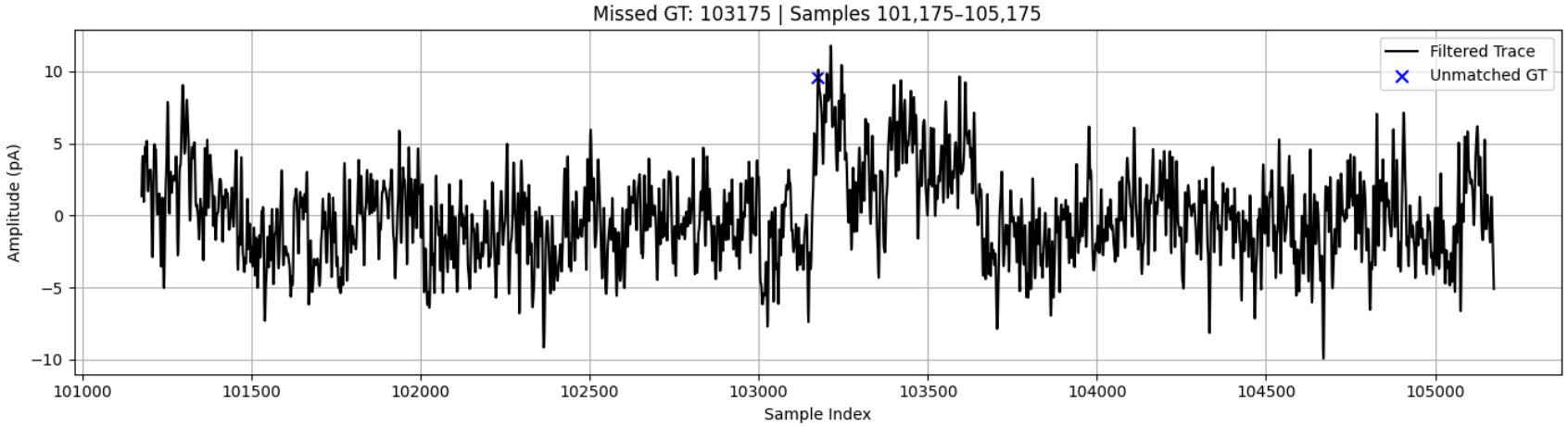
```
    refined_peaks,
    ground_truth_indices,
    filtered_data,
    cluster_labels=None,
    true_cluster=0,
    tolerance=60,
    window_radius=2000,
    show_overlay=True
):
    """
    Plot missed GT events and optionally show misclassified matches nearby.

    Parameters:
    - refined_peaks: list or array of detected peak indices
    - ground_truth_indices: list or array of ground truth sample indices
    - filtered_data: 1D array of the voltage trace
    - cluster_labels: optional list of cluster labels for each refined peak
    - true_cluster: cluster label assumed to be "correct"
    - tolerance: sample tolerance for peak-GT matching
    - window_radius: window size around GT event for plotting
    - show_overlay: if True, shows misclassified matched peaks (not in true_cluster)
    """

    unmatched_gt = []
    matched_flags = np.zeros(len(refined_peaks), dtype=bool)

    # Match peaks to GTs
    for gt in ground_truth_indices:
        found = False
        for i, peak in enumerate(refined_peaks):
            if abs(peak - gt) <= tolerance:
                matched_flags[i] = True
                found = True
                break
        if not found:
            unmatched_gt.append(gt)

    # Get unmatched peaks for overlay if needed
    misassigned_peaks = []
    if show_overlay and cluster_labels is not None:
        for i, peak in enumerate(refined_peaks):
            if matched_flags[i] and cluster_labels[i] != true_cluster:
```

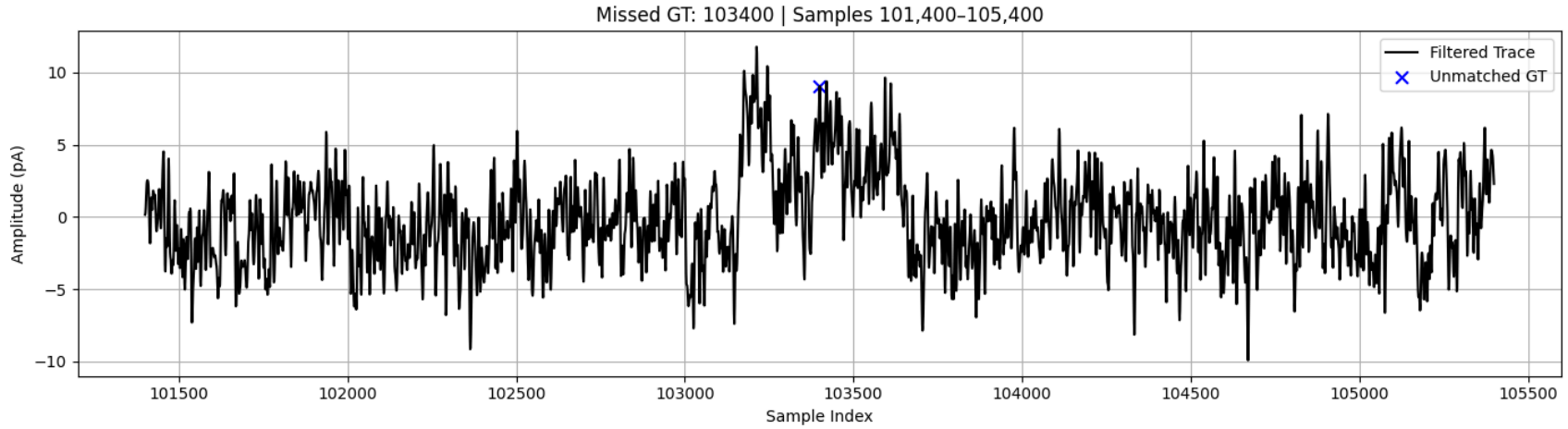
```
                misassigned_peaks.append(peak)

    print(f"🔍 Plotting {len(unmatched_gt)} unmatched GT events...")

    # Plot each unmatched GT event
    for gt in unmatched_gt:
        start = max(0, gt - window_radius)
        end = min(len(filtered_data), gt + window_radius)

        plt.figure(figsize=(14, 4))
        plt.plot(np.arange(start, end), filtered_data[start:end], color='black', label="Filtered Trace")

        # Mark missed GT with blue X
        if start <= gt < end:
            plt.scatter(gt, filtered_data[gt], color='blue', s=60, marker='x', label="Unmatched GT")

        # Optional: Overlay misassigned matched peaks
        if show_overlay:
            for peak in misassigned_peaks:
                if start <= peak < end:
                    plt.scatter(peak, filtered_data[peak], color='purple', s=40, alpha=0.8, label="Wrong Cluster")

        # Legend cleanup
        handles, labels = plt.gca().get_legend_handles_labels()
        by_label = dict(zip(labels, handles))
        plt.legend(by_label.values(), by_label.keys())

        plt.title(f"Missed GT: {gt} | Samples {start:,}–{end:,}")
        plt.xlabel("Sample Index")
        plt.ylabel("Amplitude (pA)")
        plt.grid(True)
        plt.tight_layout()
        plt.show()

plot_missed_gt_events(
    refined_peaks=refined_peaks,
    ground_truth_indices=ground_truth_indices,
    filtered_data=filtered_data,
    cluster_labels=clustering_results[use_clustering_method],  # or None
    true_cluster=0,
    tolerance=60,
    window_radius=2000,
```

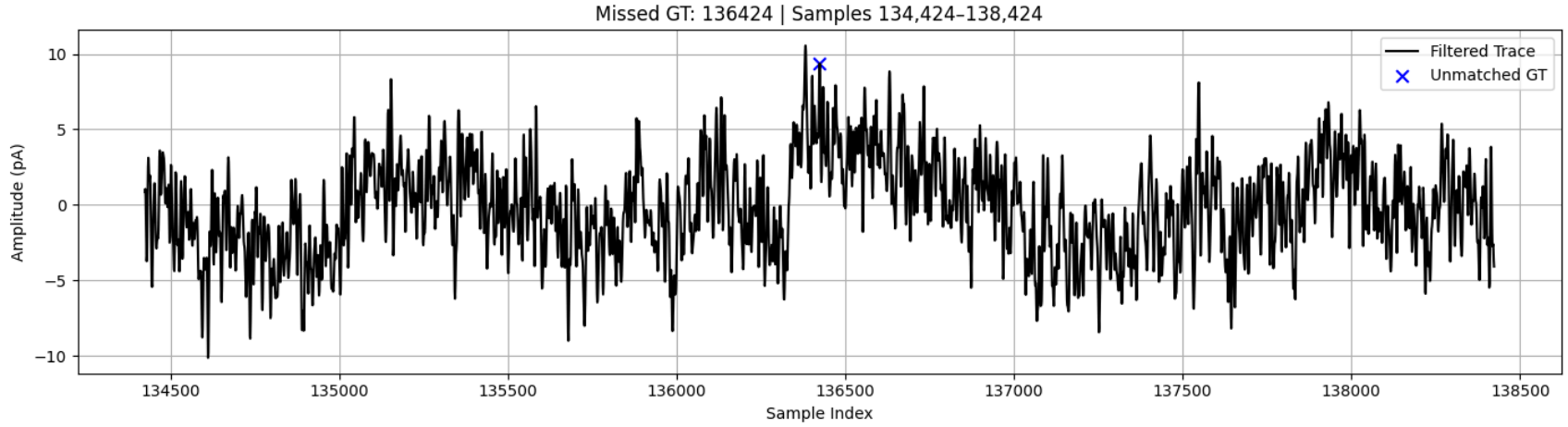
```
    show_overlay=True  # 🔄 Toggle this on or off
)
```

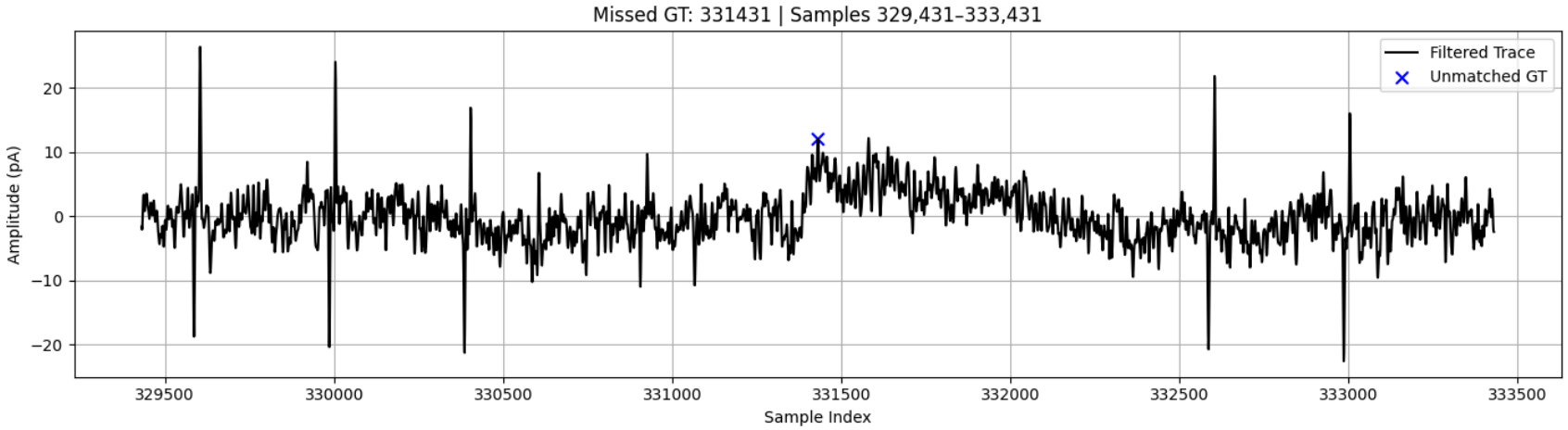
🔍 Plotting 75 unmatched GT events...

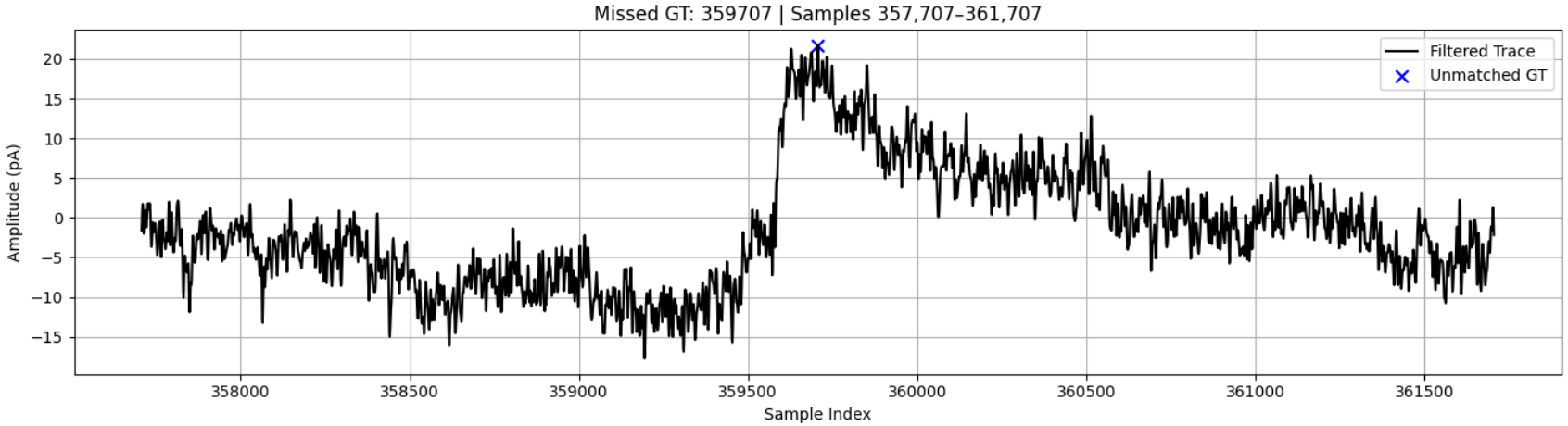


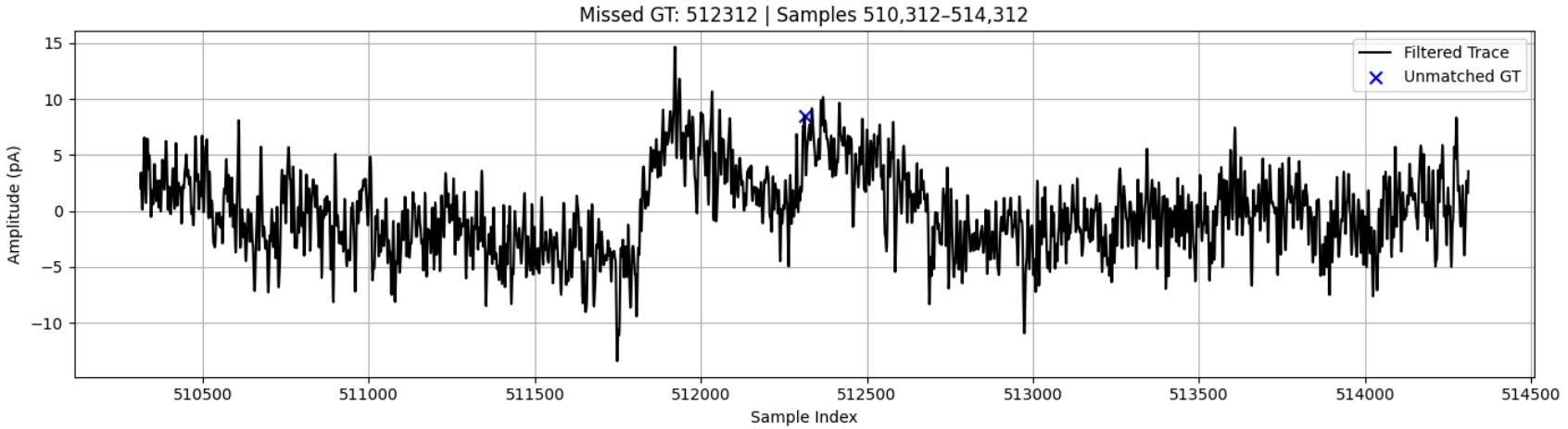


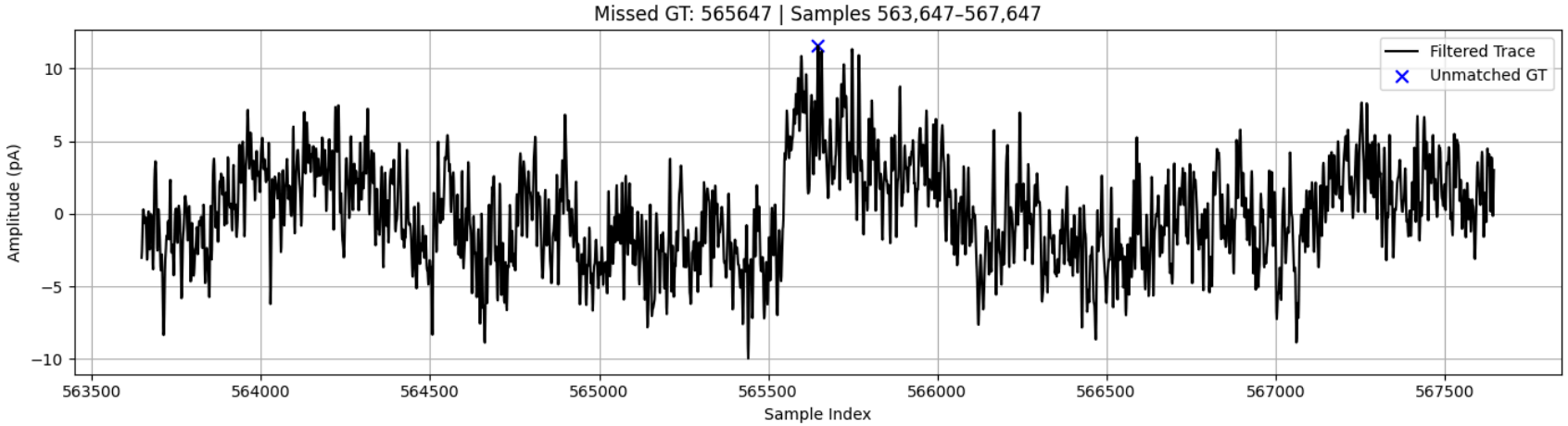

Missed GT: 103400 | Samples 101,400–105,400
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
10
5
0
−5
−10
101500
102000
102500
103000
103500
104000
104500
105000
105500

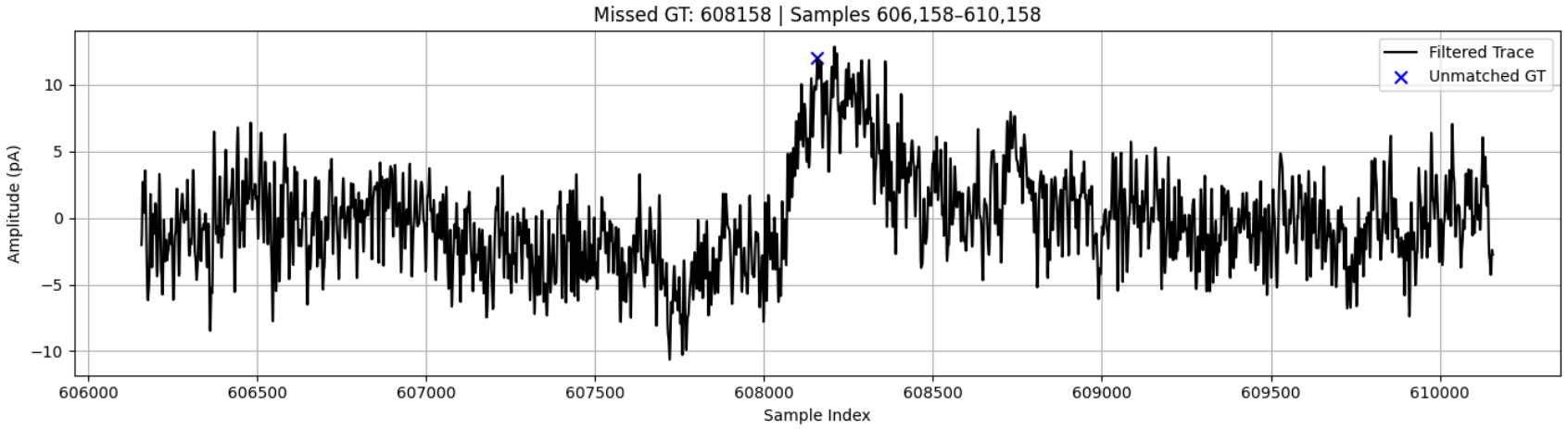

Missed GT: 136424 | Samples 134,424–138,424
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
10
5
0
−5
−10
134500
135000
135500
136000
136500
137000
137500
138000
138500

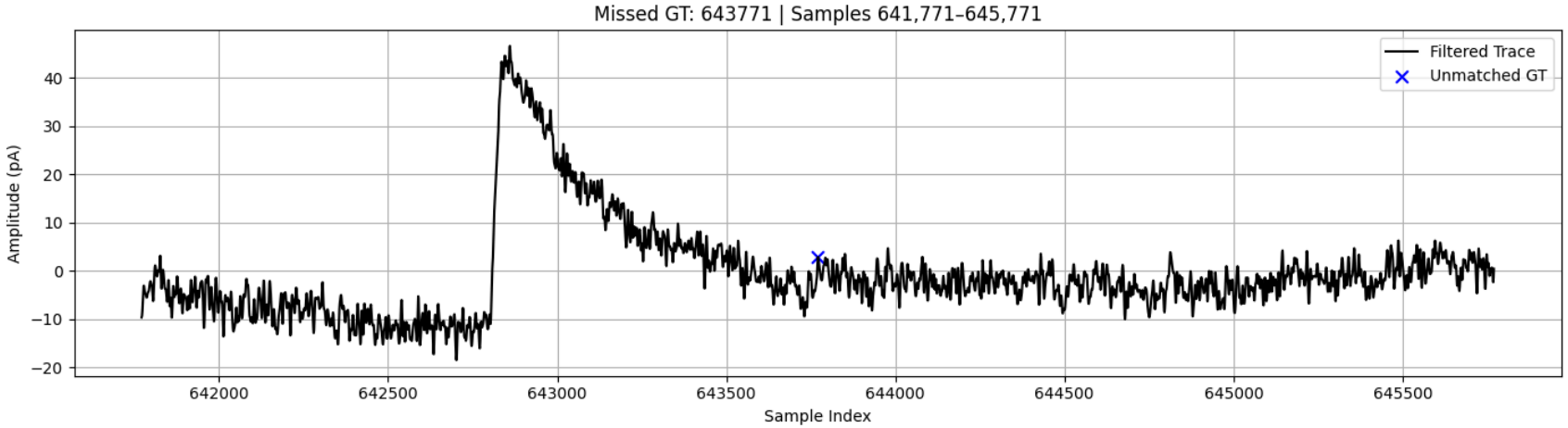

Missed GT: 331431 | Samples 329,431–333,431
Filtered Trace
Unmatched GT
Amplitude (pA)
20
10
0
−10
−20
329500
330000
330500
331000
331500
332000
332500
333000
333500
Sample Index

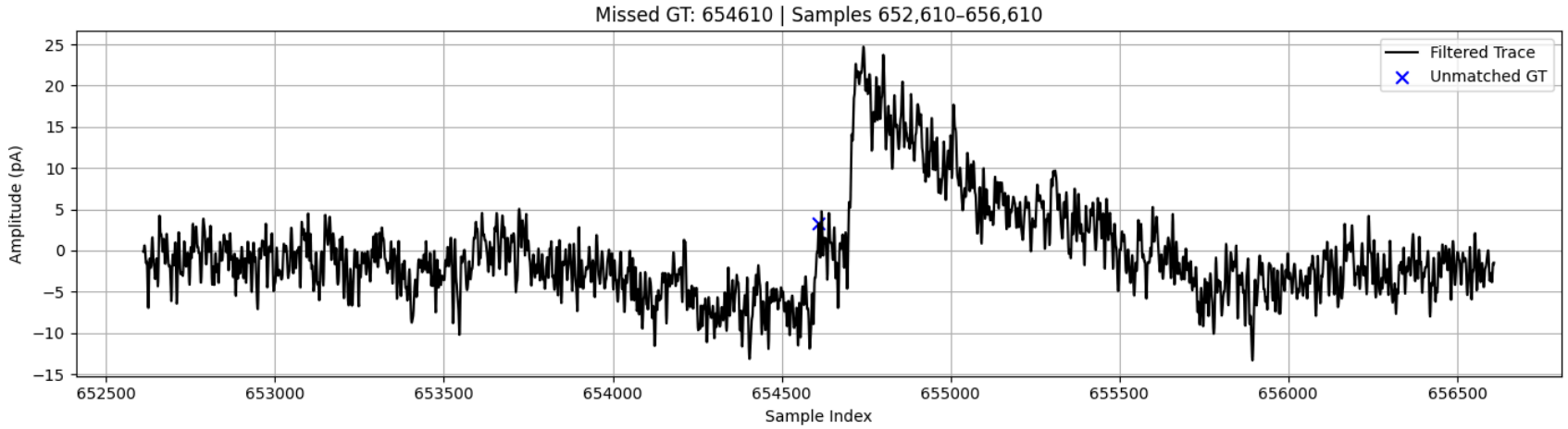

Missed GT: 359707 | Samples 357,707–361,707
Filtered Trace
Unmatched GT
Amplitude (pA)
20
15
10
5
0
−5
−10
−15
358000
358500
359000
359500
360000
360500
361000
361500
Sample Index

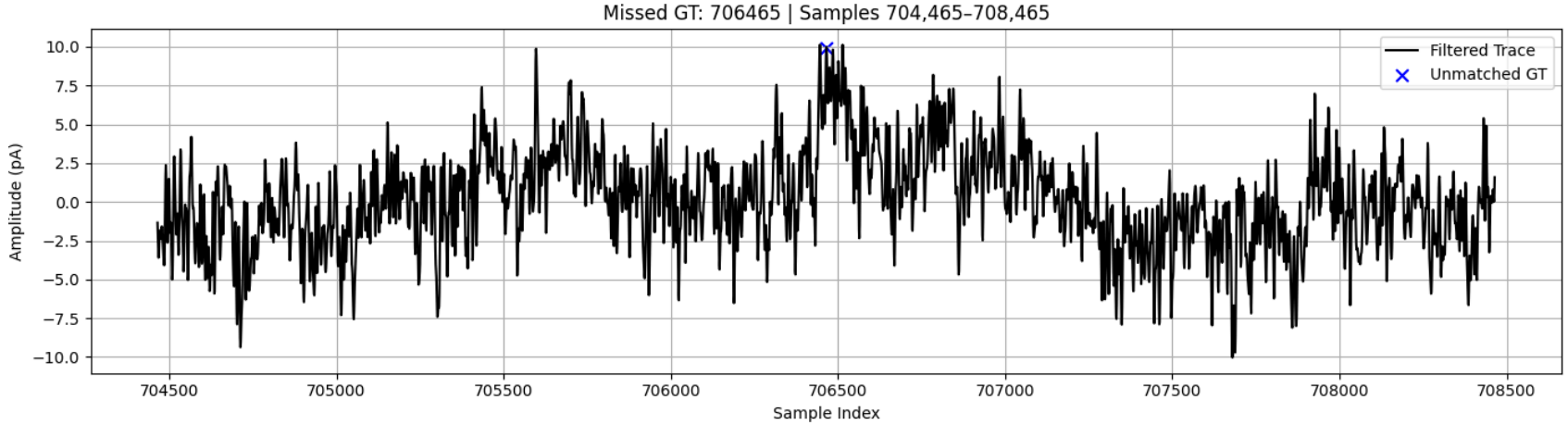

Missed GT: 512312 | Samples 510,312–514,312
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
15
10
5
0
−5
−10
510500
511000
511500
512000
512500
513000
513500
514000
514500

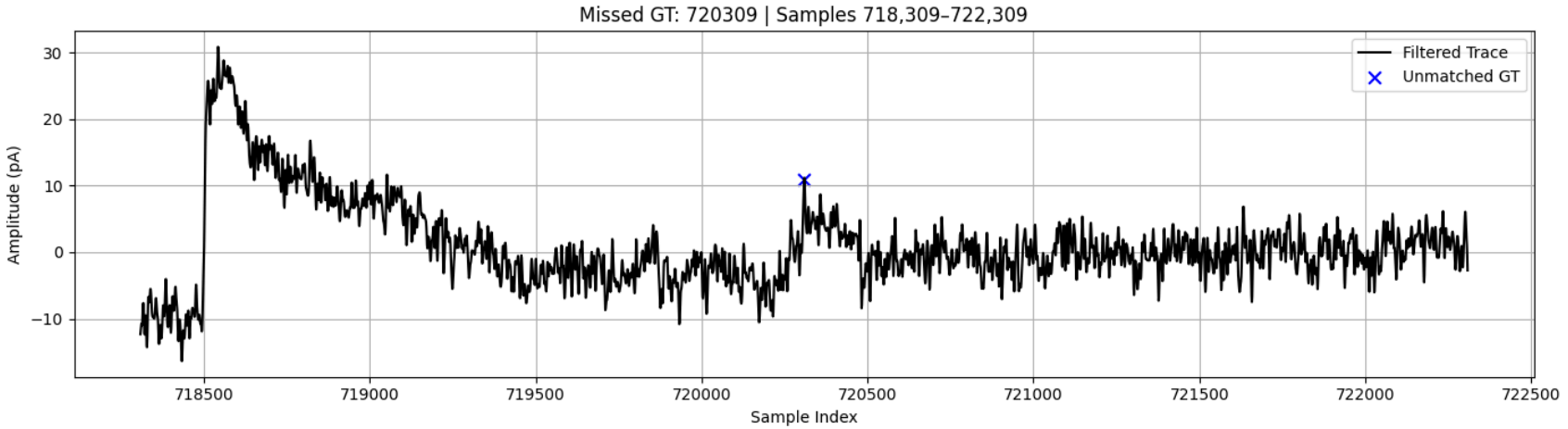

Missed GT: 565647 | Samples 563,647–567,647
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
10
5
0
−5
−10
563500
564000
564500
565000
565500
566000
566500
567000
567500

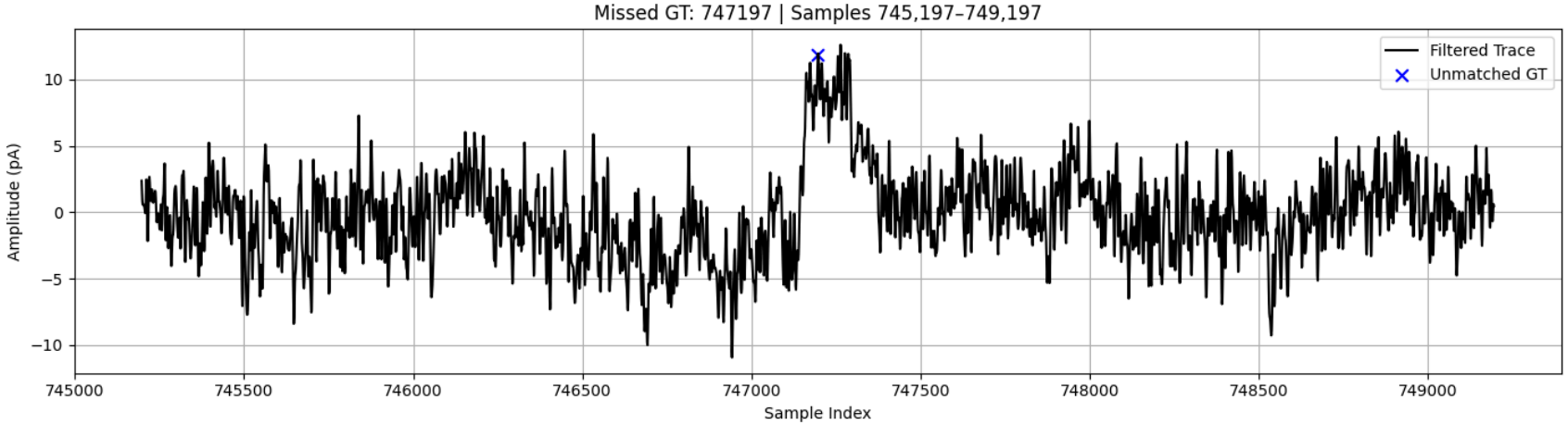

Missed GT: 608158 | Samples 606,158–610,158
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
606000
606500
607000
607500
608000
608500
609000
609500
610000
Sample Index

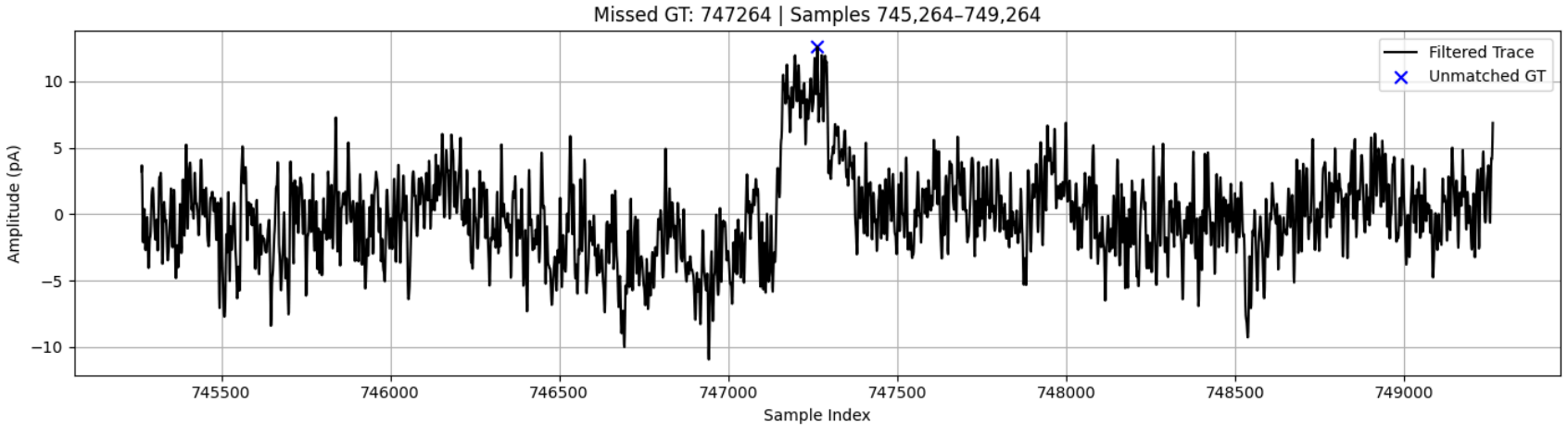

Missed GT: 643771 | Samples 641,771–645,771
Filtered Trace
Unmatched GT
Amplitude (pA)
40
30
20
10
0
−10
−20
642000
642500
643000
643500
644000
644500
645000
645500
Sample Index

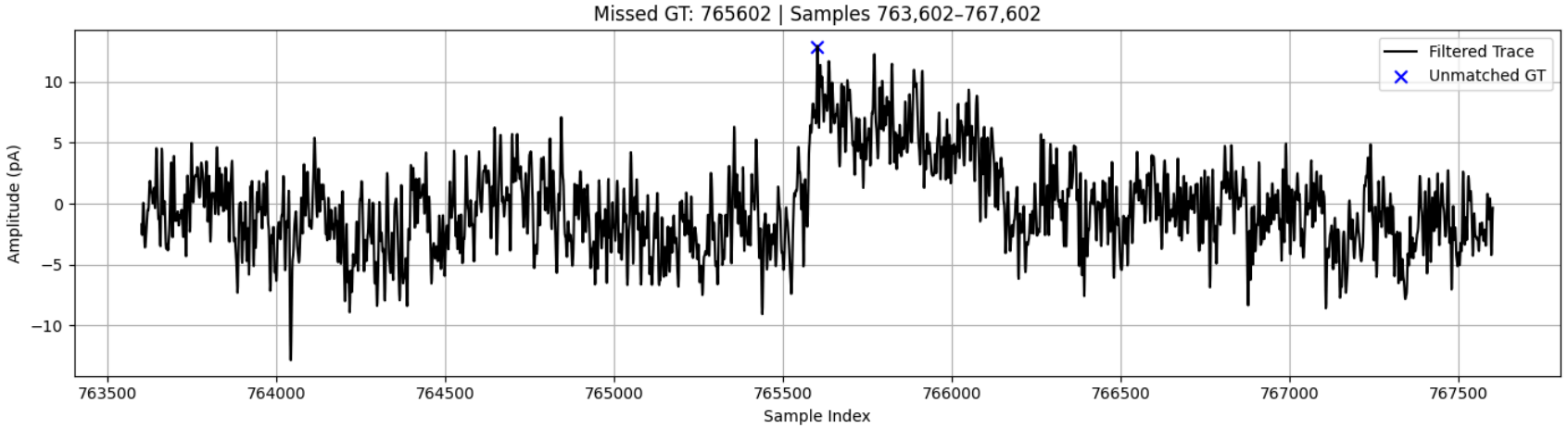

Missed GT: 654610 | Samples 652,610–656,610
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
25
20
15
10
5
0
−5
−10
−15
652500
653000
653500
654000
654500
655000
655500
656000
656500

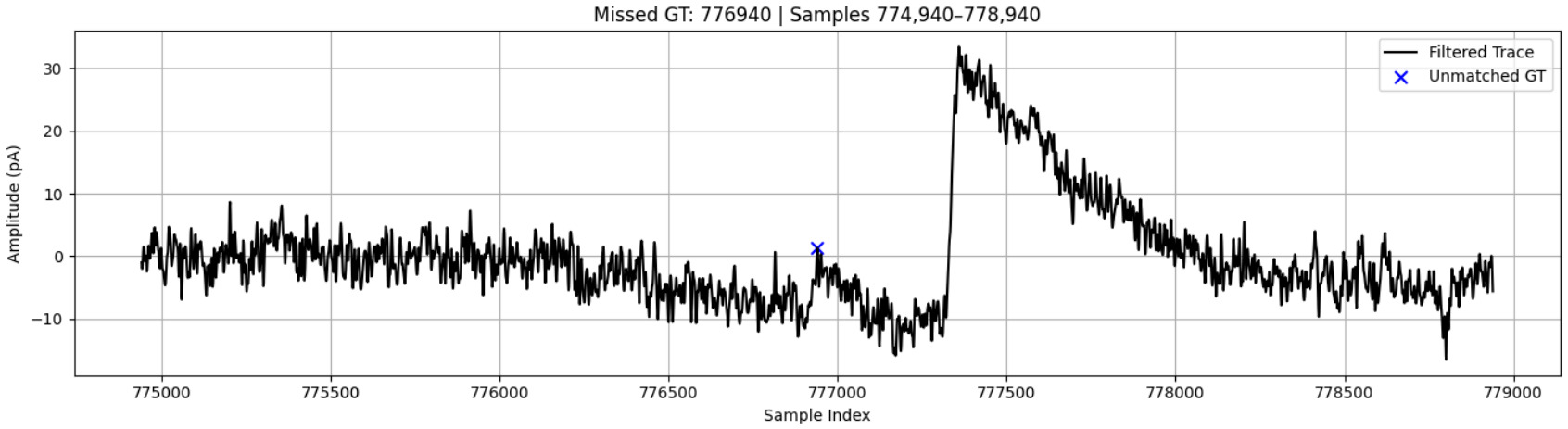

Missed GT: 706465 | Samples 704,465–708,465
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
704500
705000
705500
706000
706500
707000
707500
708000
708500

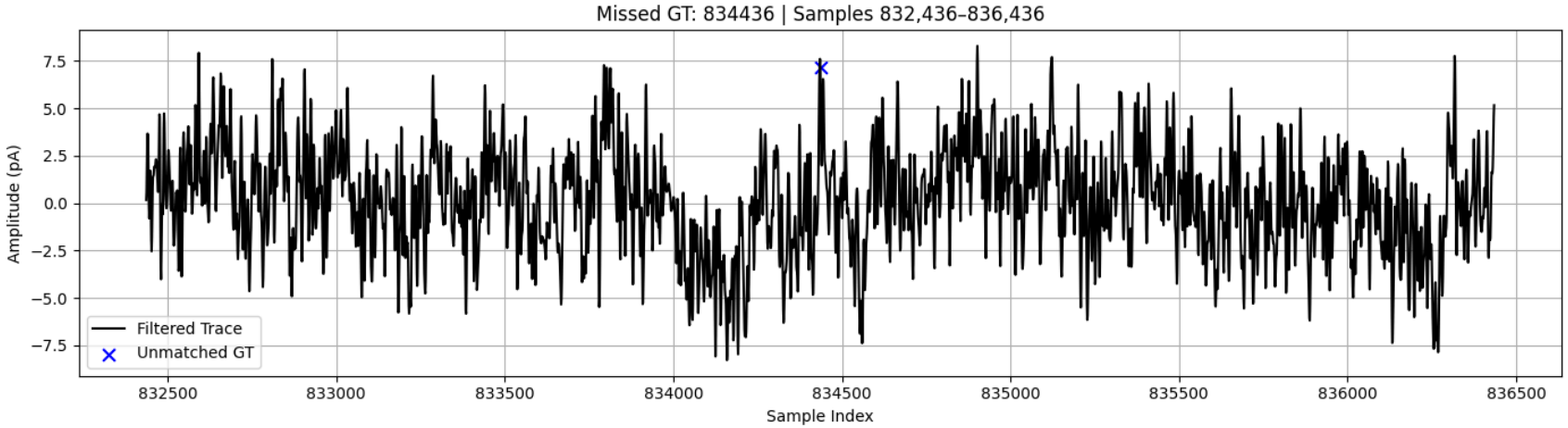

Missed GT: 720309 | Samples 718,309–722,309
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
30
20
10
0
−10
718500
719000
719500
720000
720500
721000
721500
722000
722500

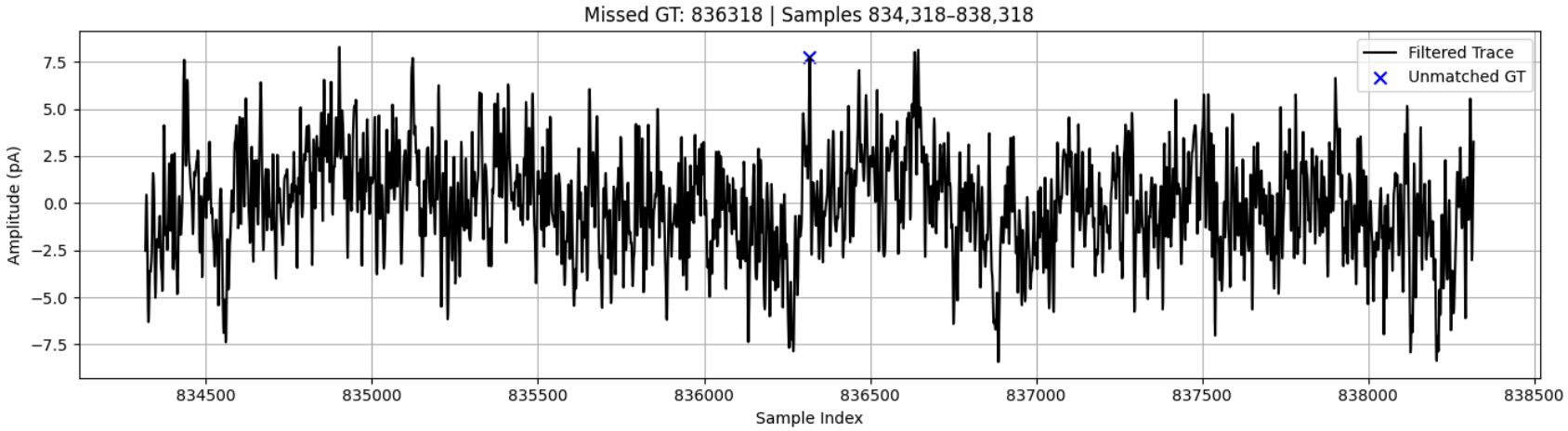

Missed GT: 747197 | Samples 745,197–749,197
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
10
5
0
−5
−10
745000
745500
746000
746500
747000
747500
748000
748500
749000

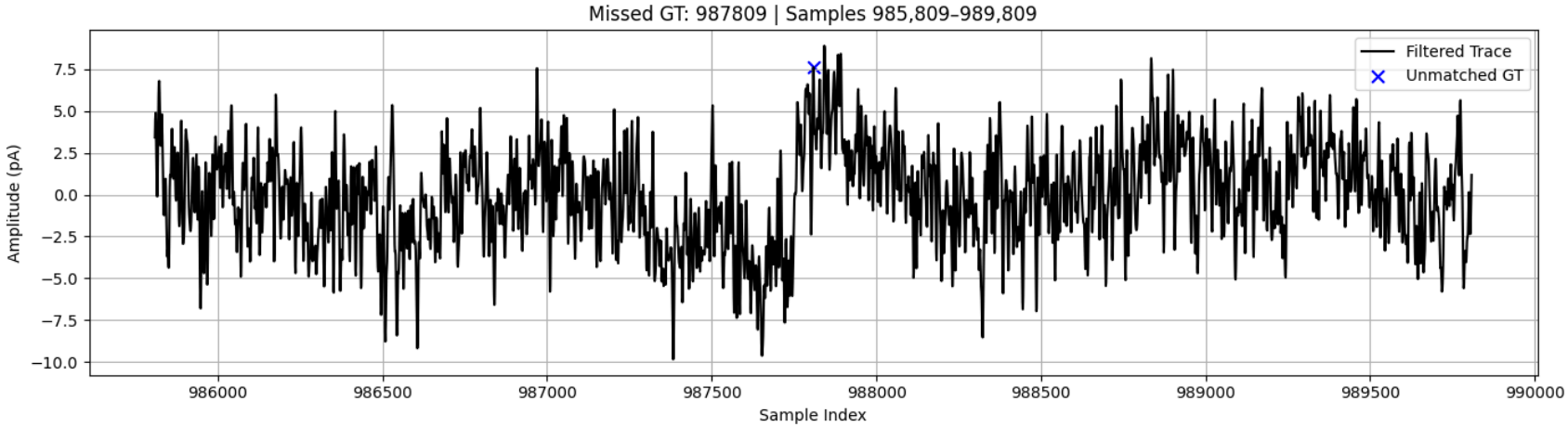

Missed GT: 747264 | Samples 745,264–749,264
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
745500
746000
746500
747000
747500
748000
748500
749000
Sample Index

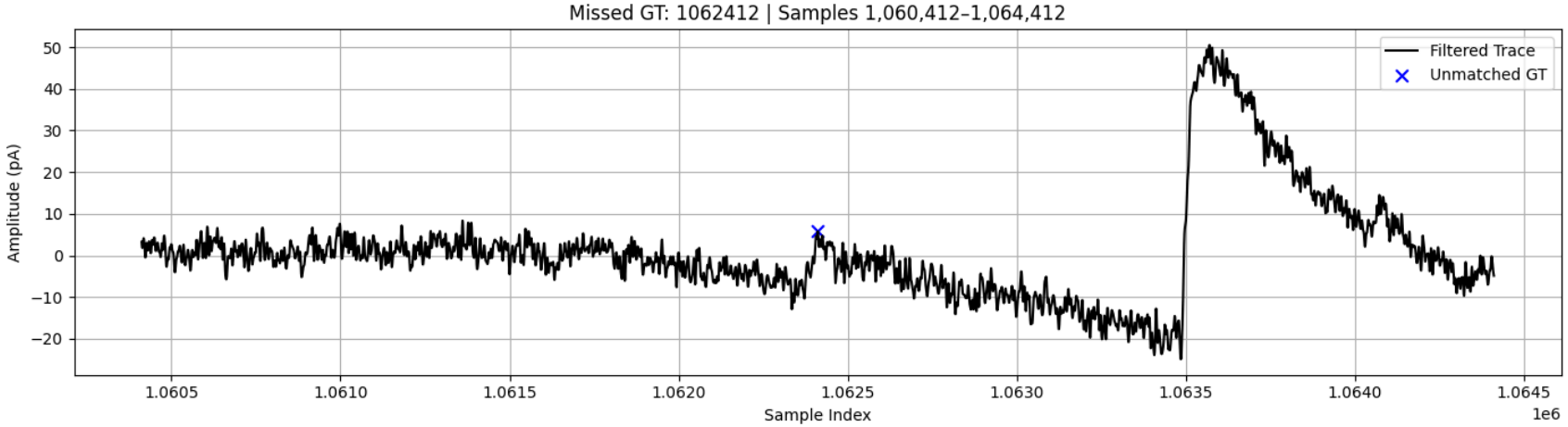

Missed GT: 765602 | Samples 763,602–767,602
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
763500
764000
764500
765000
765500
766000
766500
767000
767500
Sample Index

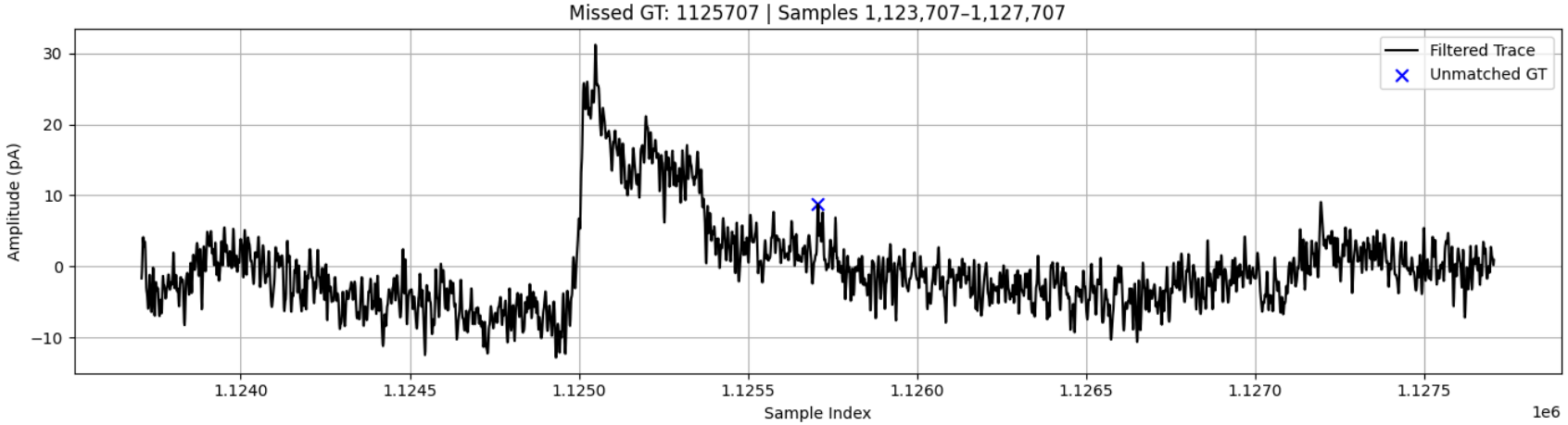

Missed GT: 776940 | Samples 774,940–778,940
Filtered Trace
Unmatched GT
Amplitude (pA)
30
20
10
0
−10
775000
775500
776000
776500
777000
777500
778000
778500
779000
Sample Index

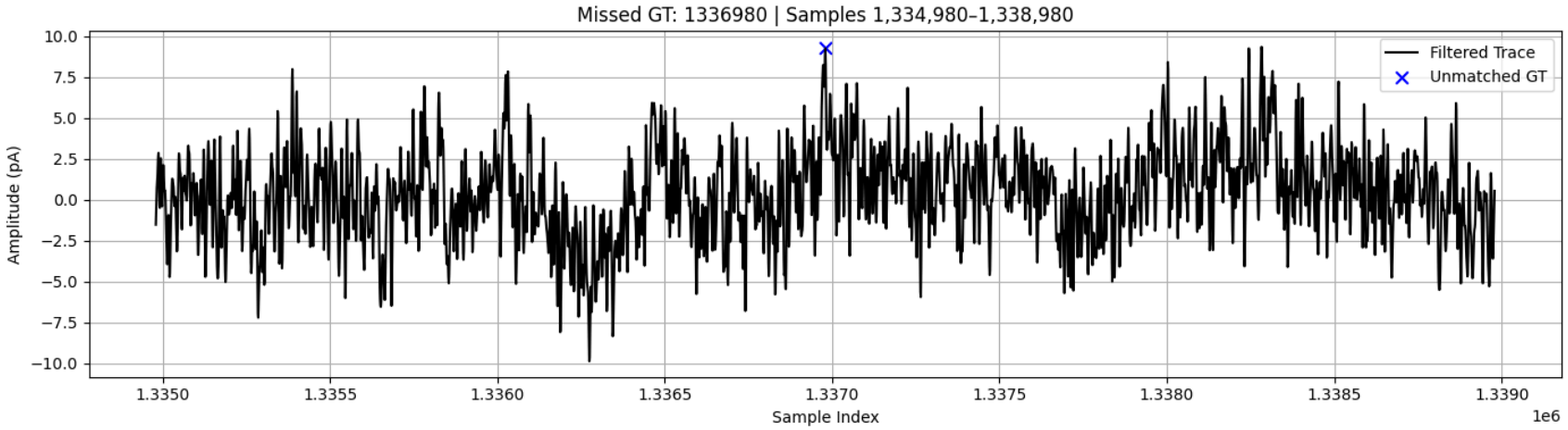

Missed GT: 834436 | Samples 832,436–836,436
Filtered Trace
Unmatched GT
Amplitude (pA)
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
832500
833000
833500
834000
834500
835000
835500
836000
836500
Sample Index

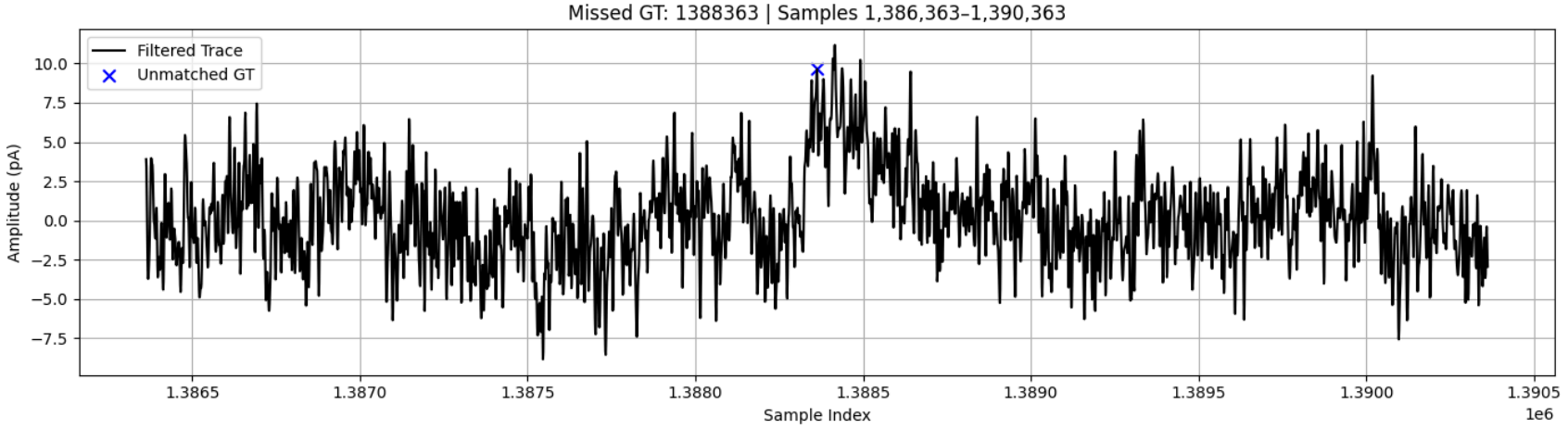

Missed GT: 836318 | Samples 834,318–838,318
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
834500
835000
835500
836000
836500
837000
837500
838000
838500

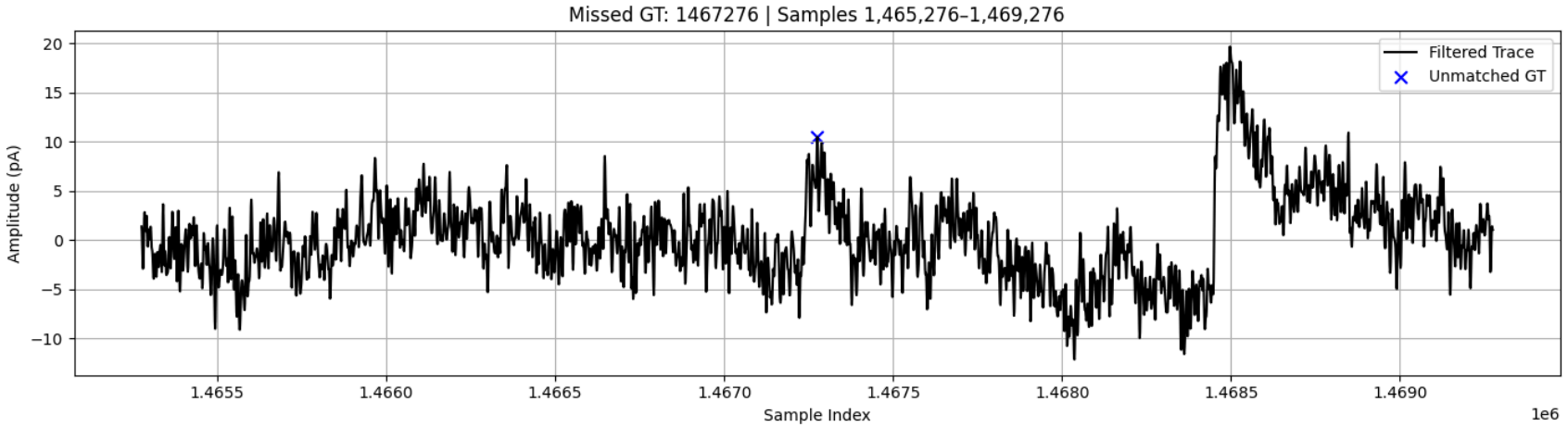

Missed GT: 987809 | Samples 985,809–989,809
Filtered Trace
Unmatched GT
Amplitude (pA)
Sample Index
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
986000
986500
987000
987500
988000
988500
989000
989500
990000

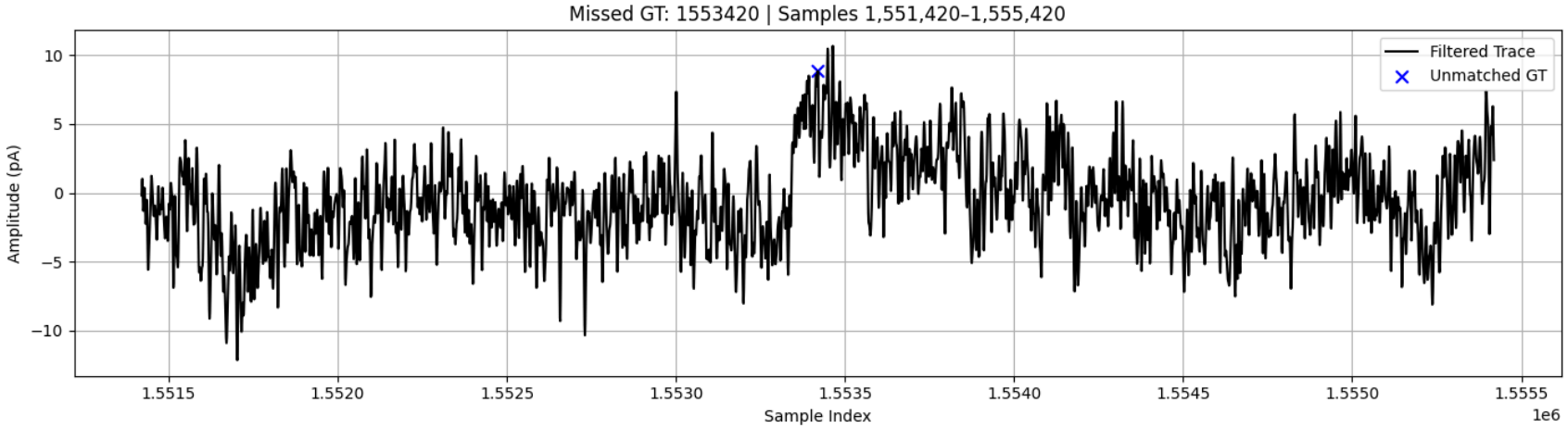

Missed GT: 1062412 | Samples 1,060,412–1,064,412
Filtered Trace
Unmatched GT
Amplitude (pA)
50
40
30
20
10
0
−10
−20
1.0605
1.0610
1.0615
1.0620
1.0625
1.0630
1.0635
1.0640
1.0645
Sample Index
1e6

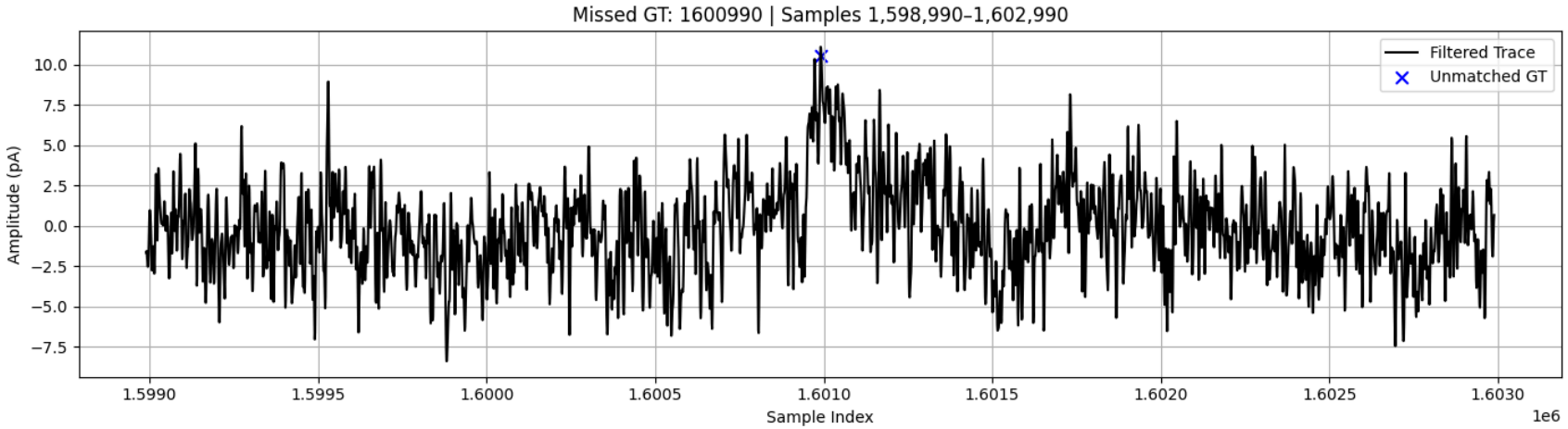

Missed GT: 1125707 | Samples 1,123,707–1,127,707
Filtered Trace
Unmatched GT
Amplitude (pA)
30
20
10
0
−10
1.1240
1.1245
1.1250
1.1255
1.1260
1.1265
1.1270
1.1275
Sample Index
1e6

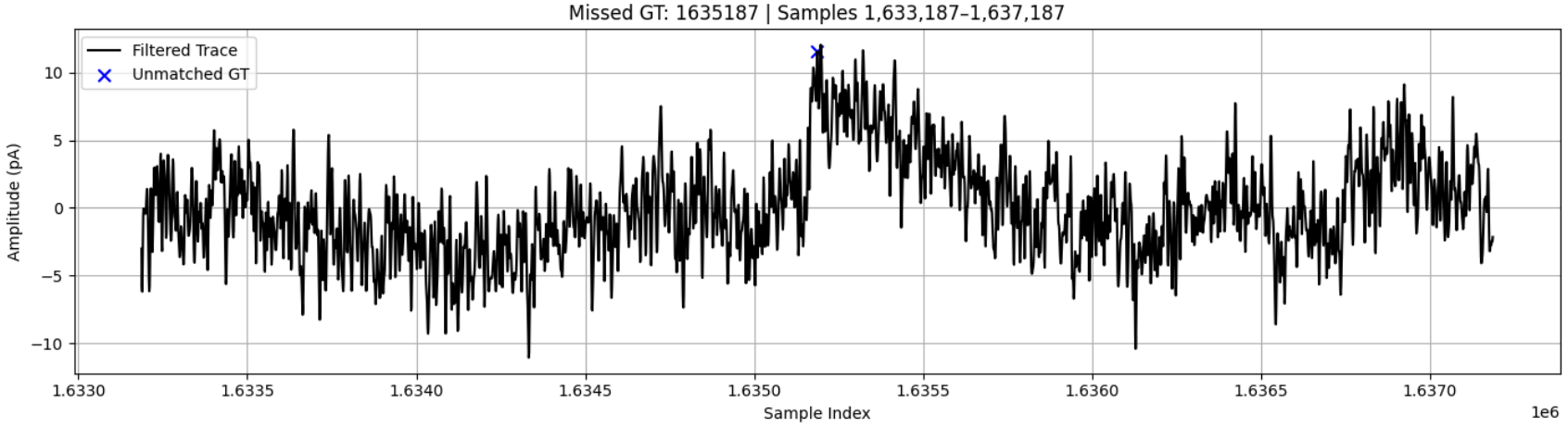

Missed GT: 1336980 | Samples 1,334,980–1,338,980
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
1.3350
1.3355
1.3360
1.3365
1.3370
1.3375
1.3380
1.3385
1.3390
1e6
Sample Index

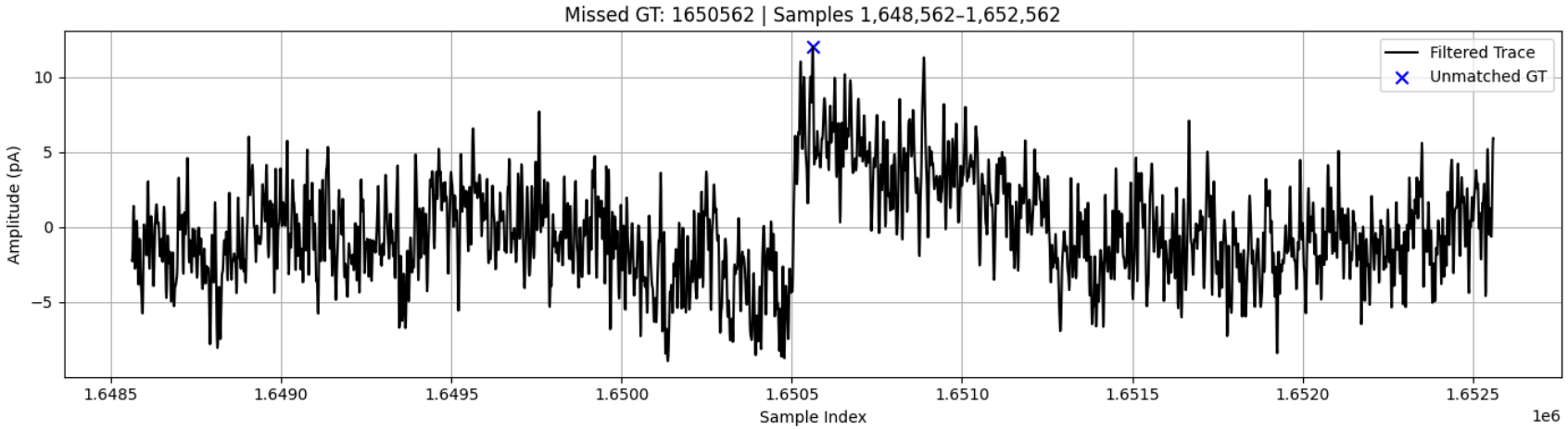

Missed GT: 1388363 | Samples 1,386,363–1,390,363
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
1.3865
1.3870
1.3875
1.3880
1.3885
1.3890
1.3895
1.3900
1.3905
1e6
Sample Index

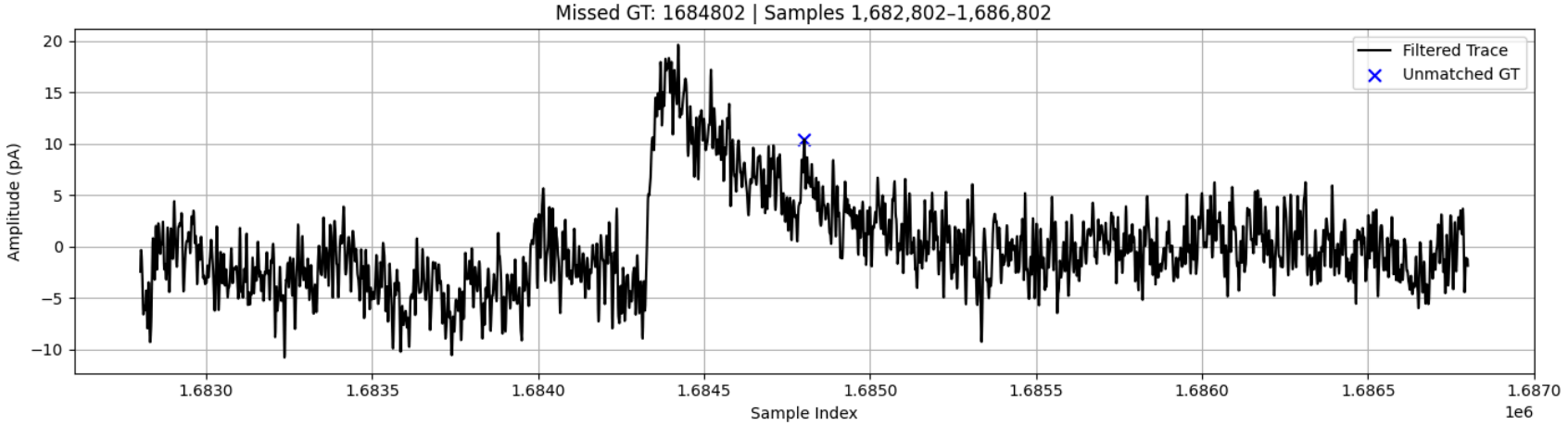

Missed GT: 1467276 | Samples 1,465,276–1,469,276
Filtered Trace
Unmatched GT
Amplitude (pA)
20
15
10
5
0
−5
−10
1.4655
1.4660
1.4665
1.4670
1.4675
1.4680
1.4685
1.4690
Sample Index
1e6

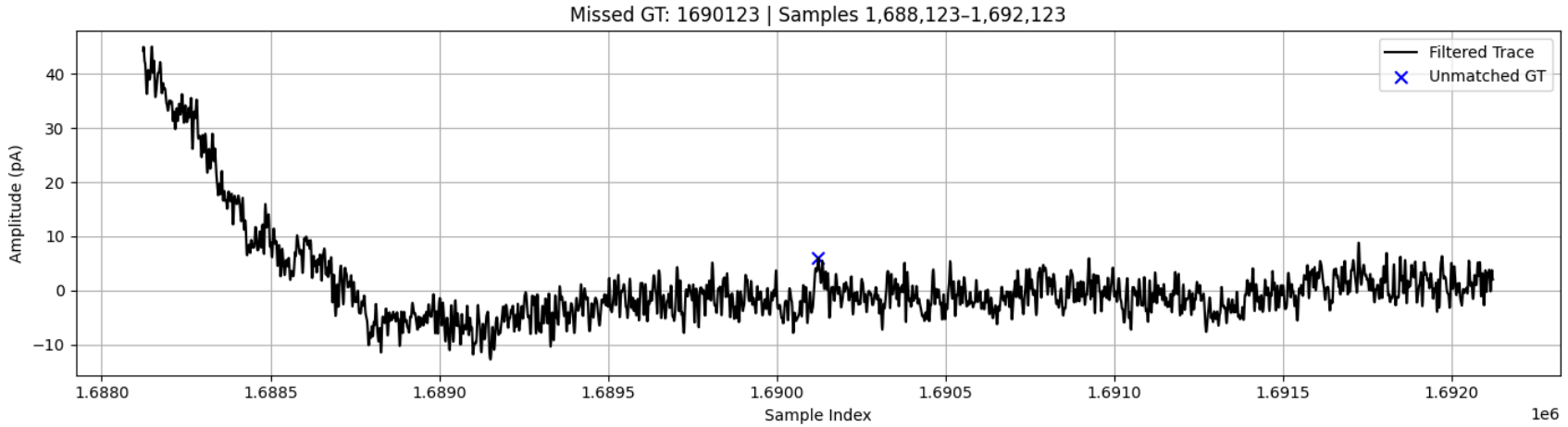

Missed GT: 1553420 | Samples 1,551,420–1,555,420
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
1.5515
1.5520
1.5525
1.5530
1.5535
1.5540
1.5545
1.5550
1.5555
Sample Index
1e6

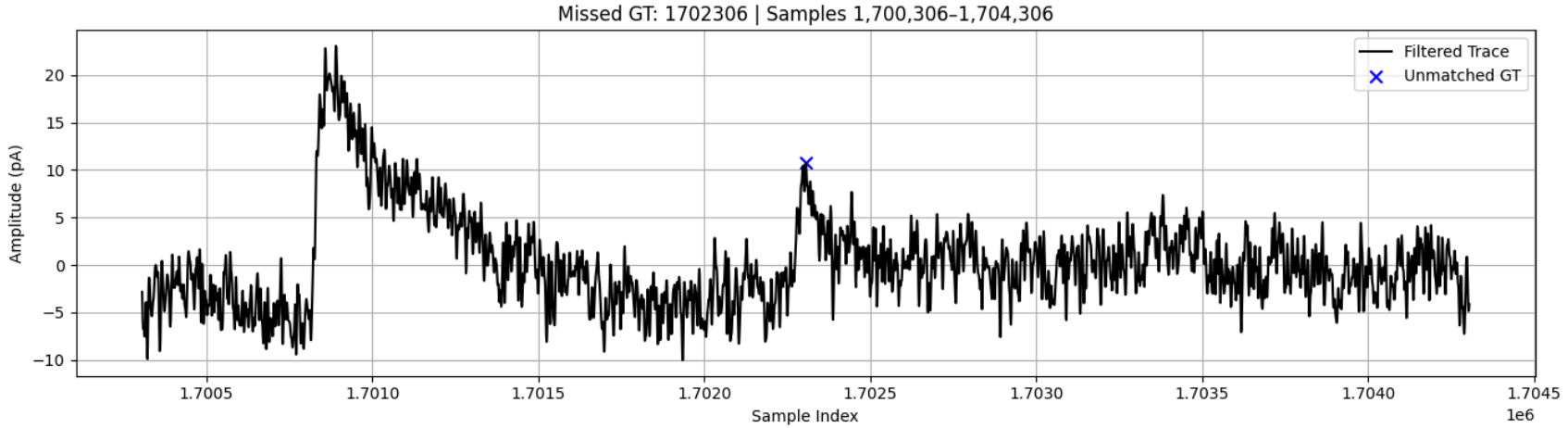

Missed GT: 1600990 | Samples 1,598,990–1,602,990
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
1.5990
1.5995
1.6000
1.6005
1.6010
1.6015
1.6020
1.6025
1.6030
Sample Index
1e6

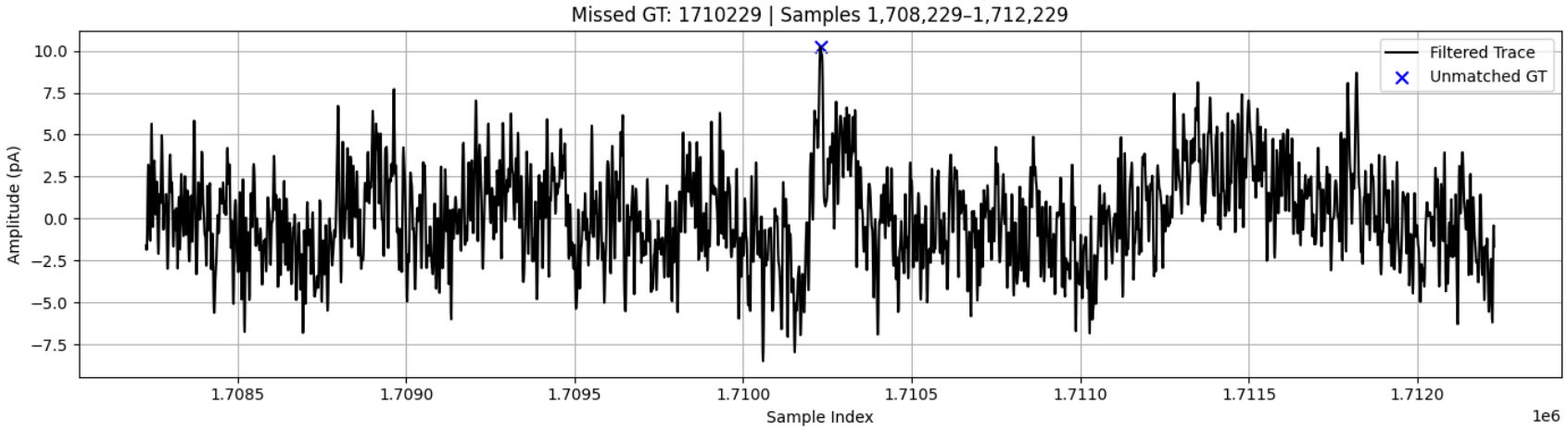

Missed GT: 1635187 | Samples 1,633,187–1,637,187
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
1.6330
1.6335
1.6340
1.6345
1.6350
1.6355
1.6360
1.6365
1.6370
Sample Index
1e6

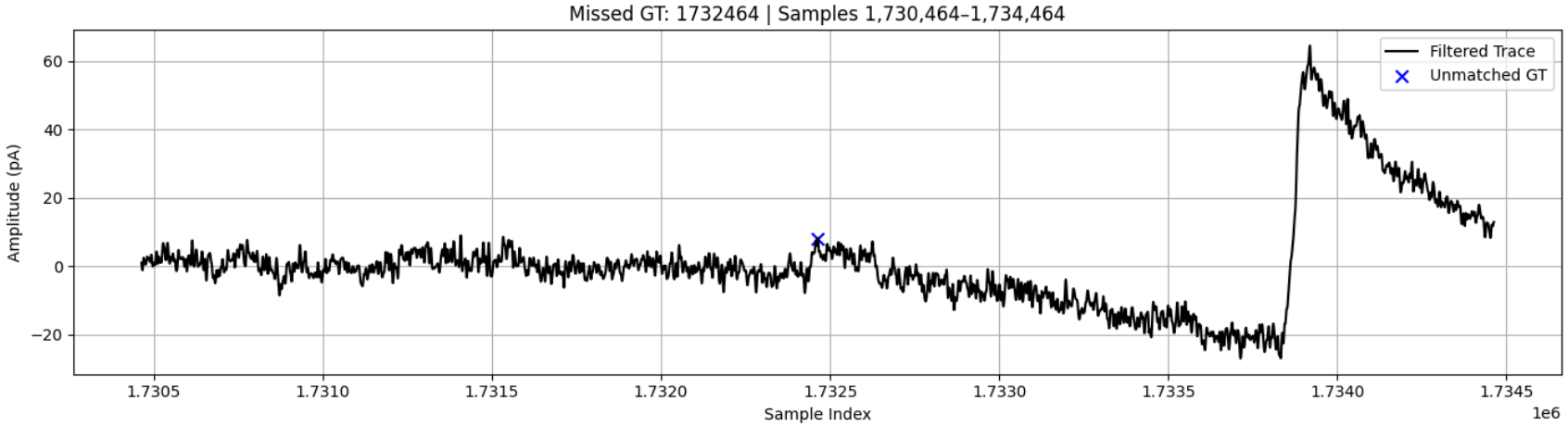

Missed GT: 1650562 | Samples 1,648,562–1,652,562
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
1.6485
1.6490
1.6495
1.6500
1.6505
1.6510
1.6515
1.6520
1.6525
Sample Index
1e6

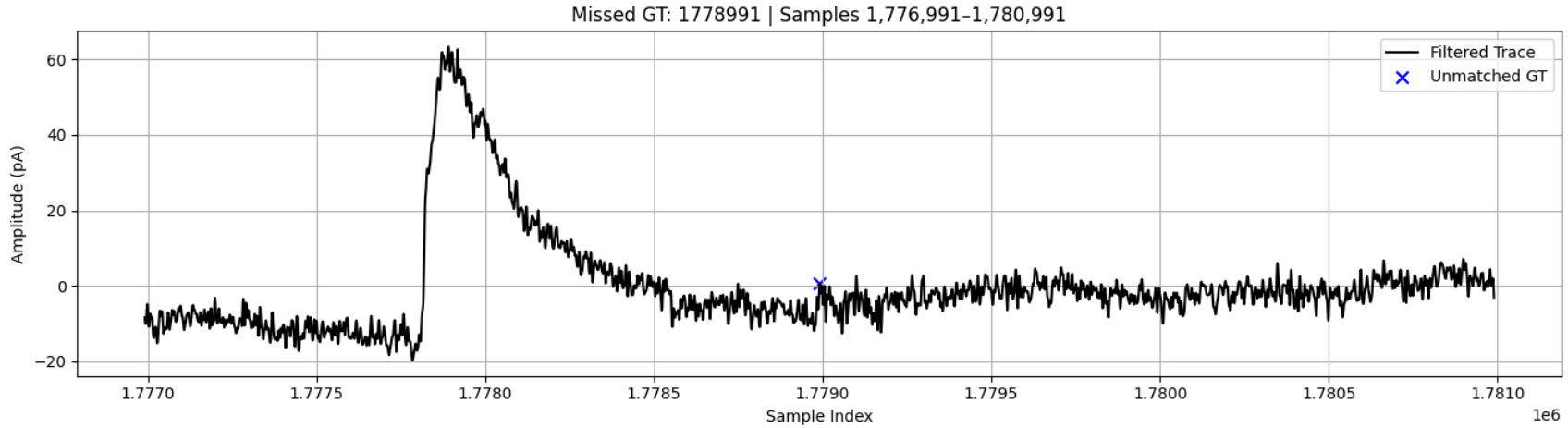

Missed GT: 1684802 | Samples 1,682,802–1,686,802
Filtered Trace
Unmatched GT
Amplitude (pA)
20
15
10
5
0
−5
−10
1.6830
1.6835
1.6840
1.6845
1.6850
1.6855
1.6860
1.6865
1.6870
Sample Index
1e6

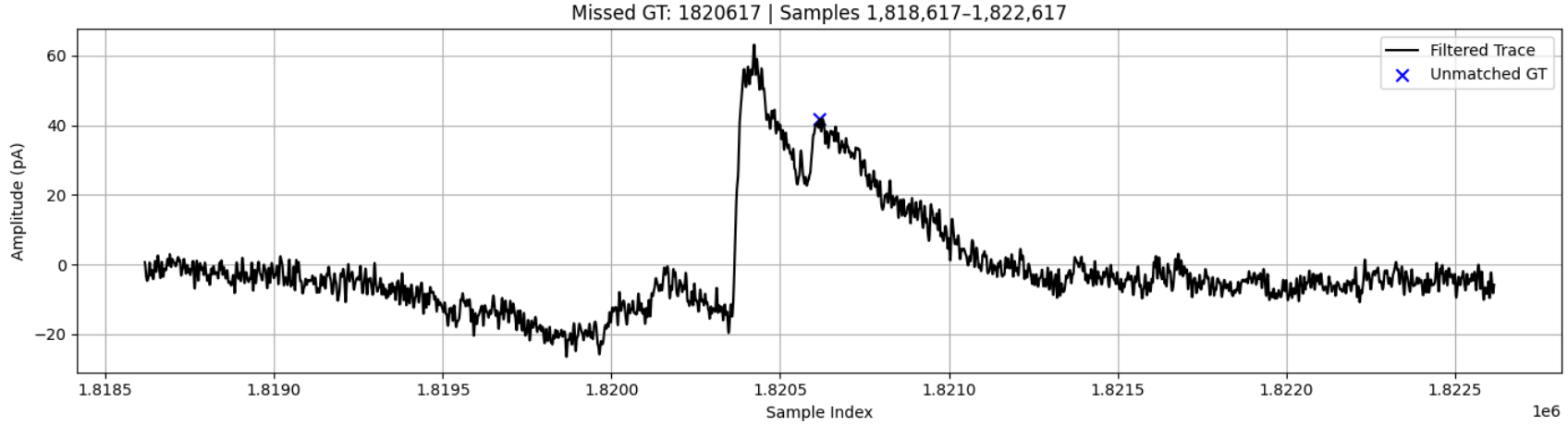

Missed GT: 1690123 | Samples 1,688,123–1,692,123
Filtered Trace
Unmatched GT
Amplitude (pA)
40
30
20
10
0
−10
1.6880
1.6885
1.6890
1.6895
1.6900
1.6905
1.6910
1.6915
1.6920
Sample Index
1e6

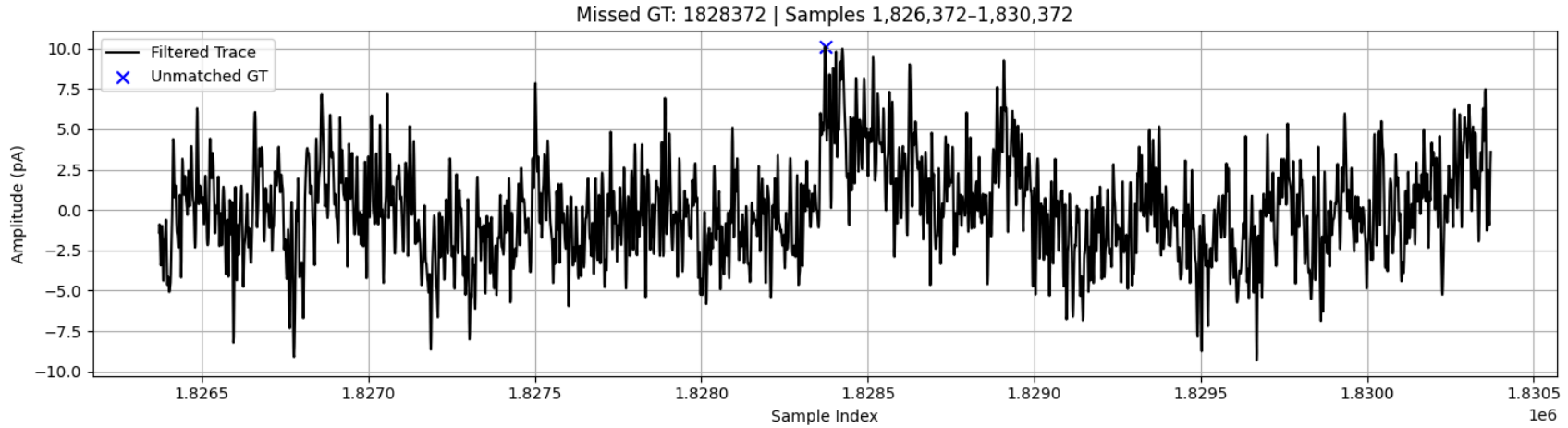

Missed GT: 1702306 | Samples 1,700,306–1,704,306
Filtered Trace
Unmatched GT
Amplitude (pA)
20
15
10
5
0
−5
−10
1.7005
1.7010
1.7015
1.7020
1.7025
1.7030
1.7035
1.7040
1.7045
Sample Index
1e6

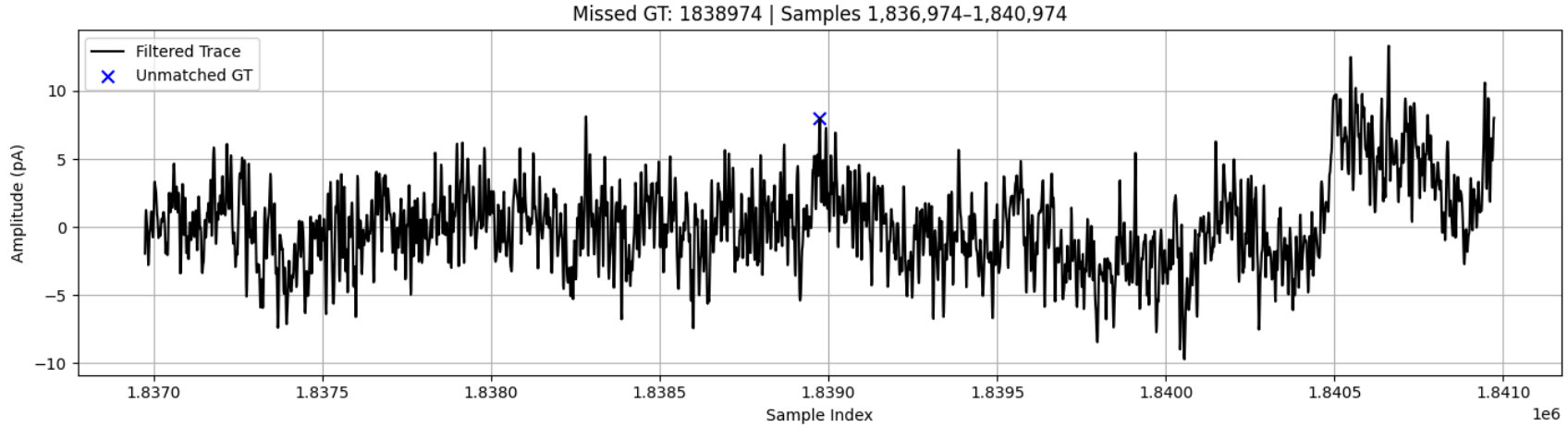

Missed GT: 1710229 | Samples 1,708,229–1,712,229
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
1.7085
1.7090
1.7095
1.7100
1.7105
1.7110
1.7115
1.7120
Sample Index
1e6

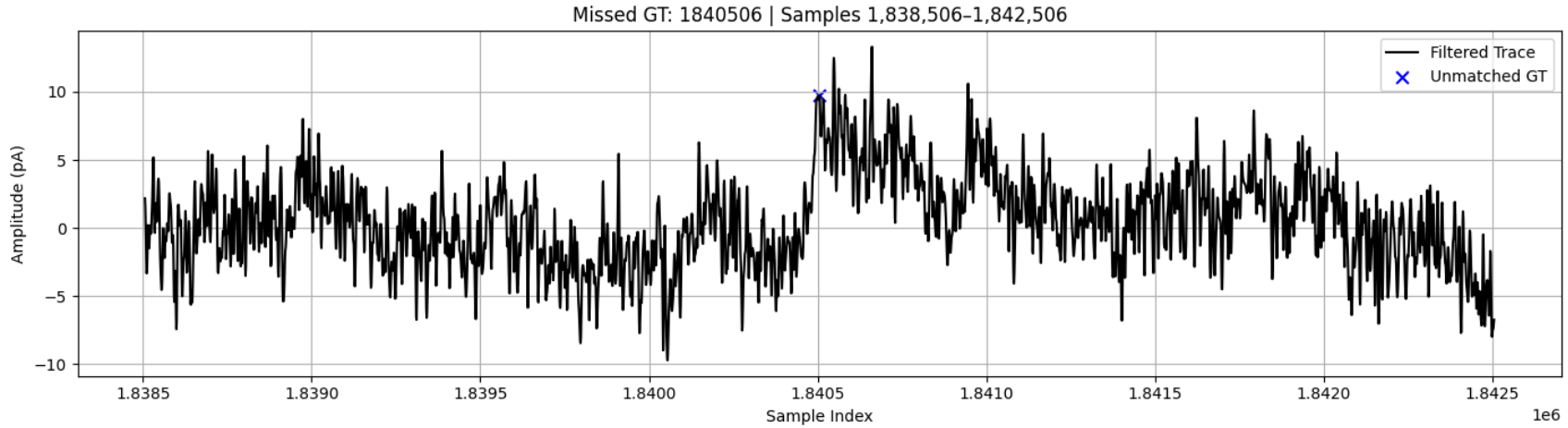

Missed GT: 1732464 | Samples 1,730,464–1,734,464
Filtered Trace
Unmatched GT
Amplitude (pA)
60
40
20
0
−20
1.7305
1.7310
1.7315
1.7320
1.7325
1.7330
1.7335
1.7340
1.7345
Sample Index
1e6

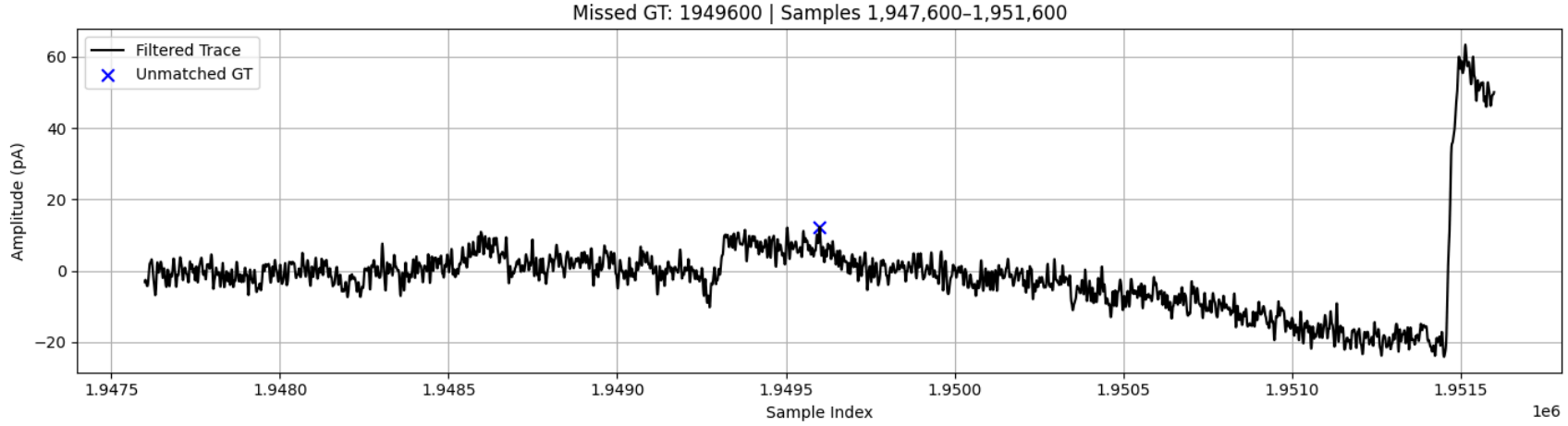

Missed GT: 1778991 | Samples 1,776,991–1,780,991
Filtered Trace
Unmatched GT
Amplitude (pA)
60
40
20
0
−20
1.7770
1.7775
1.7780
1.7785
1.7790
1.7795
1.7800
1.7805
1.7810
Sample Index
1e6

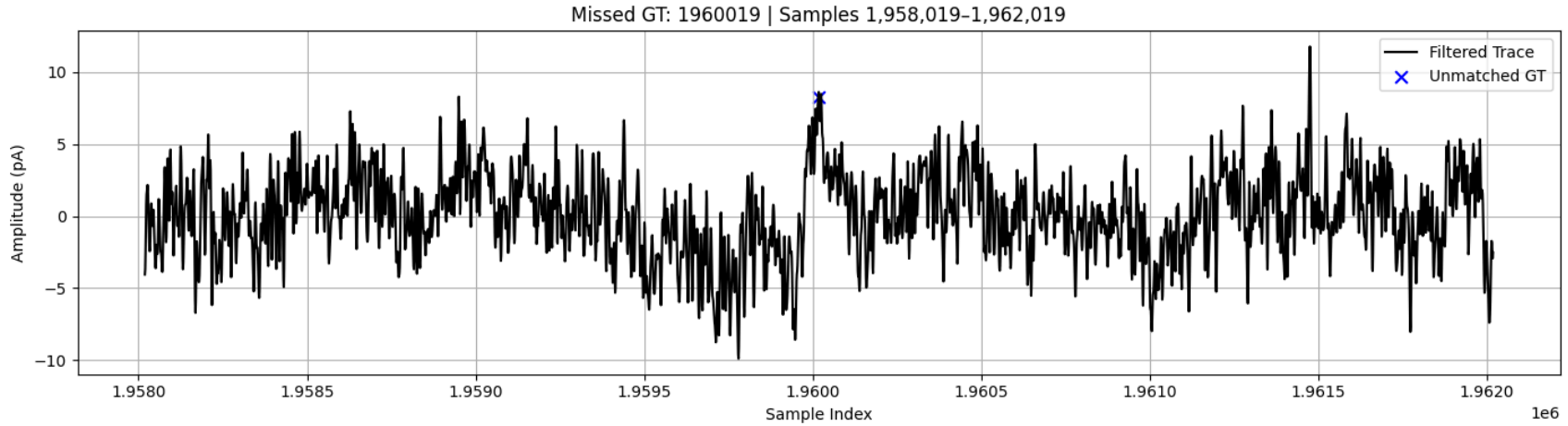

Missed GT: 1820617 | Samples 1,818,617–1,822,617
Filtered Trace
Unmatched GT
Amplitude (pA)
60
40
20
0
−20
1.8185
1.8190
1.8195
1.8200
1.8205
1.8210
1.8215
1.8220
1.8225
Sample Index
1e6

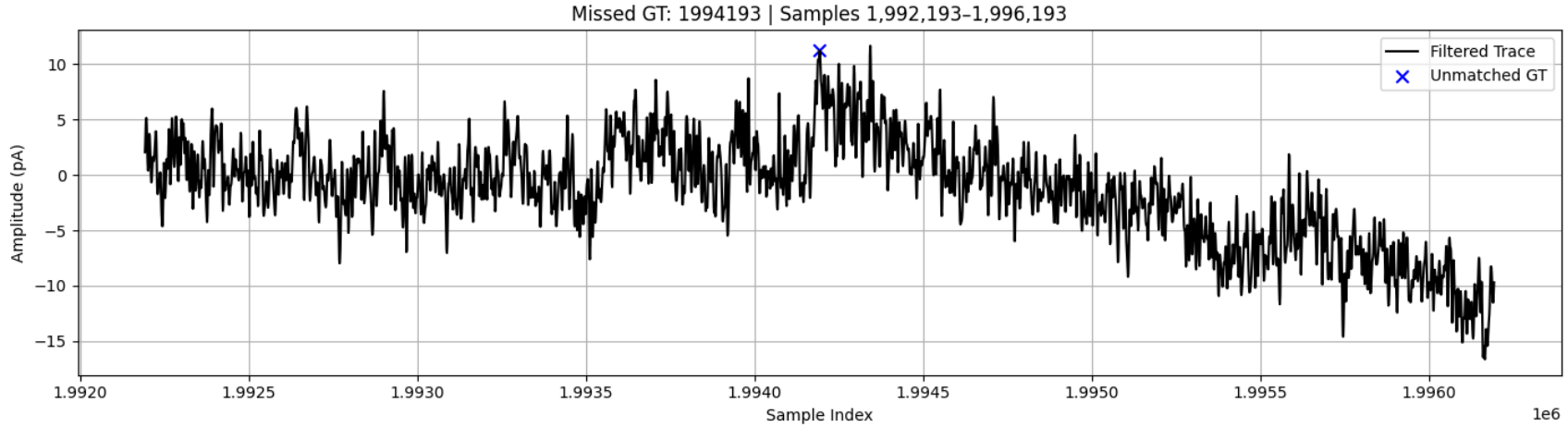

Missed GT: 1828372 | Samples 1,826,372–1,830,372
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
1.8265
1.8270
1.8275
1.8280
1.8285
1.8290
1.8295
1.8300
1.8305
Sample Index
1e6

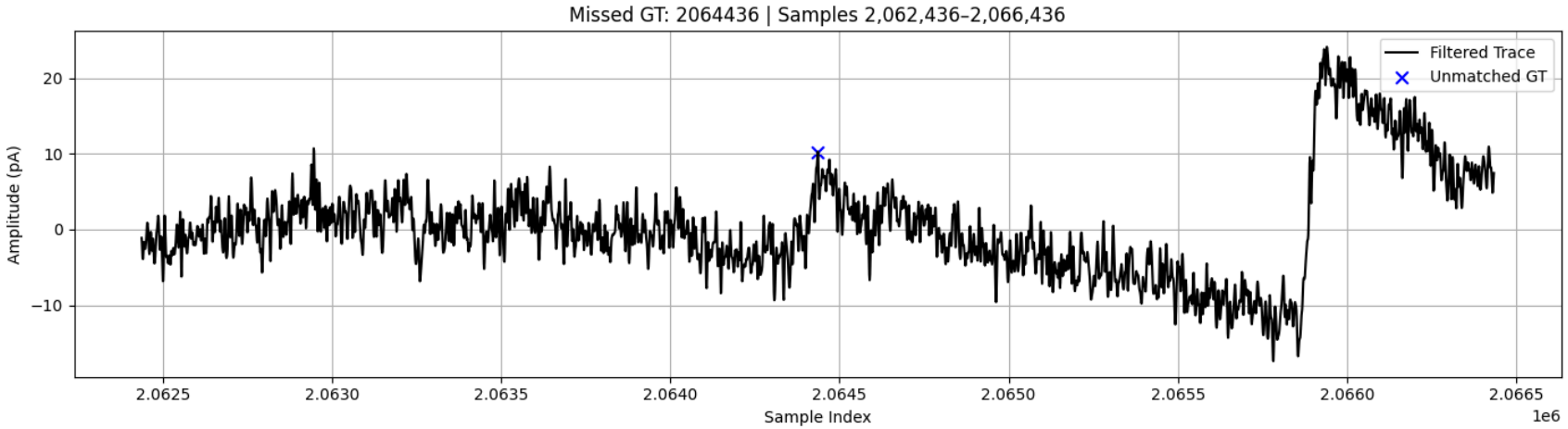

Missed GT: 1838974 | Samples 1,836,974–1,840,974
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
1.8370
1.8375
1.8380
1.8385
1.8390
1.8395
1.8400
1.8405
1.8410
Sample Index
1e6

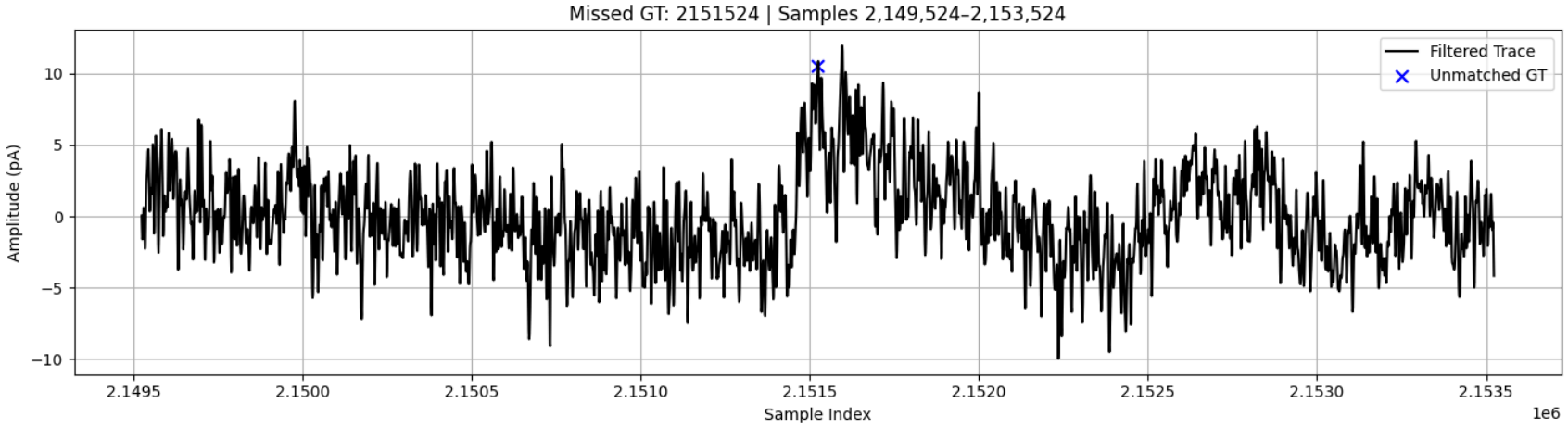

Missed GT: 1840506 | Samples 1,838,506–1,842,506
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
1.8385
1.8390
1.8395
1.8400
1.8405
1.8410
1.8415
1.8420
1.8425
Sample Index
1e6

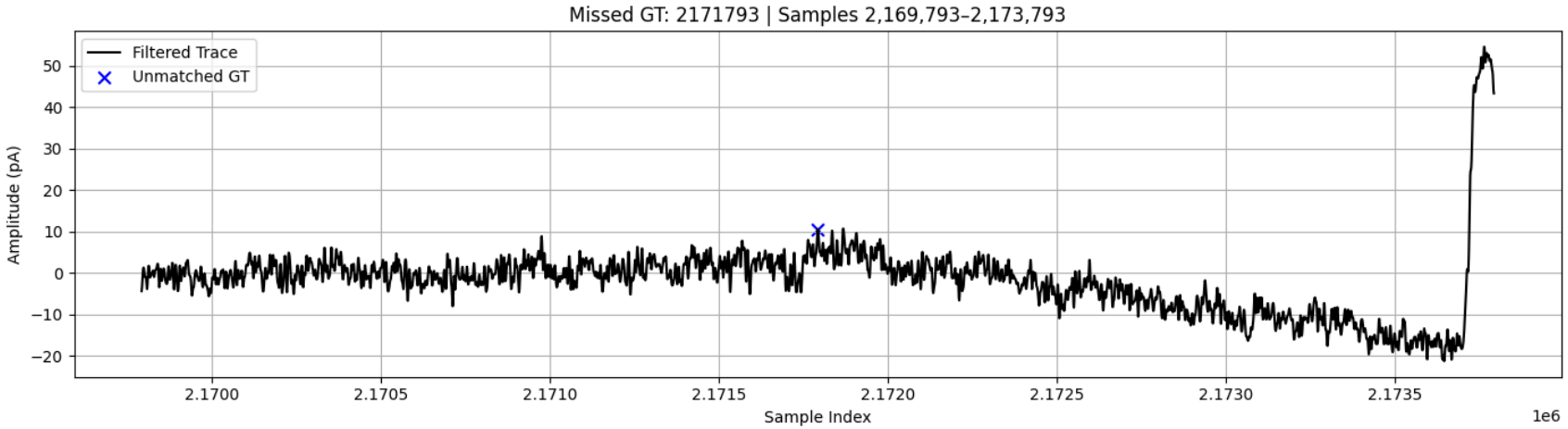

Missed GT: 1949600 | Samples 1,947,600–1,951,600
Filtered Trace
Unmatched GT
Amplitude (pA)
60
40
20
0
−20
1.9475
1.9480
1.9485
1.9490
1.9495
1.9500
1.9505
1.9510
1.9515
Sample Index
1e6

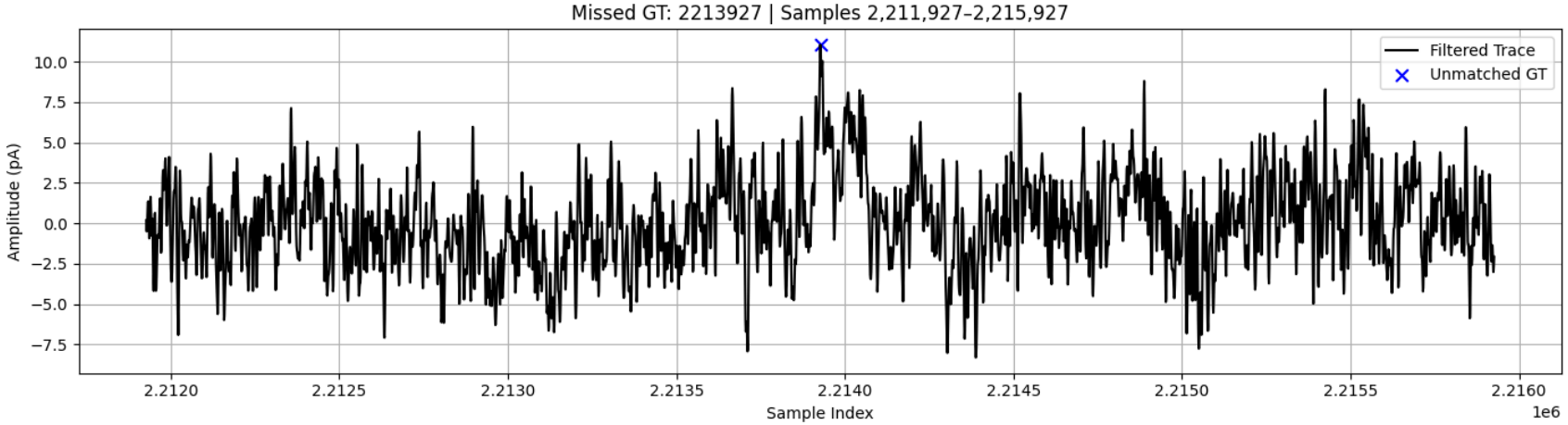

Missed GT: 1960019 | Samples 1,958,019–1,962,019
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
1.9580
1.9585
1.9590
1.9595
1.9600
1.9605
1.9610
1.9615
1.9620
Sample Index
1e6

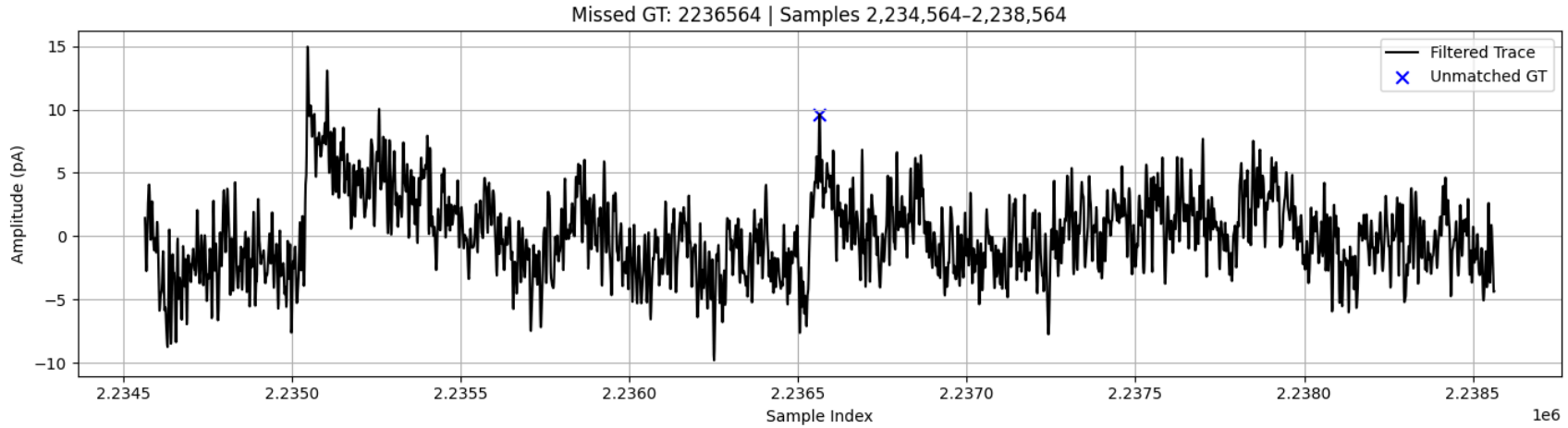

Missed GT: 1994193 | Samples 1,992,193–1,996,193
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
−15
1.9920
1.9925
1.9930
1.9935
1.9940
1.9945
1.9950
1.9955
1.9960
Sample Index
1e6

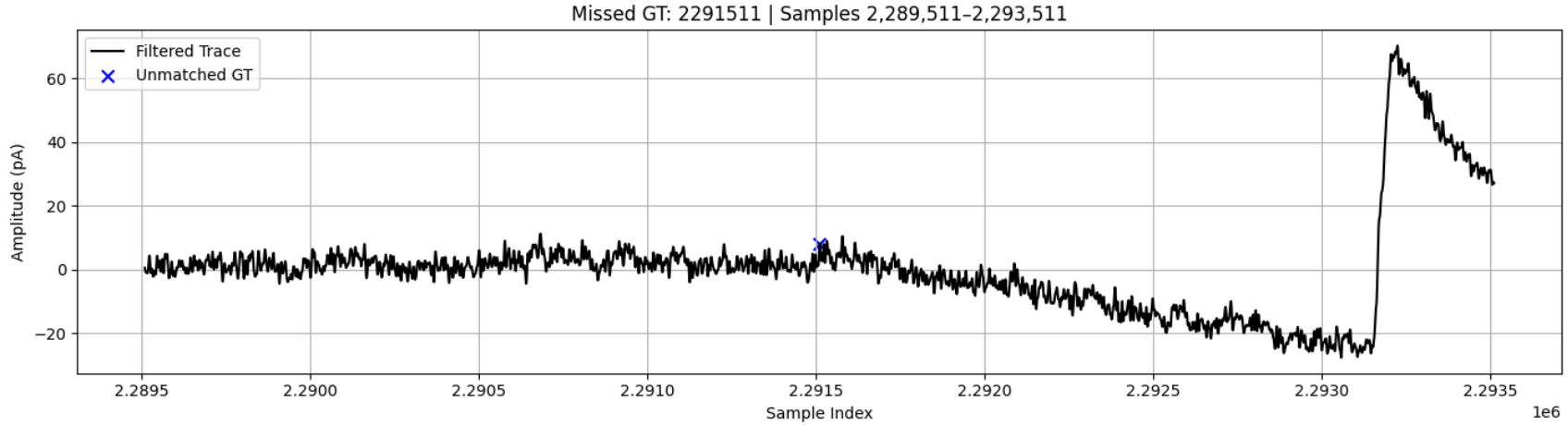

Missed GT: 2064436 | Samples 2,062,436–2,066,436
Filtered Trace
Unmatched GT
Amplitude (pA)
20
10
0
−10
2.0625
2.0630
2.0635
2.0640
2.0645
2.0650
2.0655
2.0660
2.0665
Sample Index
1e6

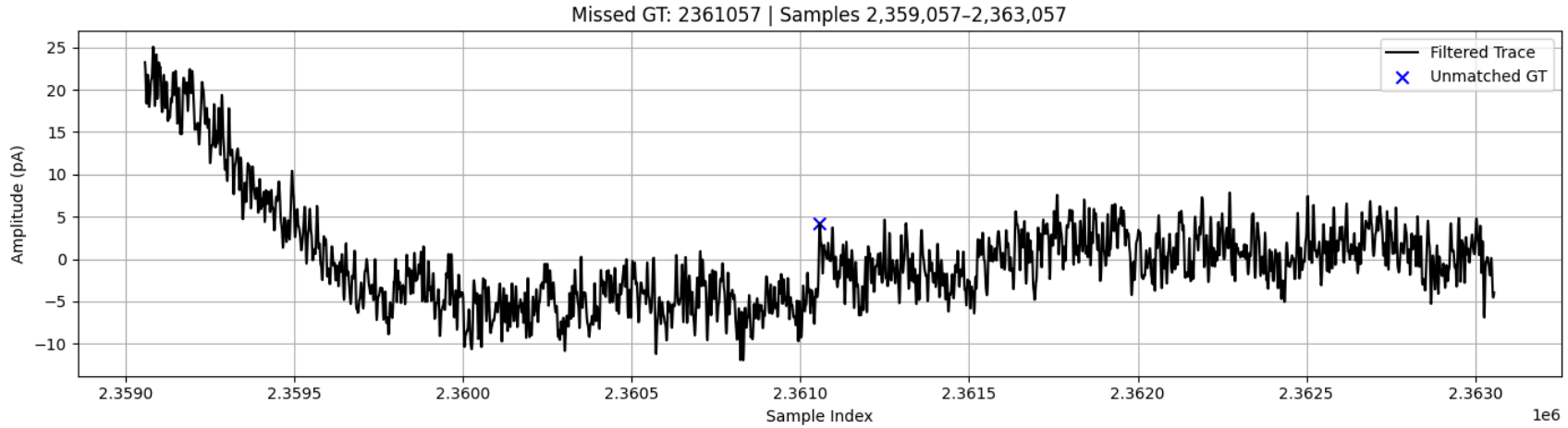

Missed GT: 2151524 | Samples 2,149,524–2,153,524
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
2.1495
2.1500
2.1505
2.1510
2.1515
2.1520
2.1525
2.1530
2.1535
Sample Index
1e6

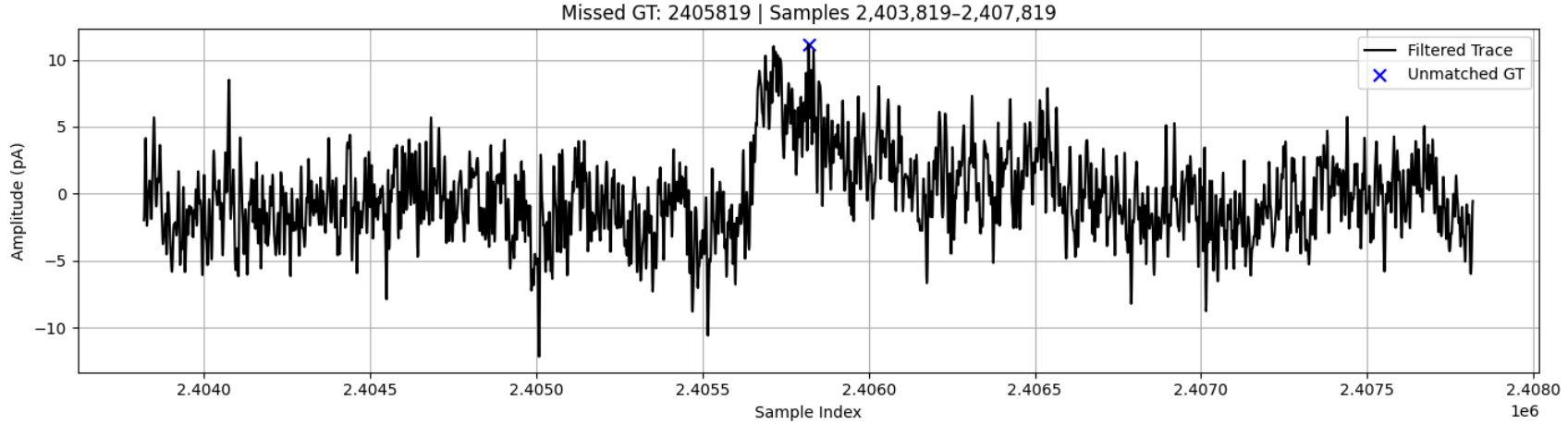

Missed GT: 2171793 | Samples 2,169,793–2,173,793
Filtered Trace
Unmatched GT
Amplitude (pA)
50
40
30
20
10
0
−10
−20
2.1700
2.1705
2.1710
2.1715
2.1720
2.1725
2.1730
2.1735
Sample Index
1e6

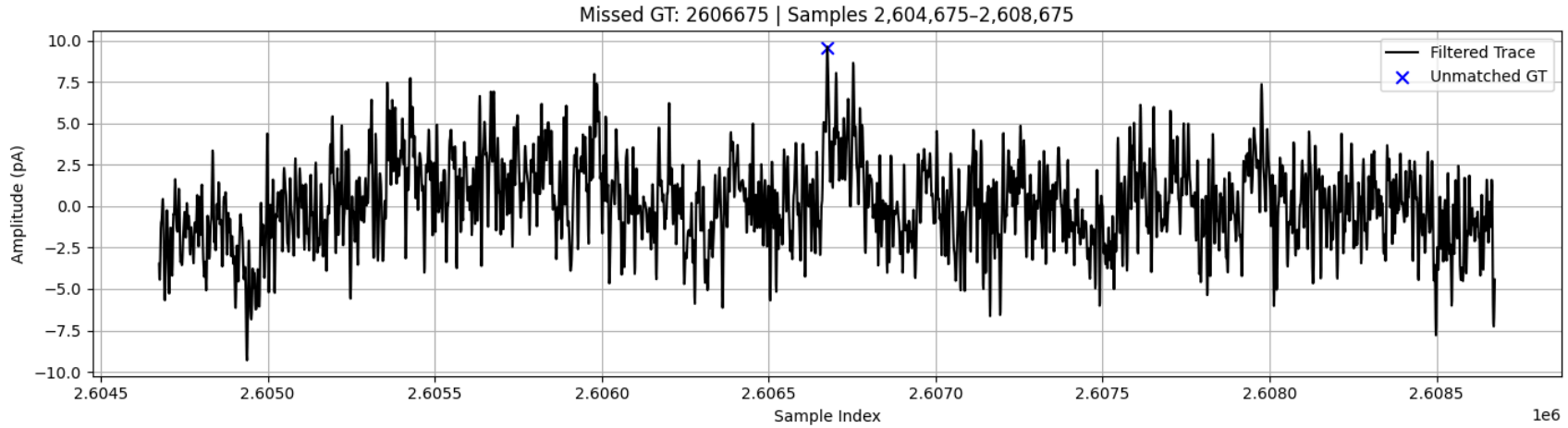

Missed GT: 2213927 | Samples 2,211,927–2,215,927
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
2.2120
2.2125
2.2130
2.2135
2.2140
2.2145
2.2150
2.2155
2.2160
Sample Index
1e6

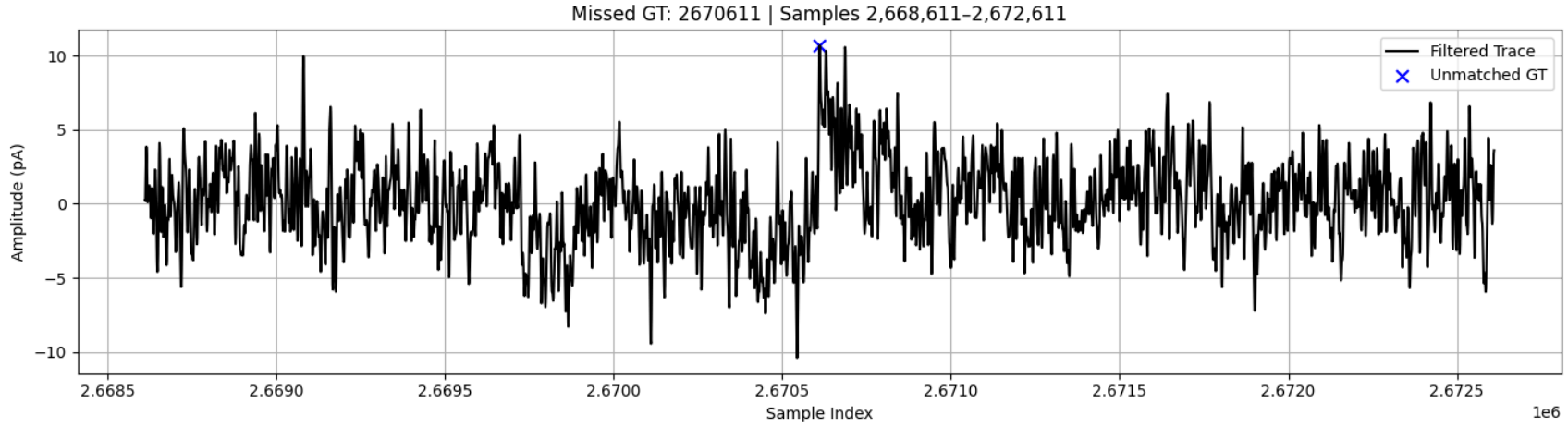

Missed GT: 2236564 | Samples 2,234,564–2,238,564
Filtered Trace
Unmatched GT
Amplitude (pA)
15
10
5
0
−5
−10
2.2345
2.2350
2.2355
2.2360
2.2365
2.2370
2.2375
2.2380
2.2385
Sample Index
1e6

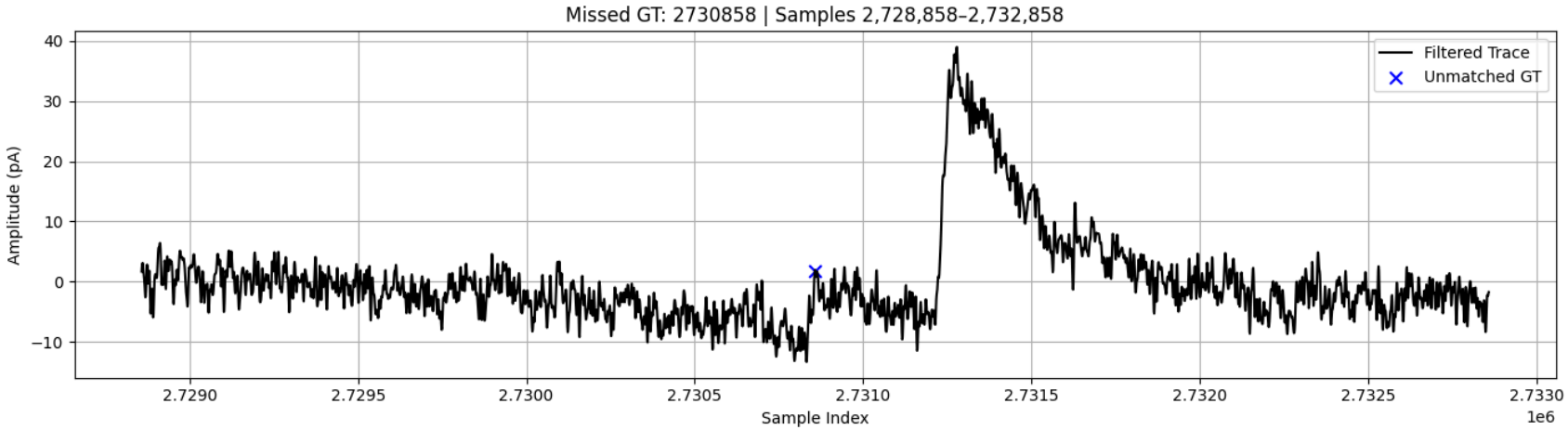

Missed GT: 2291511 | Samples 2,289,511–2,293,511
Filtered Trace
Unmatched GT
Amplitude (pA)
60
40
20
0
−20
2.2895
2.2900
2.2905
2.2910
2.2915
2.2920
2.2925
2.2930
2.2935
Sample Index
1e6

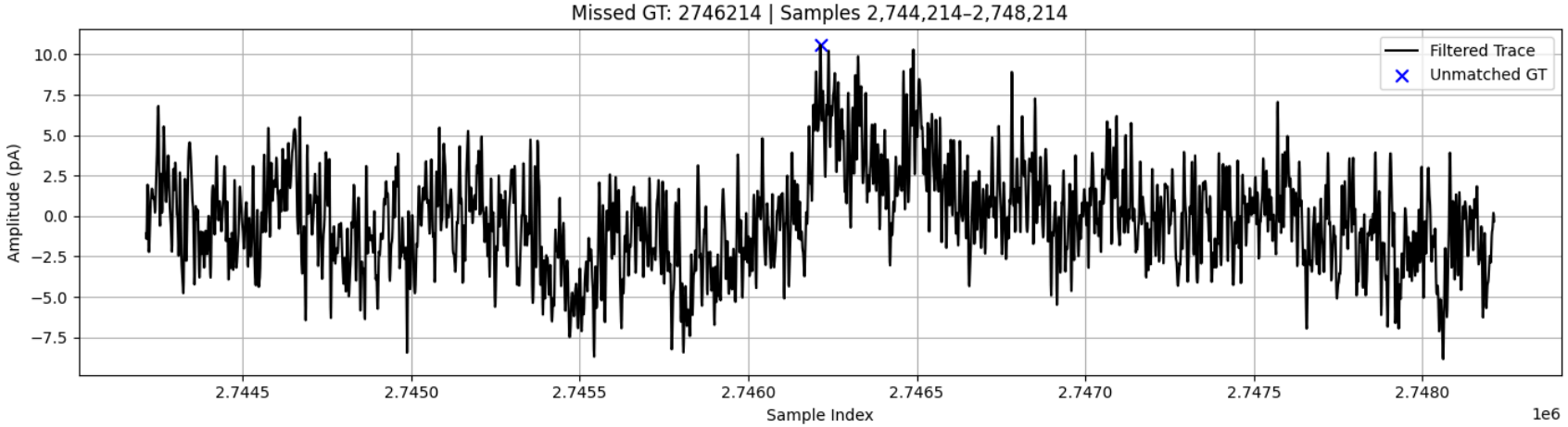

Missed GT: 2361057 | Samples 2,359,057–2,363,057
Filtered Trace
Unmatched GT
Amplitude (pA)
25
20
15
10
5
0
−5
−10
2.3590
2.3595
2.3600
2.3605
2.3610
2.3615
2.3620
2.3625
2.3630
Sample Index
1e6

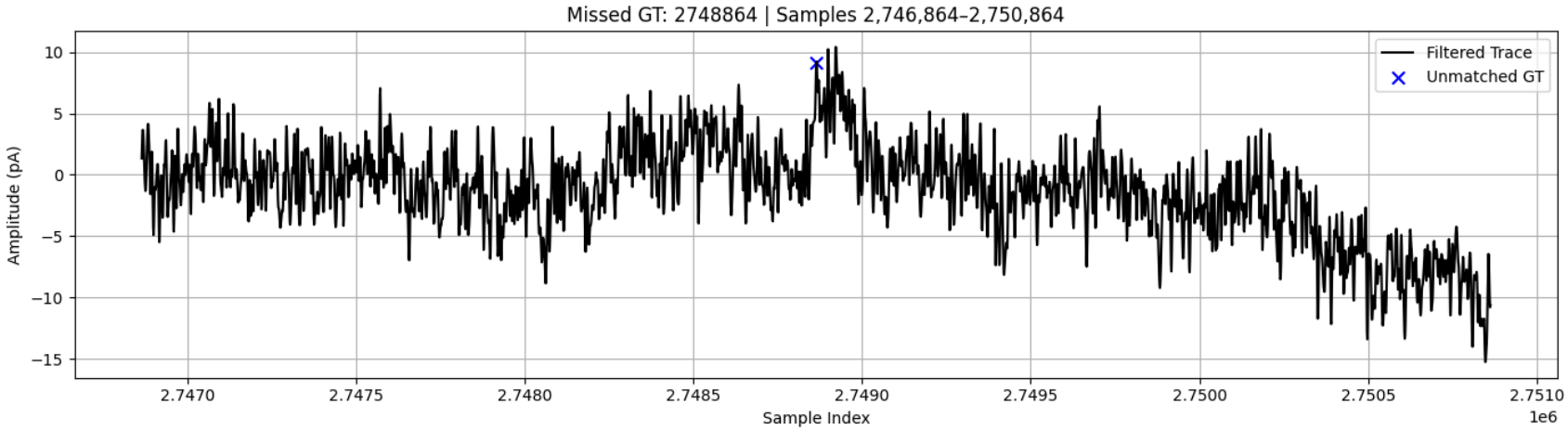

Missed GT: 2405819 | Samples 2,403,819–2,407,819
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
2.4040
2.4045
2.4050
2.4055
2.4060
2.4065
2.4070
2.4075
2.4080
Sample Index
1e6

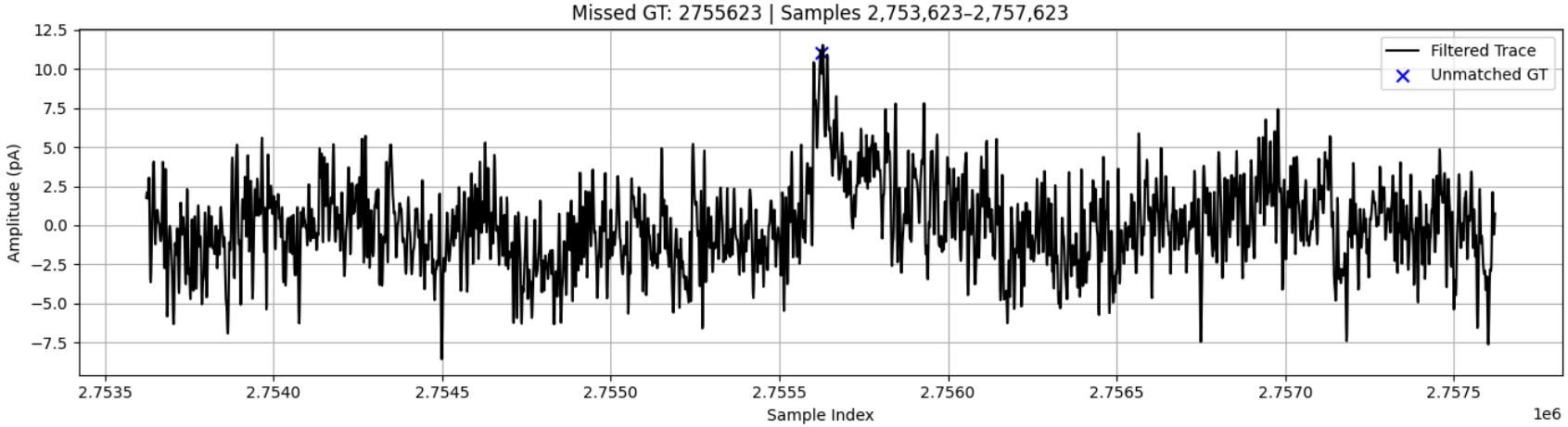

Missed GT: 2606675 | Samples 2,604,675–2,608,675
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
−10.0
2.6045
2.6050
2.6055
2.6060
2.6065
2.6070
2.6075
2.6080
2.6085
Sample Index
1e6

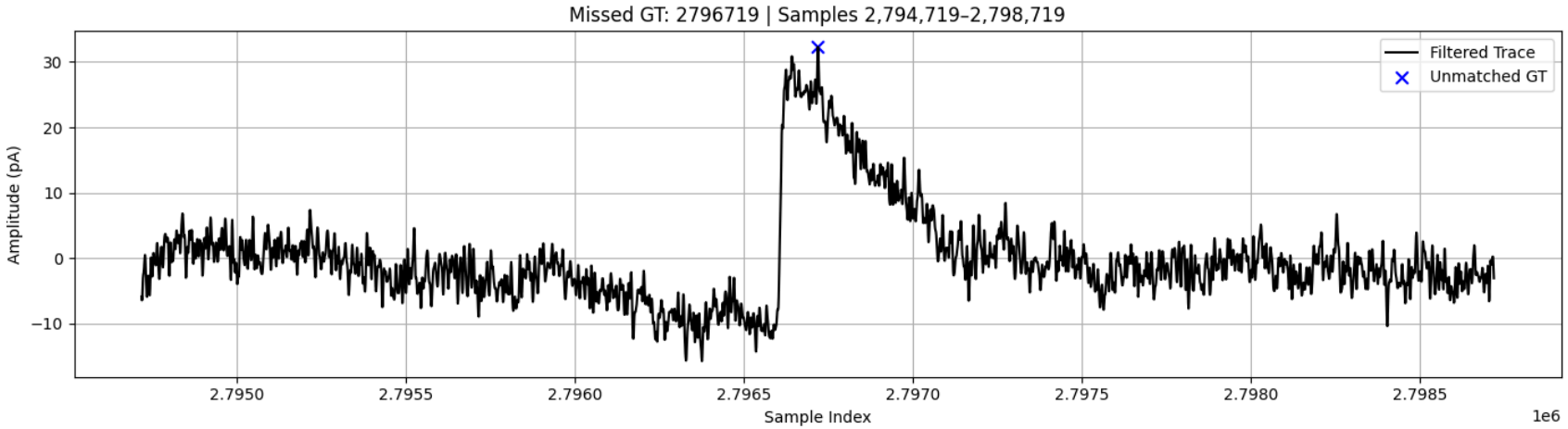

Missed GT: 2670611 | Samples 2,668,611–2,672,611
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
2.6685
2.6690
2.6695
2.6700
2.6705
2.6710
2.6715
2.6720
2.6725
Sample Index
1e6

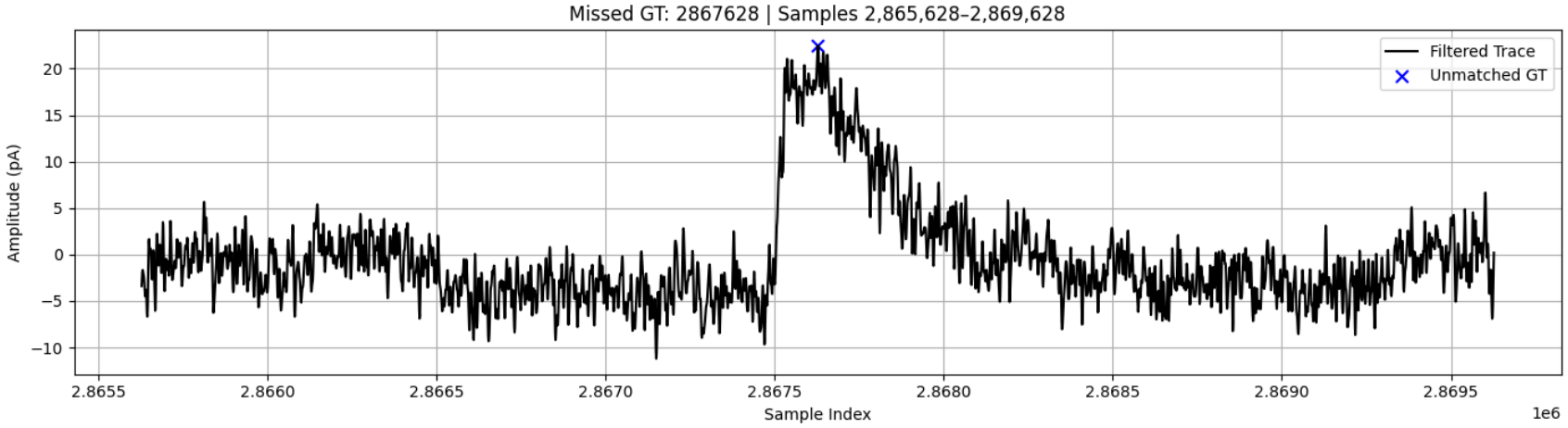

Missed GT: 2730858 | Samples 2,728,858–2,732,858
Filtered Trace
Unmatched GT
Amplitude (pA)
40
30
20
10
0
−10
2.7290
2.7295
2.7300
2.7305
2.7310
2.7315
2.7320
2.7325
2.7330
1e6
Sample Index

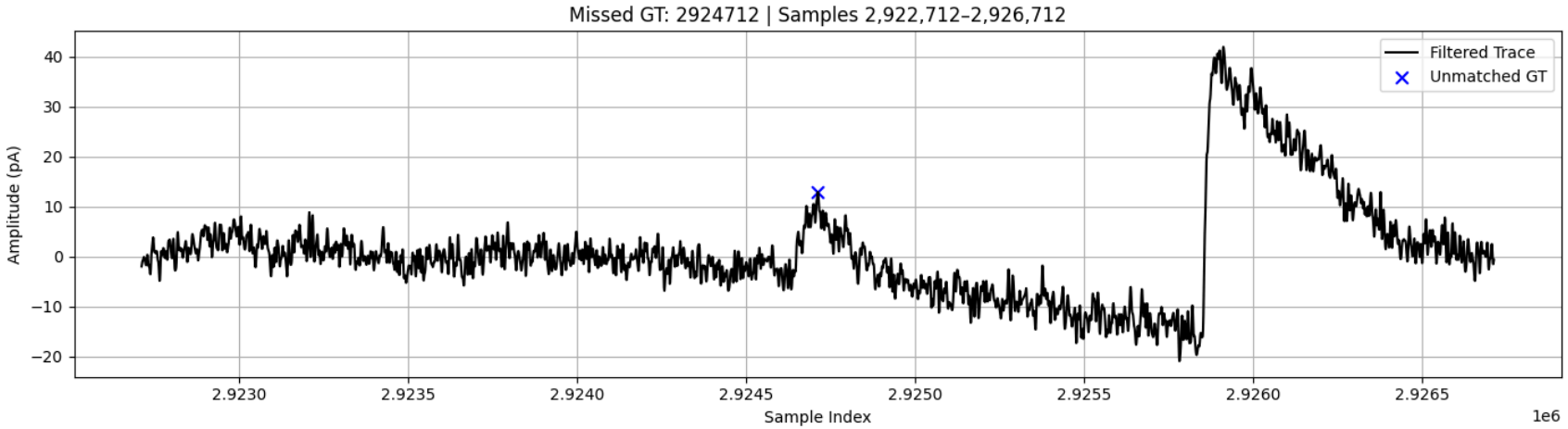

Missed GT: 2746214 | Samples 2,744,214–2,748,214
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
2.7445
2.7450
2.7455
2.7460
2.7465
2.7470
2.7475
2.7480
1e6
Sample Index

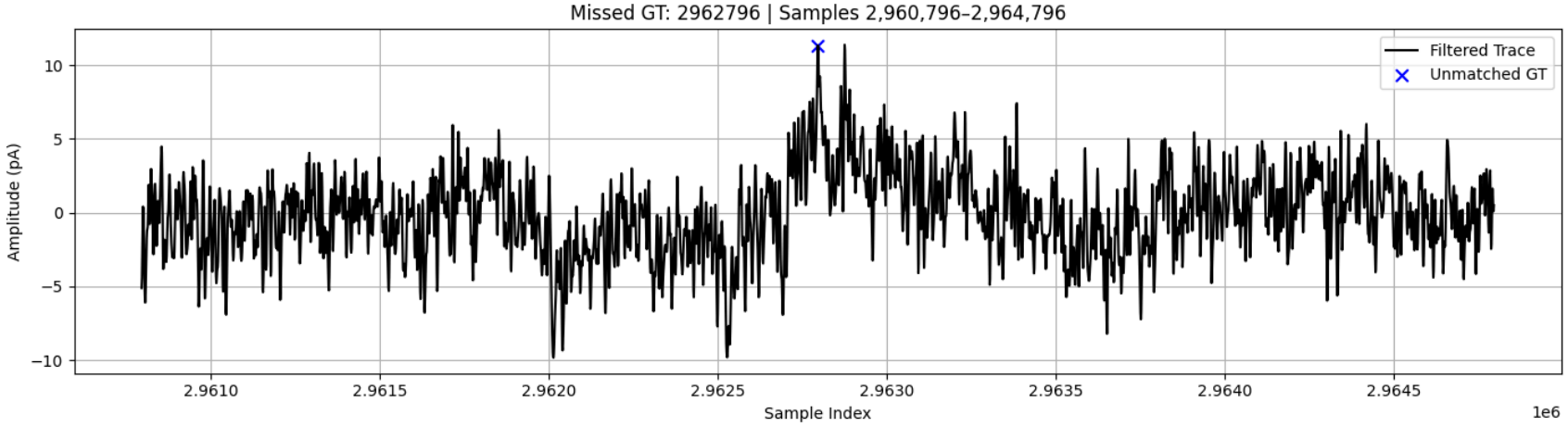

Missed GT: 2748864 | Samples 2,746,864–2,750,864
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
−15
2.7470
2.7475
2.7480
2.7485
2.7490
2.7495
2.7500
2.7505
2.7510
1e6
Sample Index

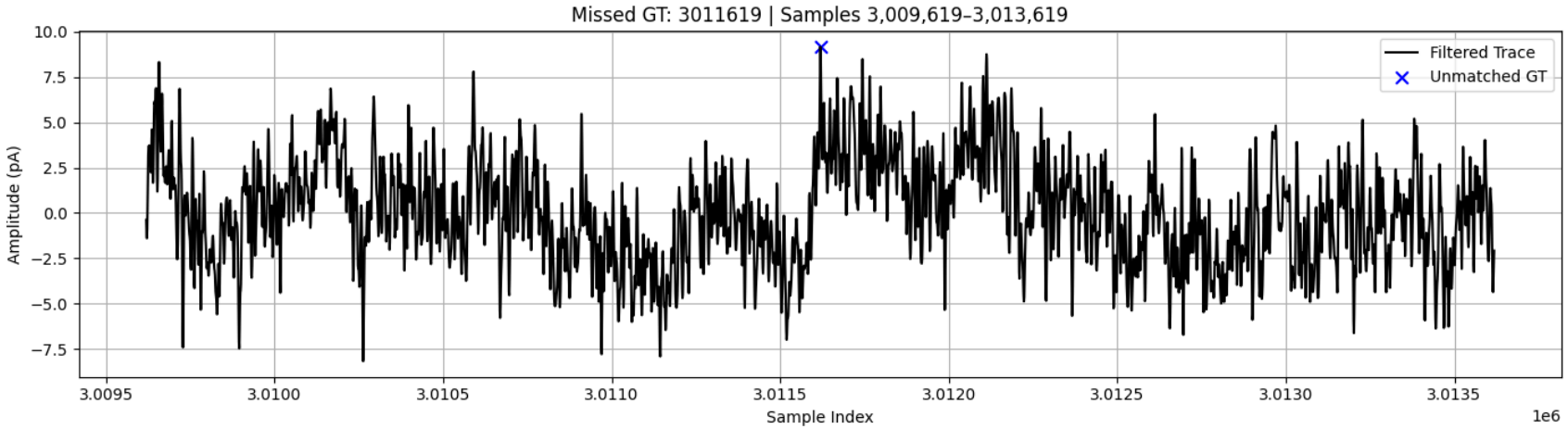

Missed GT: 2755623 | Samples 2,753,623–2,757,623
Filtered Trace
Unmatched GT
Amplitude (pA)
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
2.7535
2.7540
2.7545
2.7550
2.7555
2.7560
2.7565
2.7570
2.7575
1e6
Sample Index

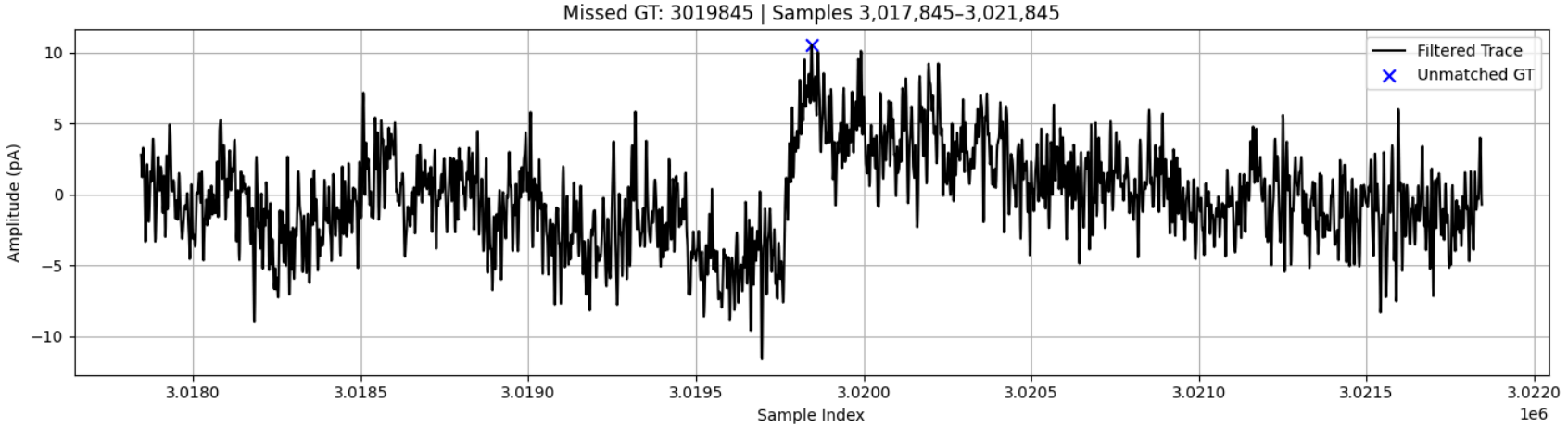

Missed GT: 2796719 | Samples 2,794,719–2,798,719
Filtered Trace
Unmatched GT
Amplitude (pA)
30
20
10
0
−10
2.7950
2.7955
2.7960
2.7965
2.7970
2.7975
2.7980
2.7985
Sample Index
1e6

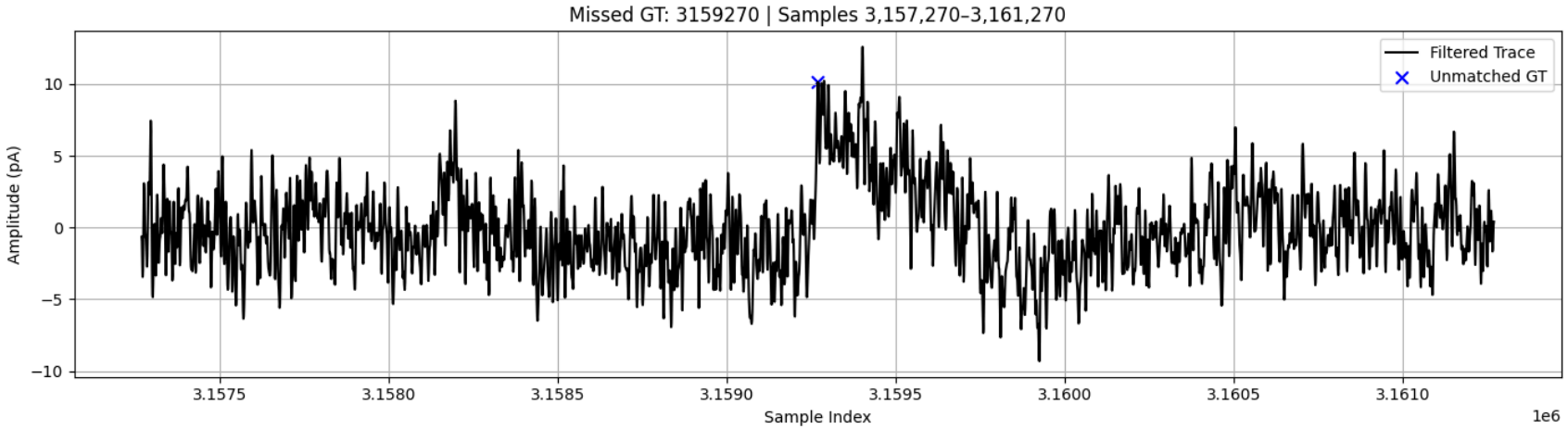

Missed GT: 2867628 | Samples 2,865,628–2,869,628
Filtered Trace
Unmatched GT
Amplitude (pA)
20
15
10
5
0
−5
−10
2.8655
2.8660
2.8665
2.8670
2.8675
2.8680
2.8685
2.8690
2.8695
Sample Index
1e6

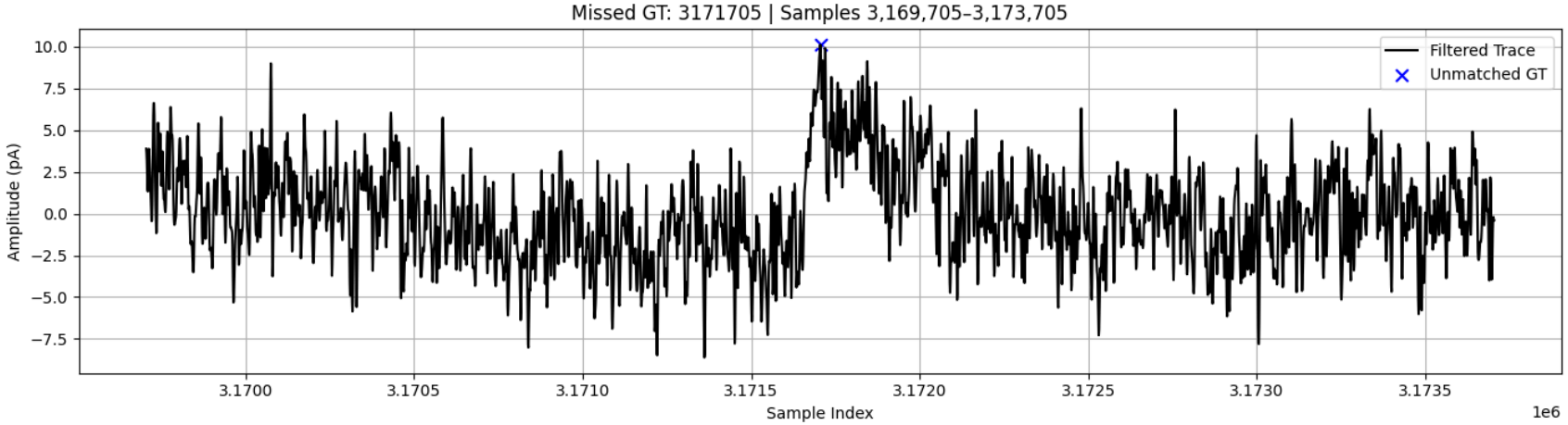

Missed GT: 2924712 | Samples 2,922,712–2,926,712
Filtered Trace
Unmatched GT
Amplitude (pA)
40
30
20
10
0
−10
−20
2.9230
2.9235
2.9240
2.9245
2.9250
2.9255
2.9260
2.9265
Sample Index
1e6

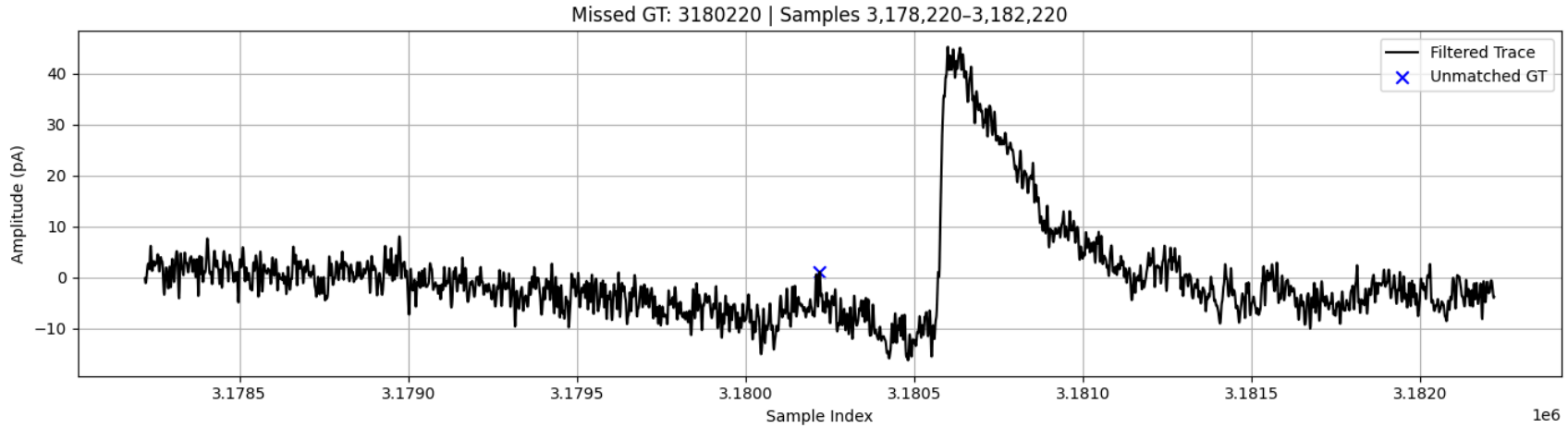

Missed GT: 2962796 | Samples 2,960,796–2,964,796
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
2.9610
2.9615
2.9620
2.9625
2.9630
2.9635
2.9640
2.9645
Sample Index
1e6

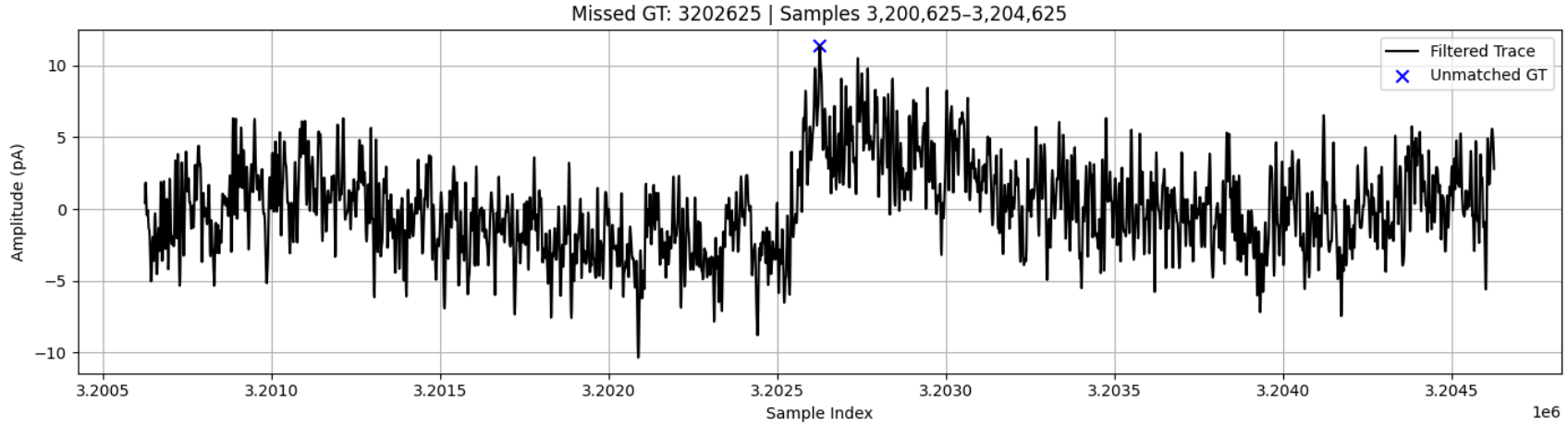

Missed GT: 3011619 | Samples 3,009,619–3,013,619
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
3.0095
3.0100
3.0105
3.0110
3.0115
3.0120
3.0125
3.0130
3.0135
Sample Index
1e6

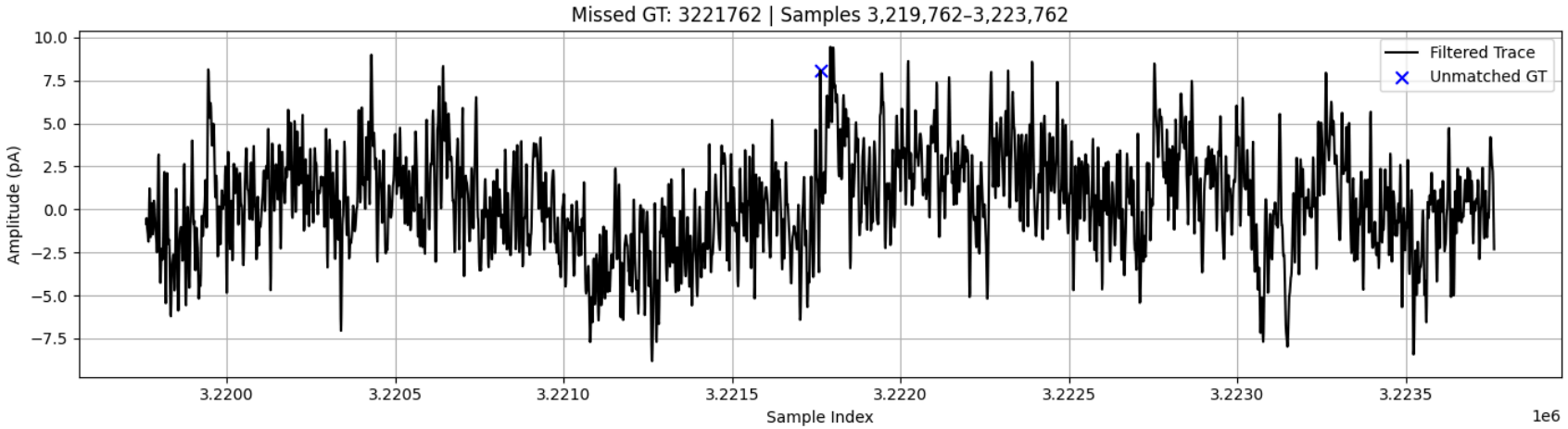

Missed GT: 3019845 | Samples 3,017,845–3,021,845
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
3.0180
3.0185
3.0190
3.0195
3.0200
3.0205
3.0210
3.0215
3.0220
Sample Index
1e6

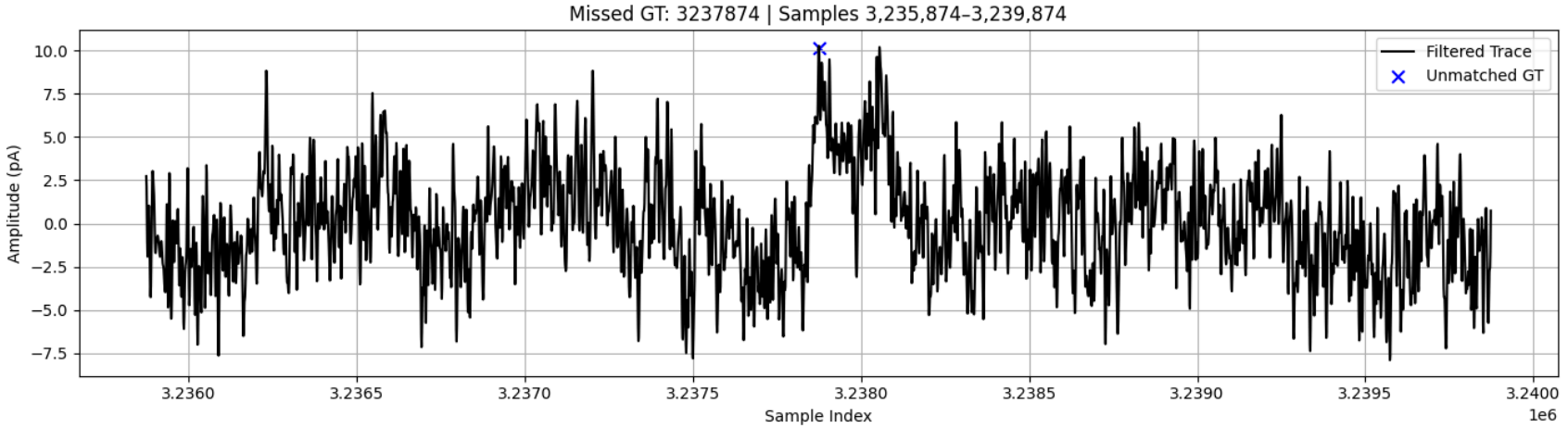

Missed GT: 3159270 | Samples 3,157,270–3,161,270
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
3.1575
3.1580
3.1585
3.1590
3.1595
3.1600
3.1605
3.1610
Sample Index
1e6

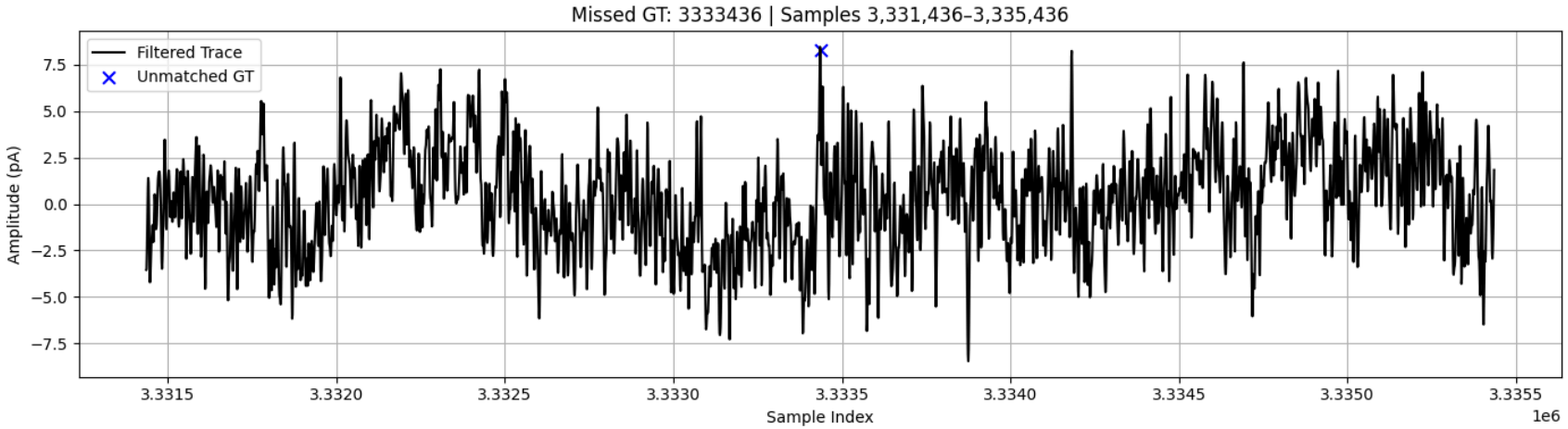

Missed GT: 3171705 | Samples 3,169,705–3,173,705
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
3.1700
3.1705
3.1710
3.1715
3.1720
3.1725
3.1730
3.1735
Sample Index
1e6

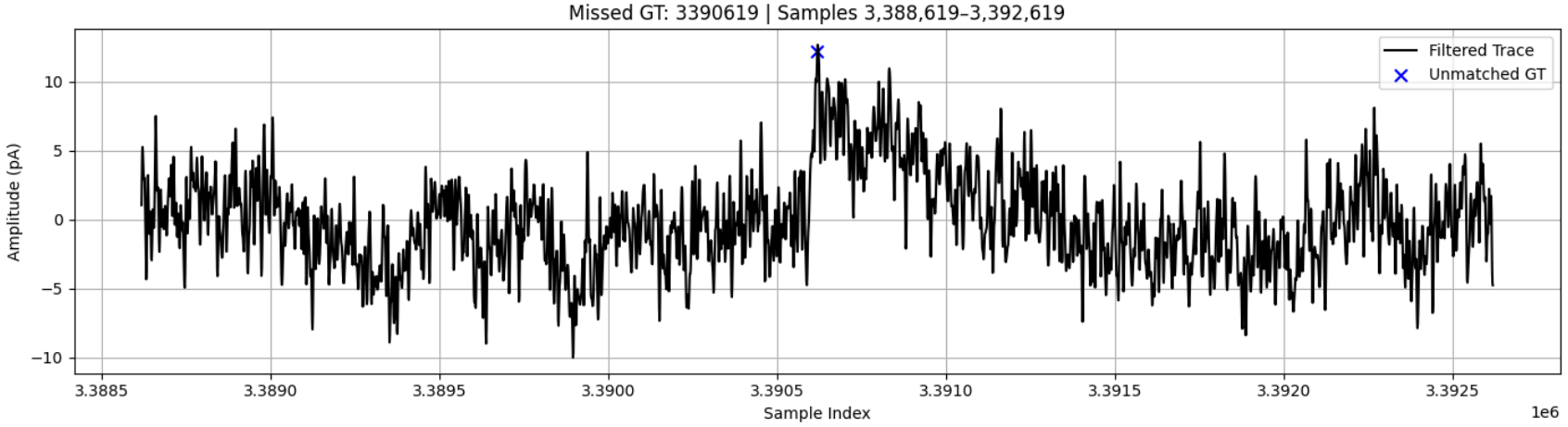

Missed GT: 3180220 | Samples 3,178,220–3,182,220
Filtered Trace
Unmatched GT
Amplitude (pA)
40
30
20
10
0
−10
3.1785
3.1790
3.1795
3.1800
3.1805
3.1810
3.1815
3.1820
Sample Index
1e6

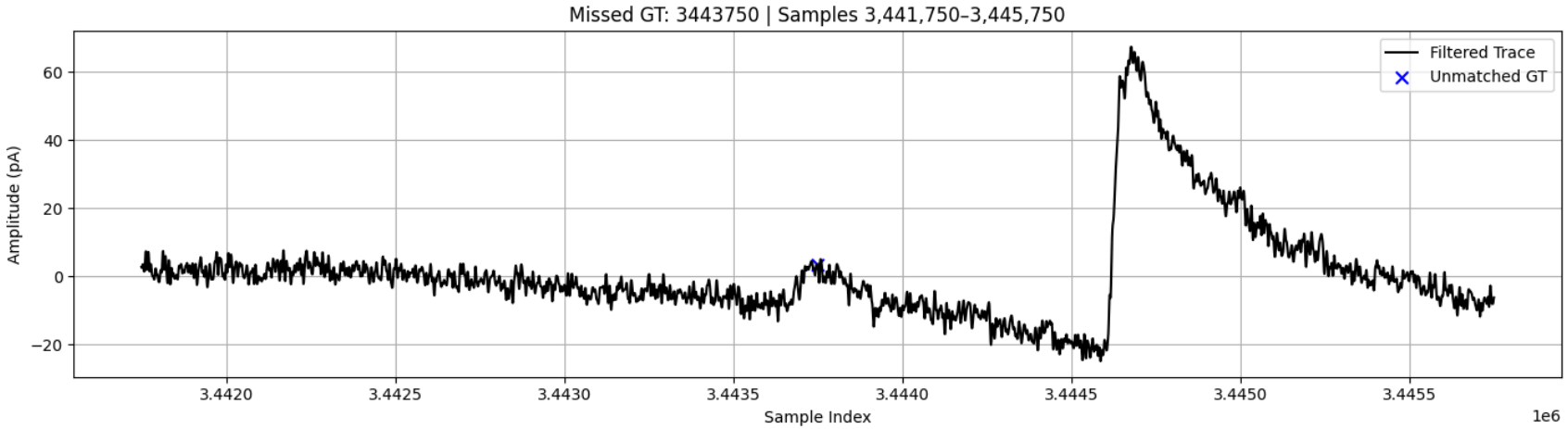

Missed GT: 3202625 | Samples 3,200,625–3,204,625
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
3.2005
3.2010
3.2015
3.2020
3.2025
3.2030
3.2035
3.2040
3.2045
Sample Index
1e6

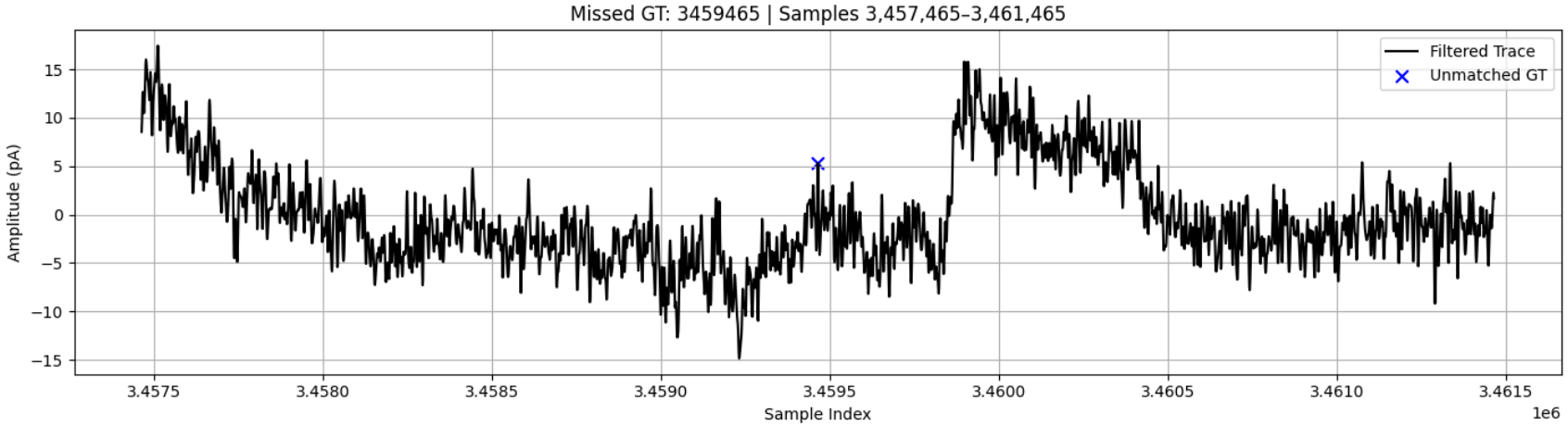

Missed GT: 3221762 | Samples 3,219,762–3,223,762
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
3.2200
3.2205
3.2210
3.2215
3.2220
3.2225
3.2230
3.2235
Sample Index
1e6

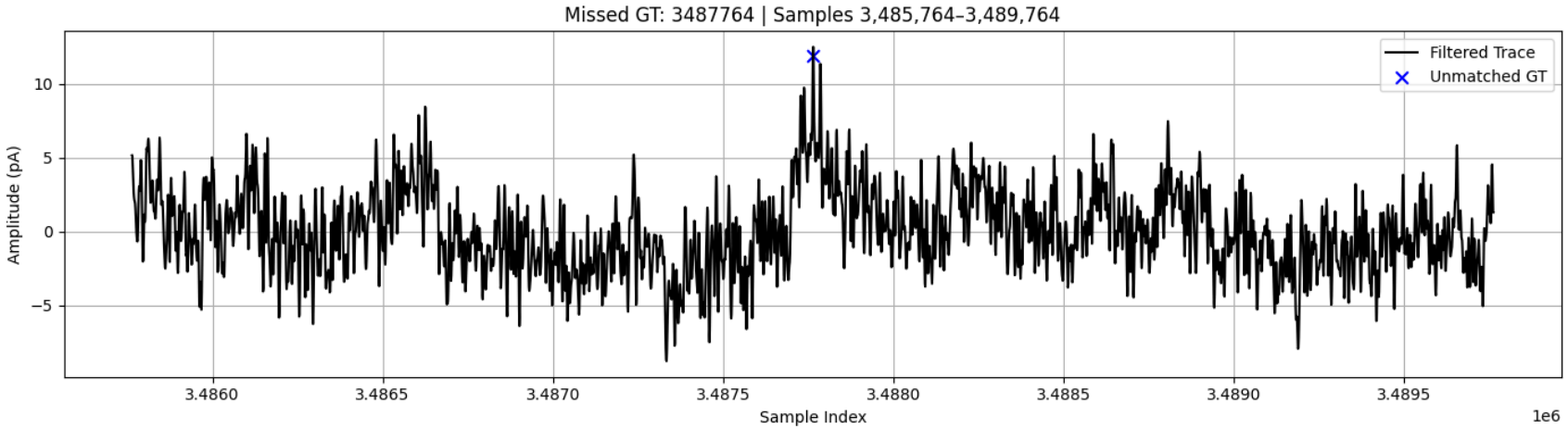

Missed GT: 3237874 | Samples 3,235,874–3,239,874
Filtered Trace
Unmatched GT
Amplitude (pA)
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
3.2360
3.2365
3.2370
3.2375
3.2380
3.2385
3.2390
3.2395
3.2400
Sample Index
1e6

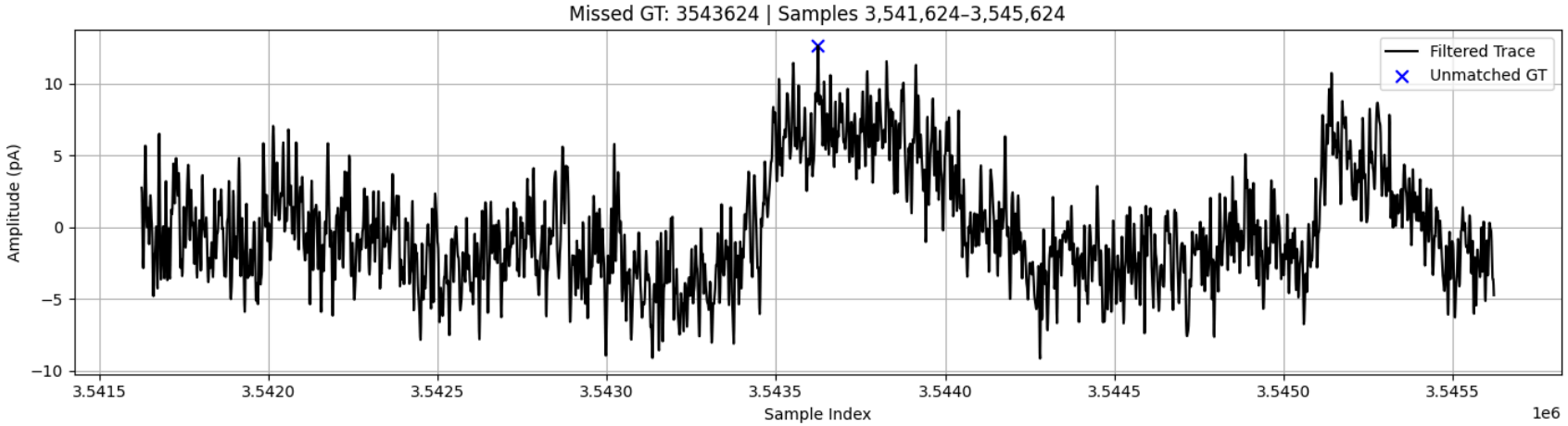

Missed GT: 3333436 | Samples 3,331,436–3,335,436
Filtered Trace
Unmatched GT
Amplitude (pA)
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
3.3315
3.3320
3.3325
3.3330
3.3335
3.3340
3.3345
3.3350
3.3355
Sample Index
1e6

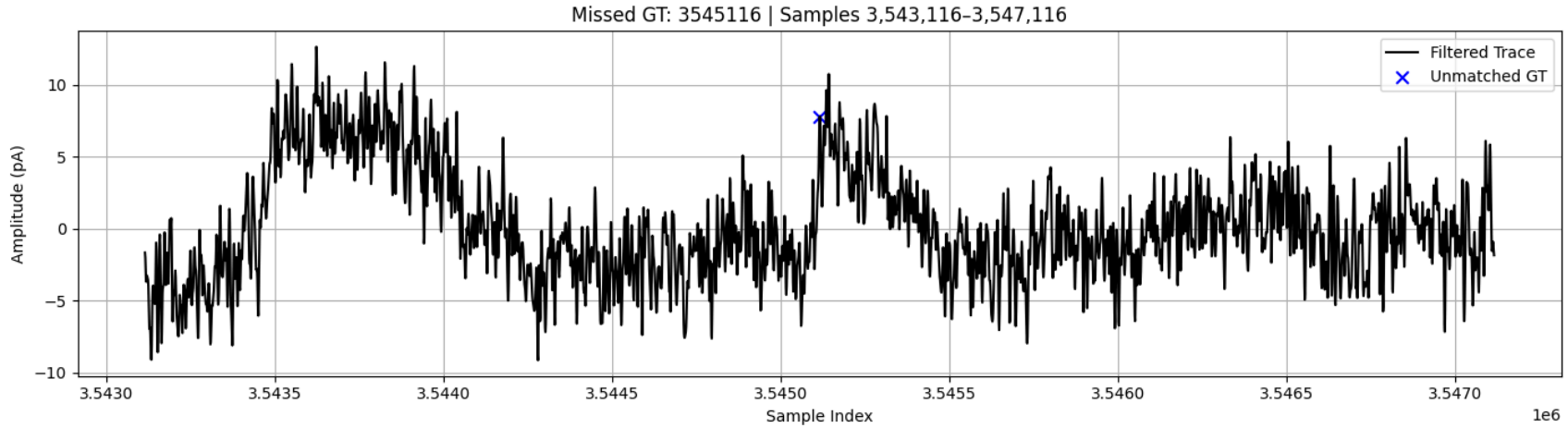

Missed GT: 3390619 | Samples 3,388,619–3,392,619
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
3.3885
3.3890
3.3895
3.3900
3.3905
3.3910
3.3915
3.3920
3.3925
Sample Index
1e6


Missed GT: 3443750 | Samples 3,441,750–3,445,750
Filtered Trace
Unmatched GT
Amplitude (pA)
60
40
20
0
−20
3.4420
3.4425
3.4430
3.4435
3.4440
3.4445
3.4450
3.4455
Sample Index
1e6


Missed GT: 3459465 | Samples 3,457,465–3,461,465
Filtered Trace
Unmatched GT
Amplitude (pA)
15
10
5
0
−5
−10
−15
3.4575
3.4580
3.4585
3.4590
3.4595
3.4600
3.4605
3.4610
3.4615
Sample Index
1e6


Missed GT: 3487764 | Samples 3,485,764–3,489,764
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
3.4860
3.4865
3.4870
3.4875
3.4880
3.4885
3.4890
3.4895
Sample Index
1e6

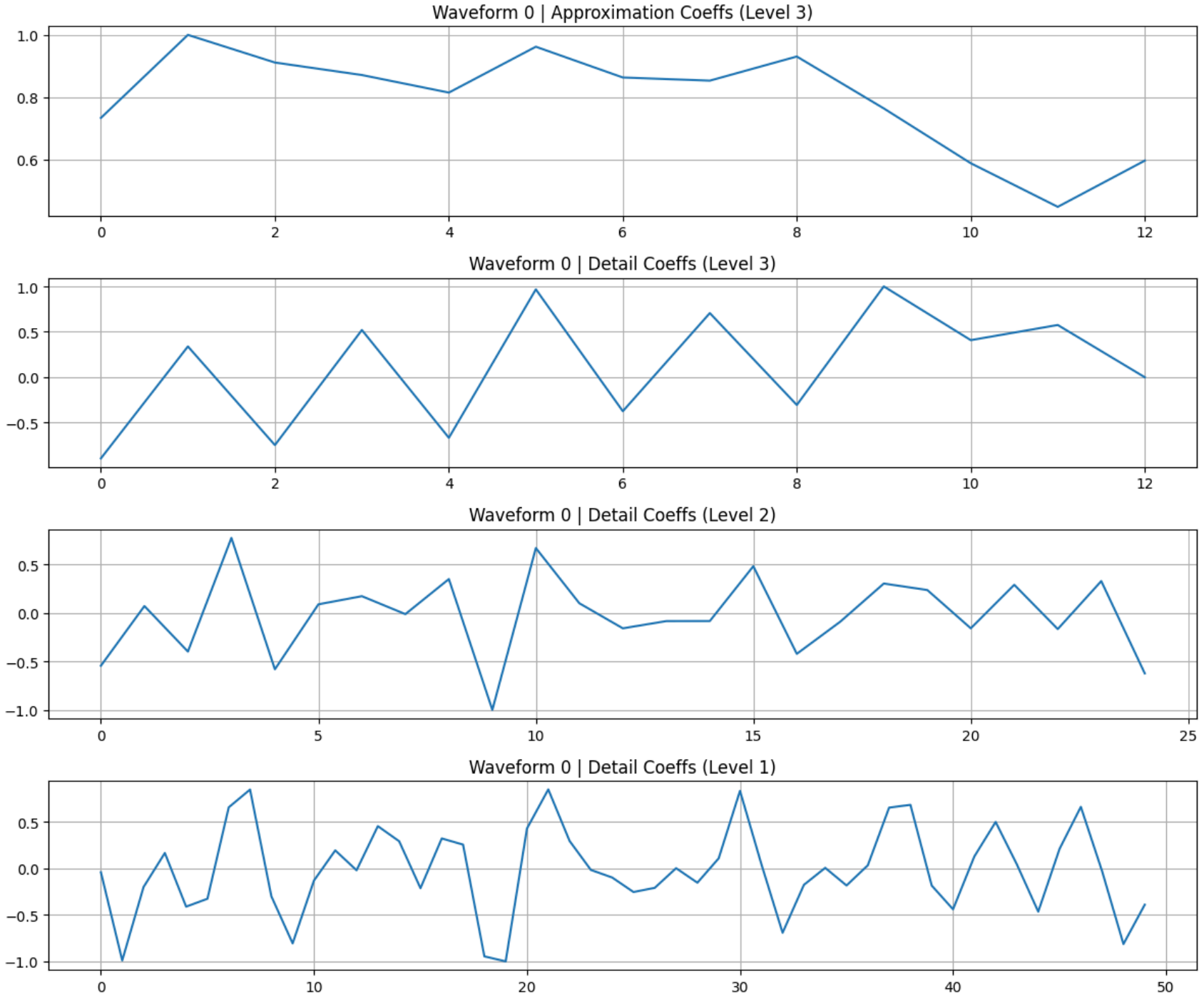

Missed GT: 3543624 | Samples 3,541,624–3,545,624
Filtered Trace
Unmatched GT
Amplitude (pA)
10
5
0
−5
−10
3.5415
3.5420
3.5425
3.5430
3.5435
3.5440
3.5445
3.5450
3.5455
Sample Index
1e6

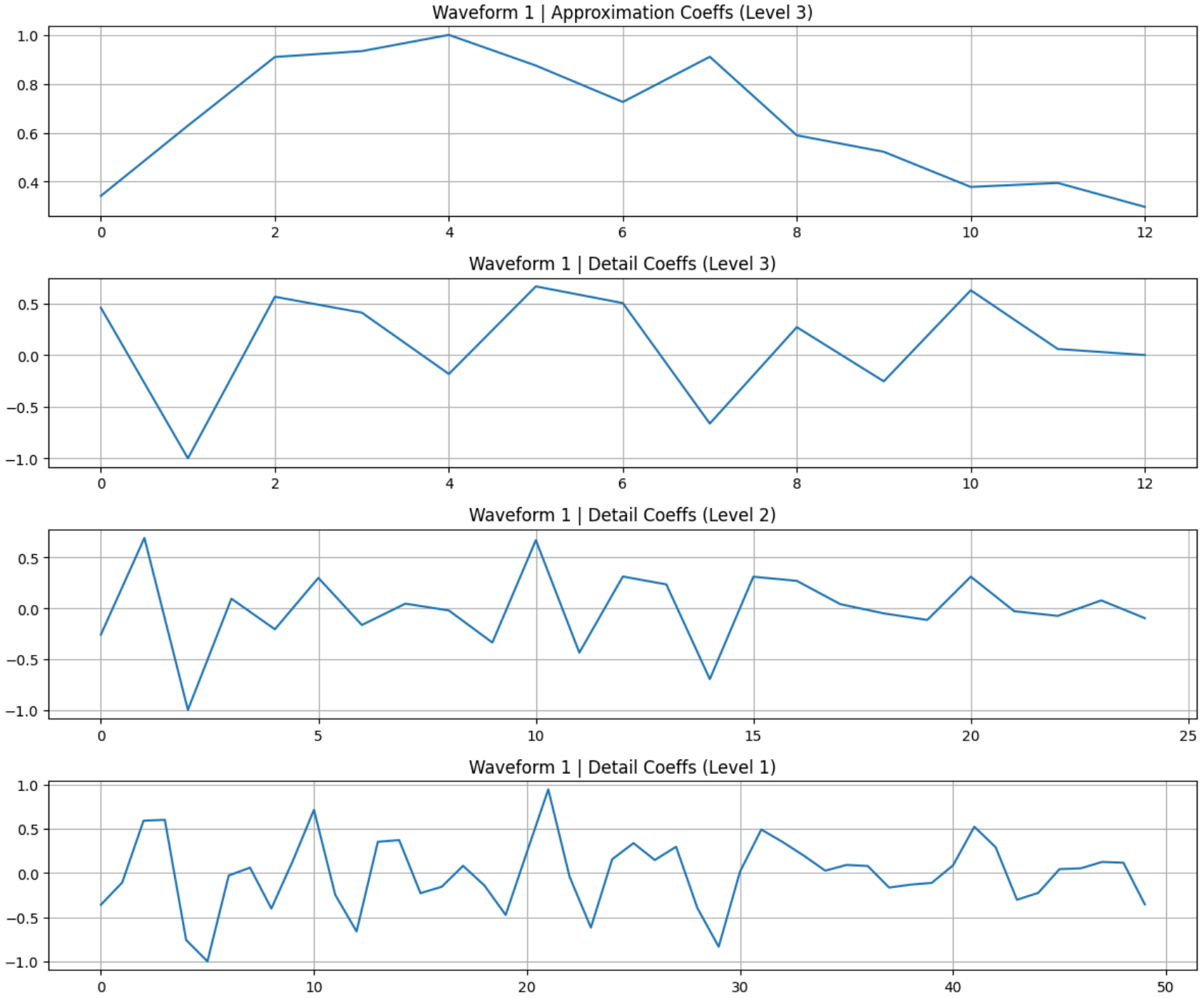


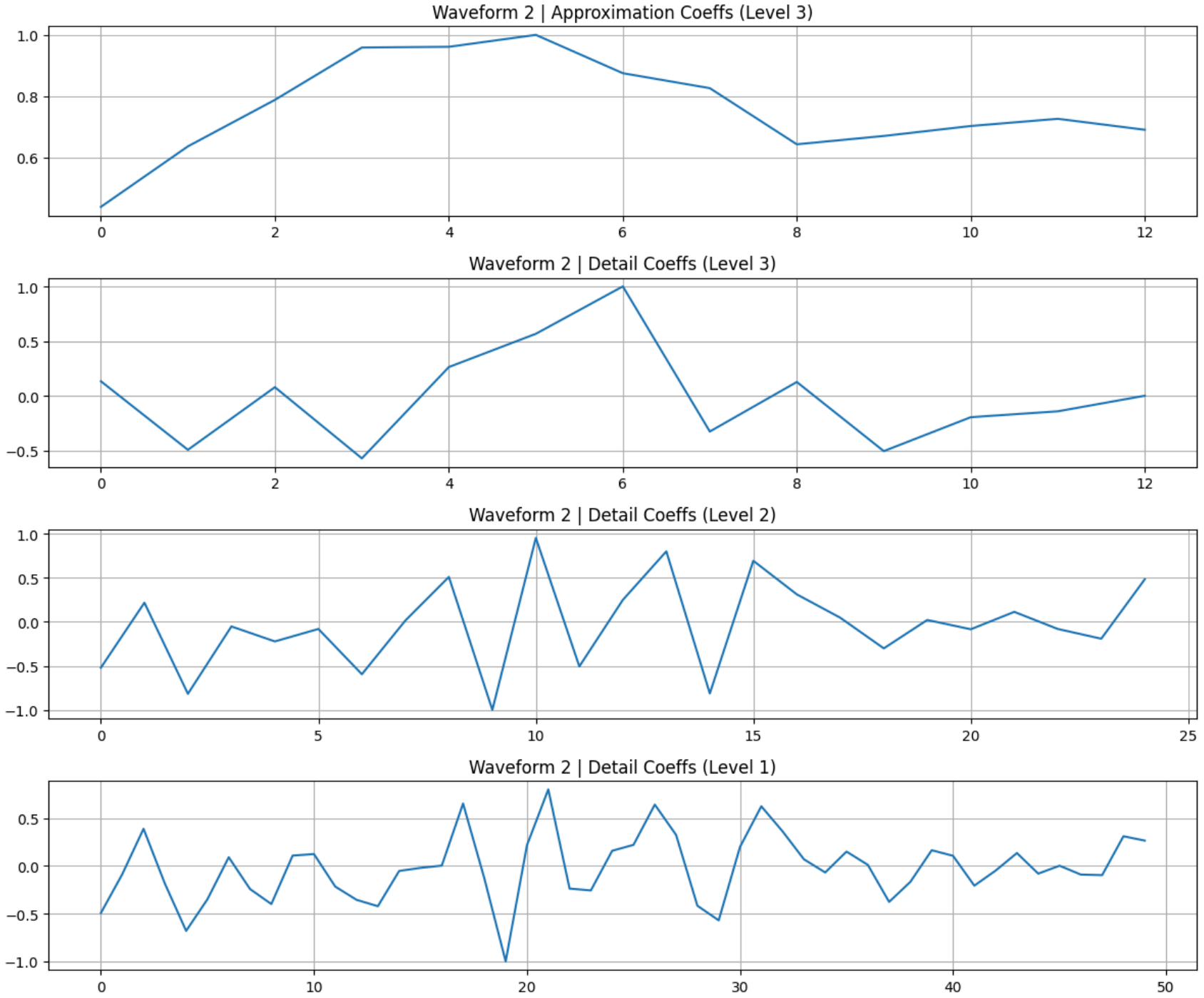
In [ ]:
```python
# Function to visualize wavelet decomposition coefficients of a 1D signal (e.g., waveform) IGNRORE : FOR DATA EXPLORI
def visualize_wavelet_coefficients(
    waveform,
    wavelet='db4',             # Type of wavelet (e.g., 'db4', 'haar')
    level=3,                   # Number of decomposition levels
    title_prefix="Waveform",  # Title prefix for plotting
    normalize_coeffs=False,   # Whether to normalize each set of coefficients
    return_coeffs=False       # Whether to return the coefficient arrays
):
    """
    Visualizes wavelet decomposition coefficients across levels.

    Parameters:
    - waveform: 1D array of the signal
    - wavelet: string, type of wavelet (e.g., 'db4', 'haar', etc.)
    - level: int, number of decomposition levels
    - title_prefix: str, prefix for plot titles
    - normalize_coeffs: bool, whether to normalize each coefficient array for comparison
    - return_coeffs: bool, if True, returns the list of coefficients

    Returns:
    - coeffs (optional): list of arrays [Approximation, Detail L3, ..., Detail L1]
    """

    # Perform wavelet decomposition → returns [A3, D3, D2, D1]
    coeffs = pywt.wavedec(waveform, wavelet=wavelet, level=level)
```

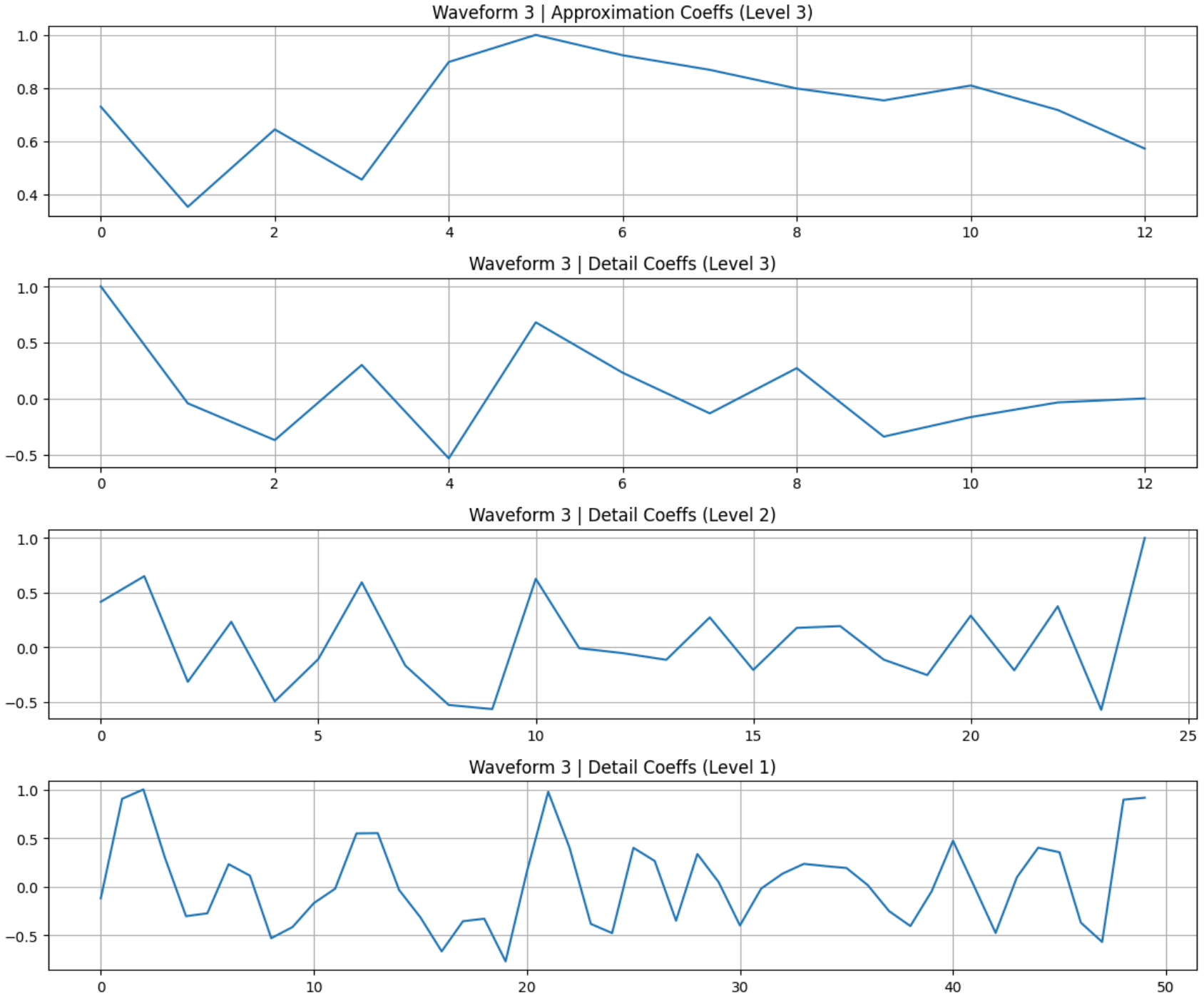
```
    # Create a tall figure with enough space for each level
    plt.figure(figsize=(12, 2.5 * (level + 1)))

    # Loop through each level's coefficients
    for i, c in enumerate(coeffs):
        # Optionally normalize each coefficient array (good for visual comparisons)
        if normalize_coeffs:
            c = c / np.max(np.abs(c)) if np.max(np.abs(c)) != 0 else c

        # Plot each coefficient array in a separate subplot
        plt.subplot(level + 1, 1, i + 1)
        plt.plot(c, linewidth=1.5)

        # Title logic: first element is Approximation, rest are Detail levels
        if i == 0:
            plt.title(f"{title_prefix} | Approximation Coeffs (Level {level})")
        else:
            plt.title(f"{title_prefix} | Detail Coeffs (Level {level - i + 1})")

        plt.grid(True)

    # Adjust spacing between subplots
    plt.tight_layout()
    plt.show()

    # Return coefficients if requested
    if return_coeffs:
        return coeffs


# Display how many waveforms you have
print(f"number of normalized waveforms: {len(normalized_waveforms)}")

# Visualize wavelet decomposition for specific waveforms by index
for idx in [0, 1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21, 22]:  # You can change this list to any wavefor
    visualize_wavelet_coefficients(
        waveform=normalized_waveforms[idx],
        wavelet='haar',
        level=3, # Level of details
        title_prefix=f"Waveform {idx}",
        normalize_coeffs=True
    )
```

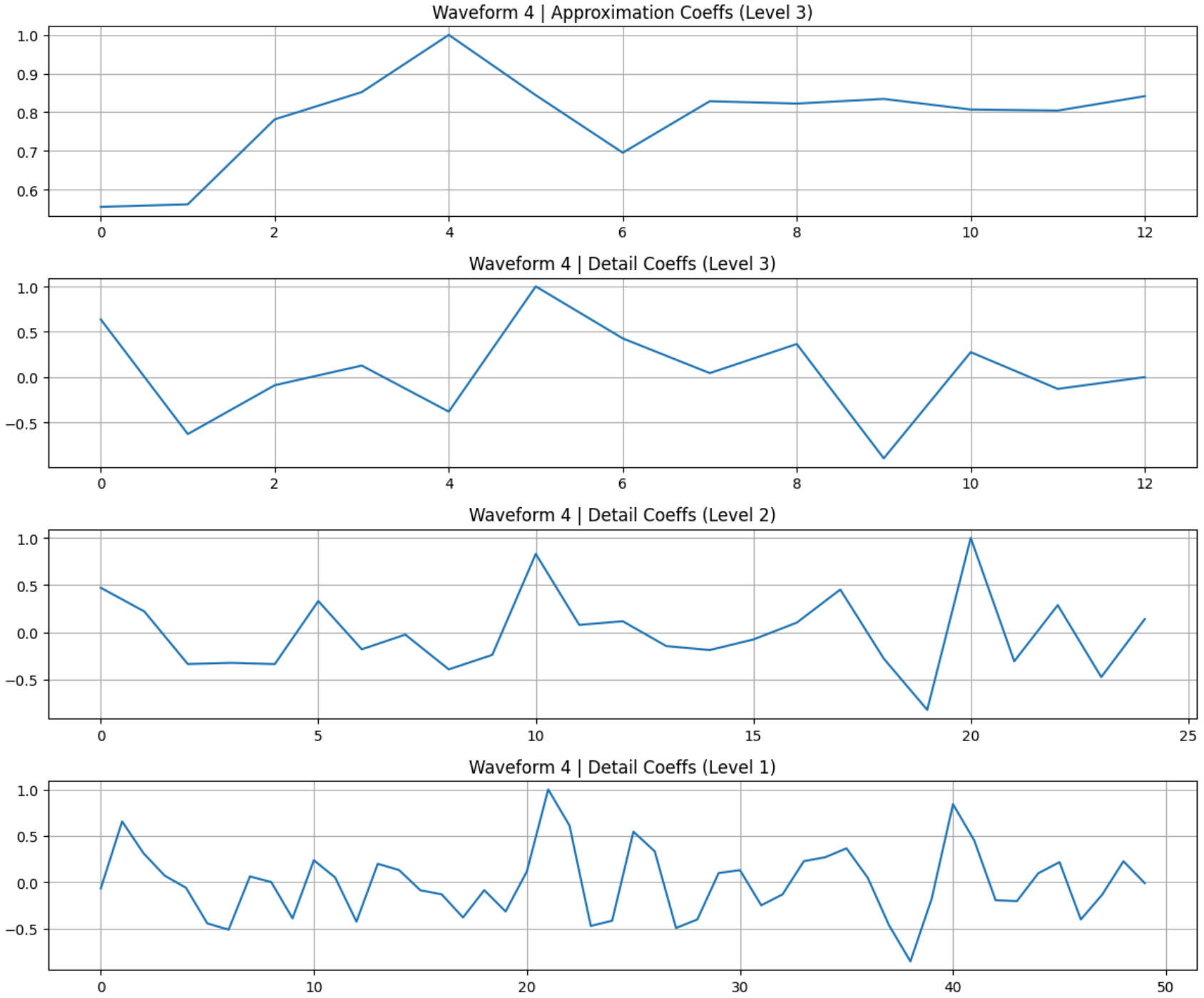
```
number of normalized waveforms: 367
```

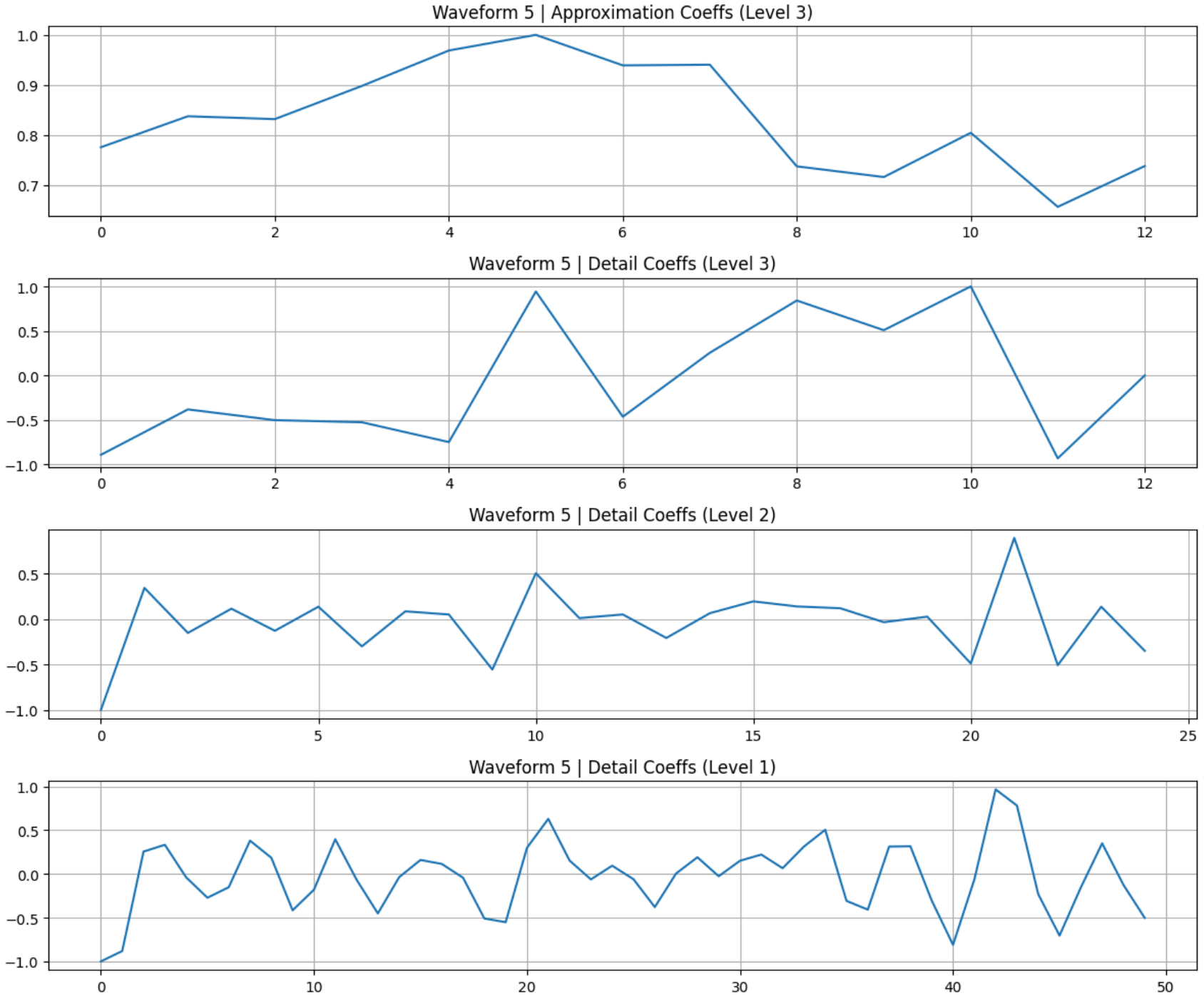

Waveform 0 | Approximation Coeffs (Level 3)
Waveform 0 | Detail Coeffs (Level 3)
Waveform 0 | Detail Coeffs (Level 2)
Waveform 0 | Detail Coeffs (Level 1)

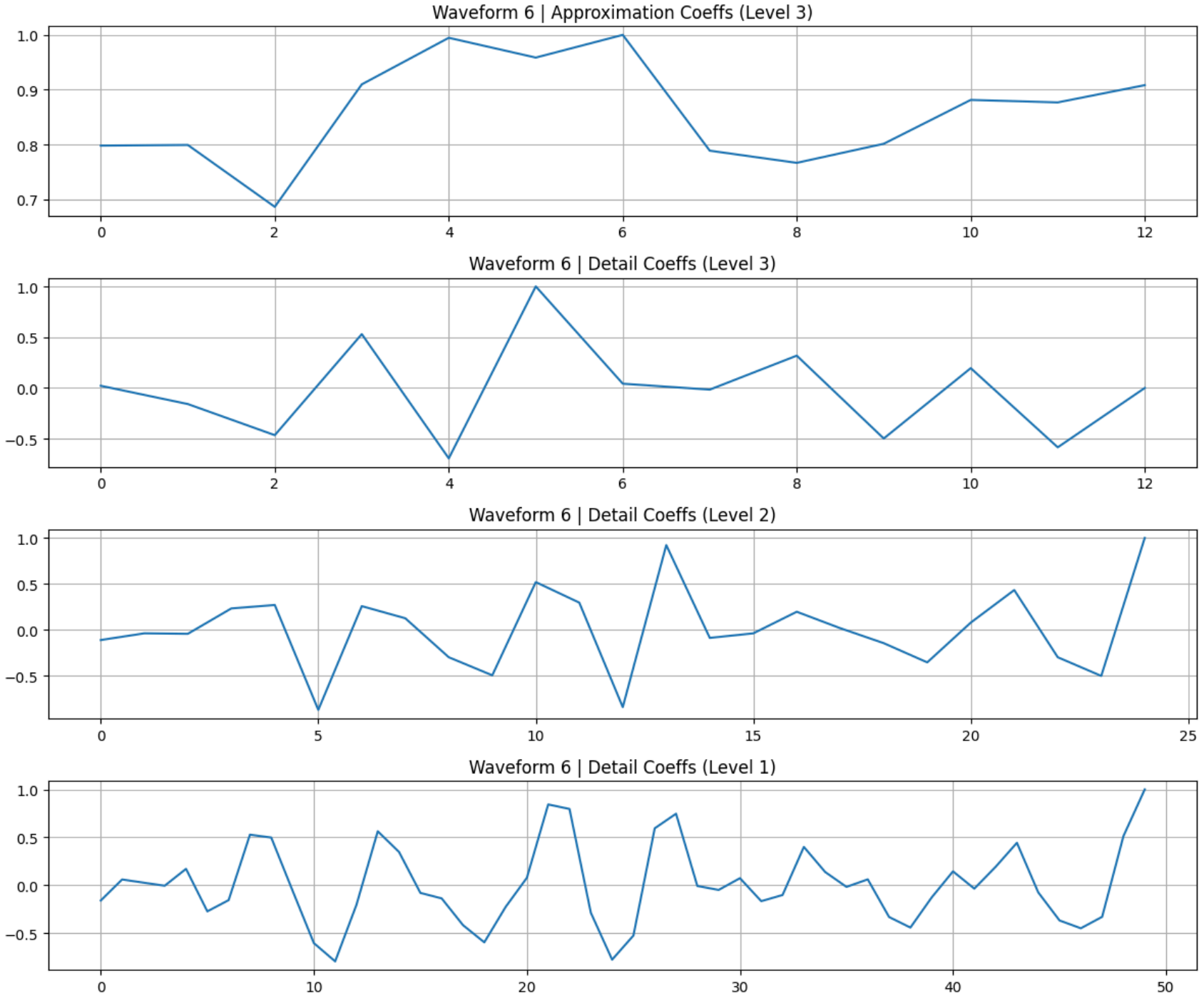

Waveform 1 | Approximation Coeffs (Level 3)
Waveform 1 | Detail Coeffs (Level 3)
Waveform 1 | Detail Coeffs (Level 2)
Waveform 1 | Detail Coeffs (Level 1)

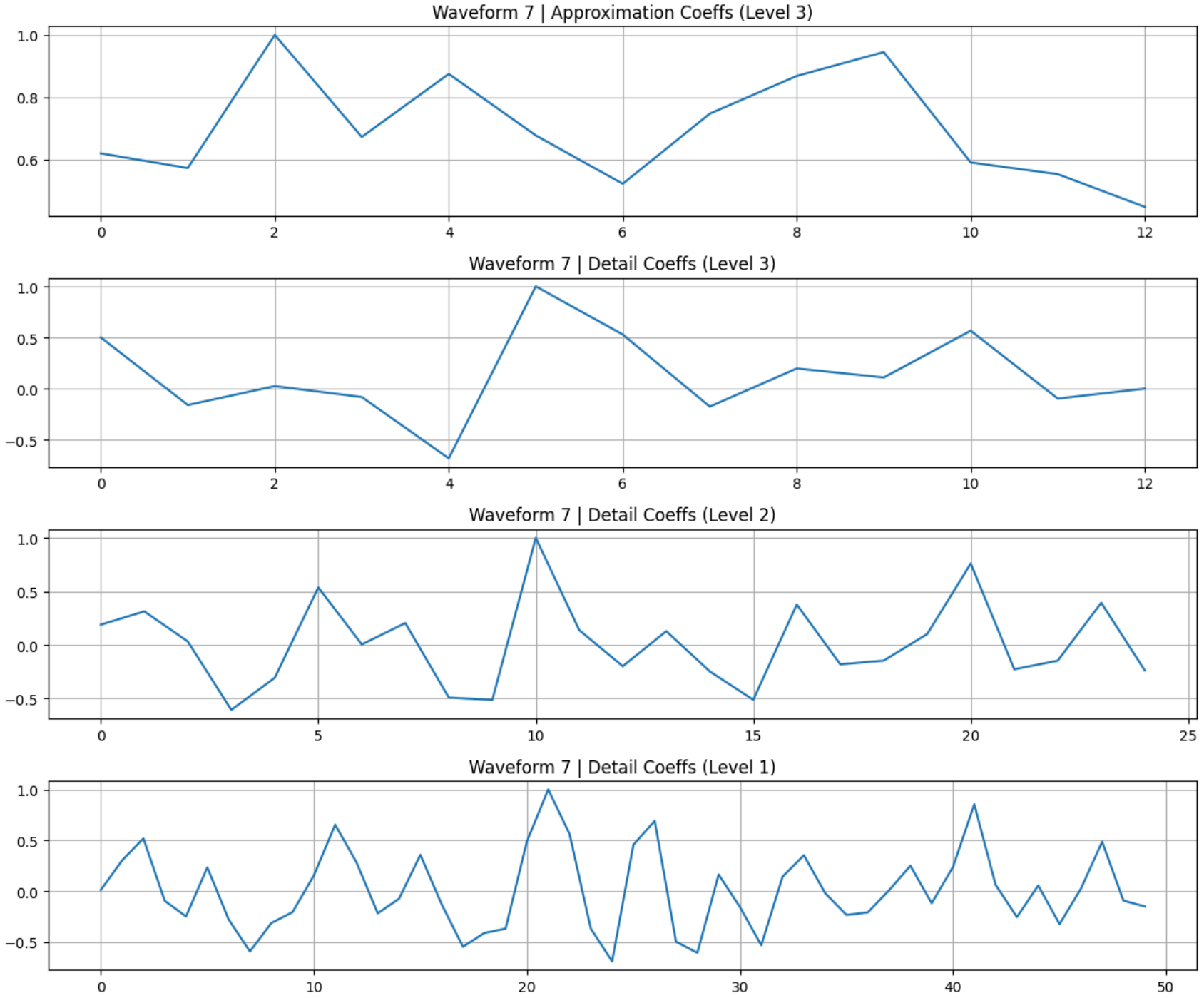

Waveform 2 | Approximation Coeffs (Level 3)
Waveform 2 | Detail Coeffs (Level 3)
Waveform 2 | Detail Coeffs (Level 2)
Waveform 2 | Detail Coeffs (Level 1)

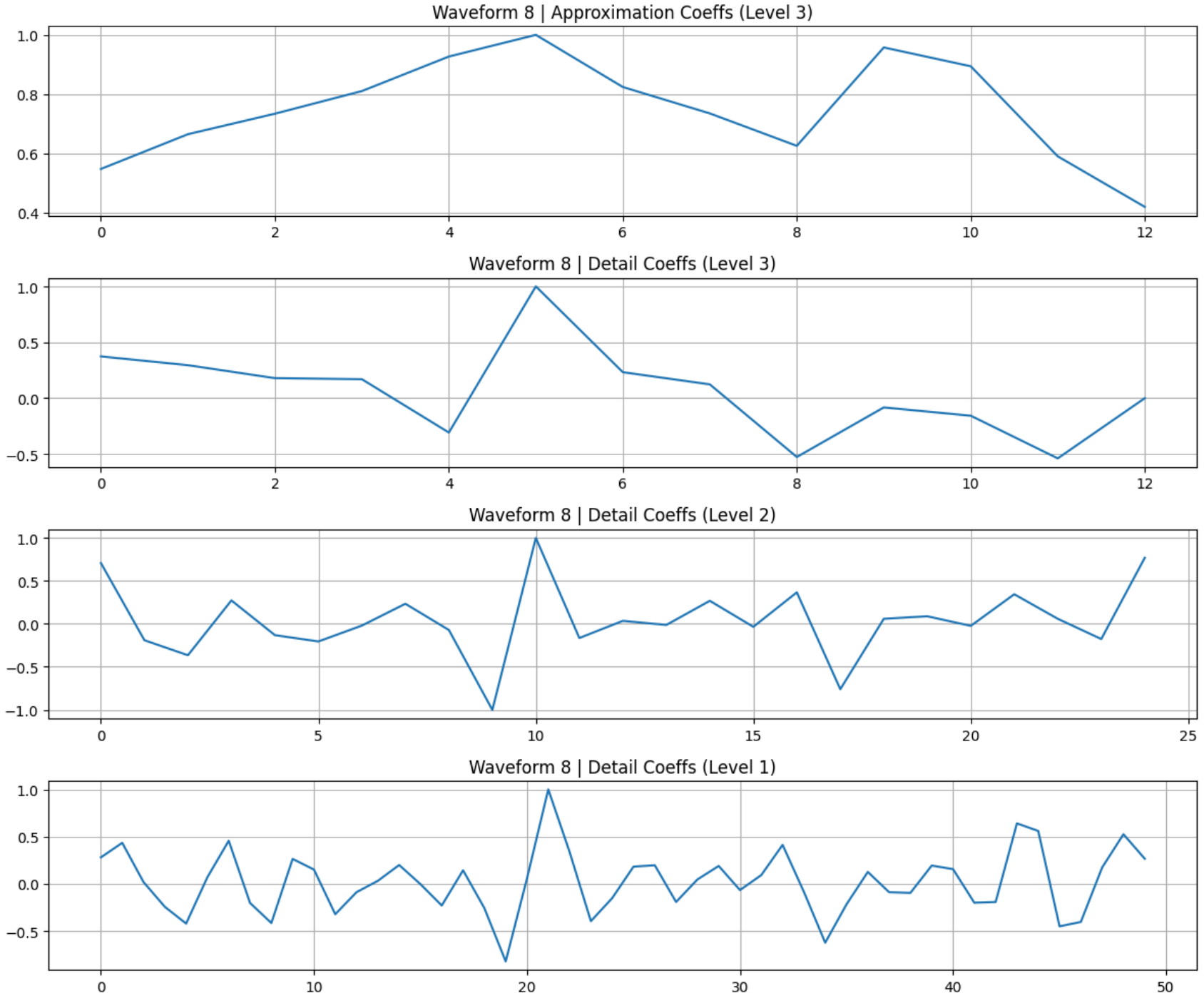

Waveform 3 | Approximation Coeffs (Level 3)
Waveform 3 | Detail Coeffs (Level 3)
Waveform 3 | Detail Coeffs (Level 2)
Waveform 3 | Detail Coeffs (Level 1)

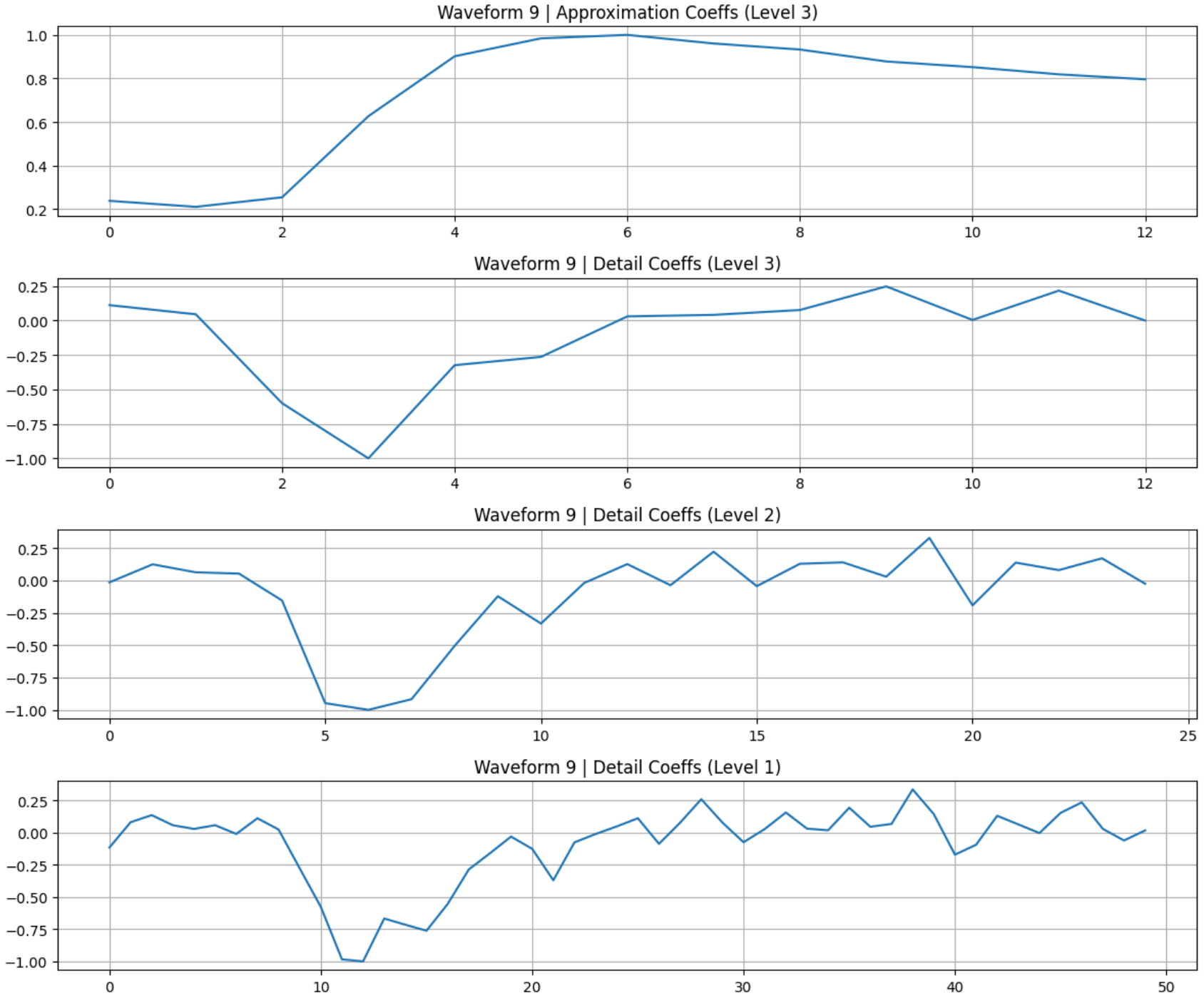

Waveform 4 | Approximation Coeffs (Level 3)
Waveform 4 | Detail Coeffs (Level 3)
Waveform 4 | Detail Coeffs (Level 2)
Waveform 4 | Detail Coeffs (Level 1)

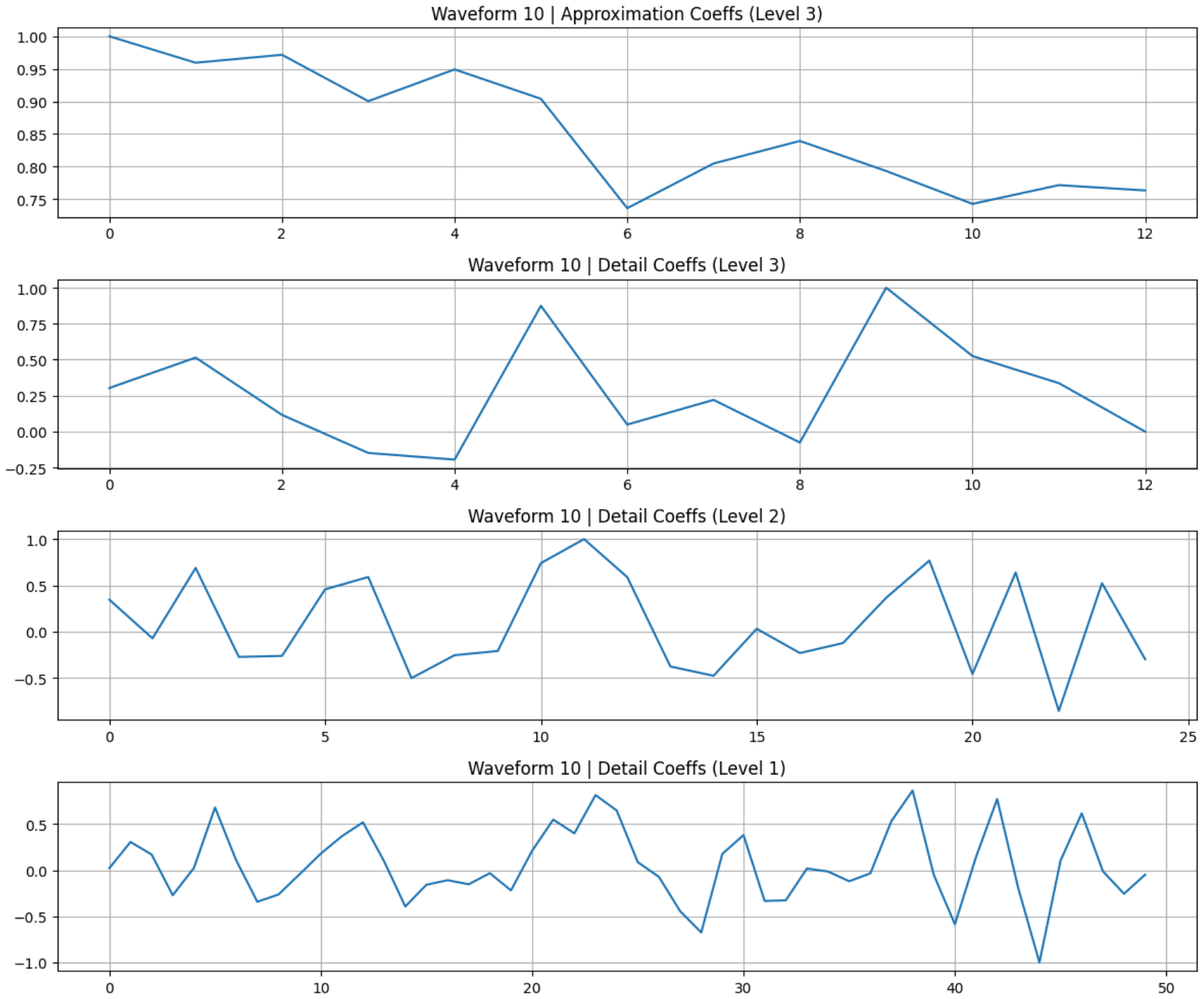

Waveform 5 | Approximation Coeffs (Level 3)
Waveform 5 | Detail Coeffs (Level 3)
Waveform 5 | Detail Coeffs (Level 2)
Waveform 5 | Detail Coeffs (Level 1)

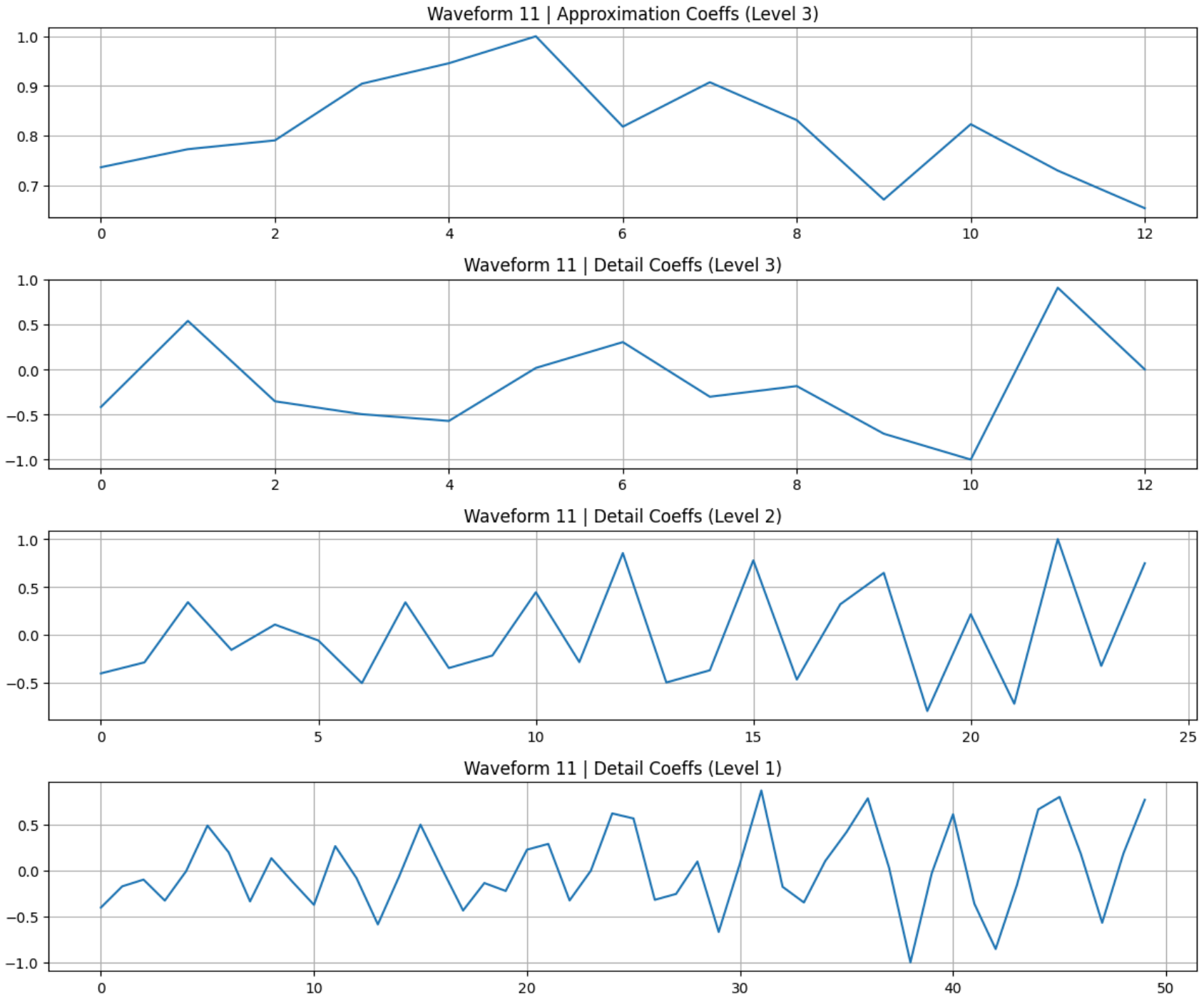

Waveform 6 | Approximation Coeffs (Level 3)
Waveform 6 | Detail Coeffs (Level 3)
Waveform 6 | Detail Coeffs (Level 2)
Waveform 6 | Detail Coeffs (Level 1)

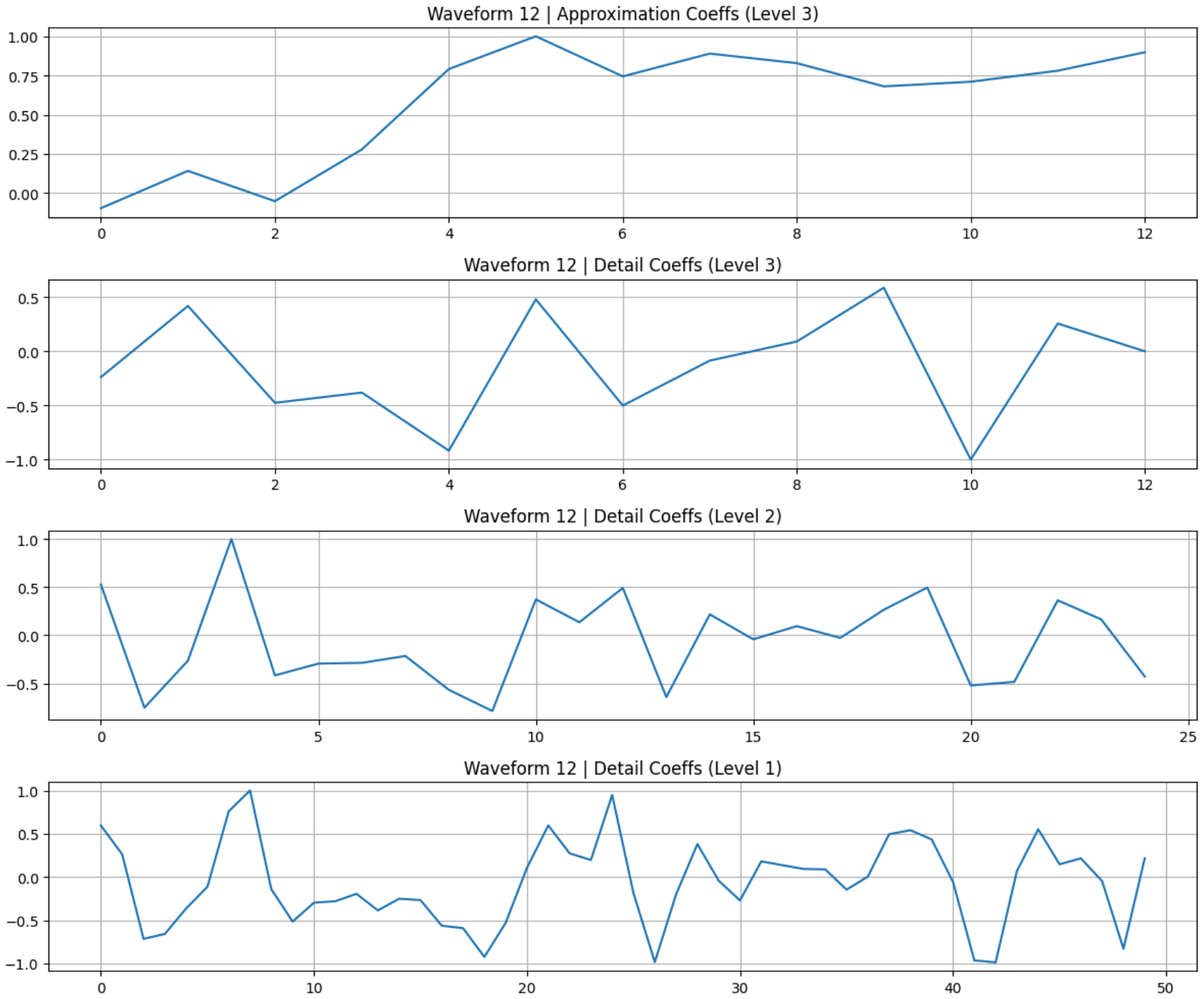

Waveform 7 | Approximation Coeffs (Level 3)
Waveform 7 | Detail Coeffs (Level 3)
Waveform 7 | Detail Coeffs (Level 2)
Waveform 7 | Detail Coeffs (Level 1)

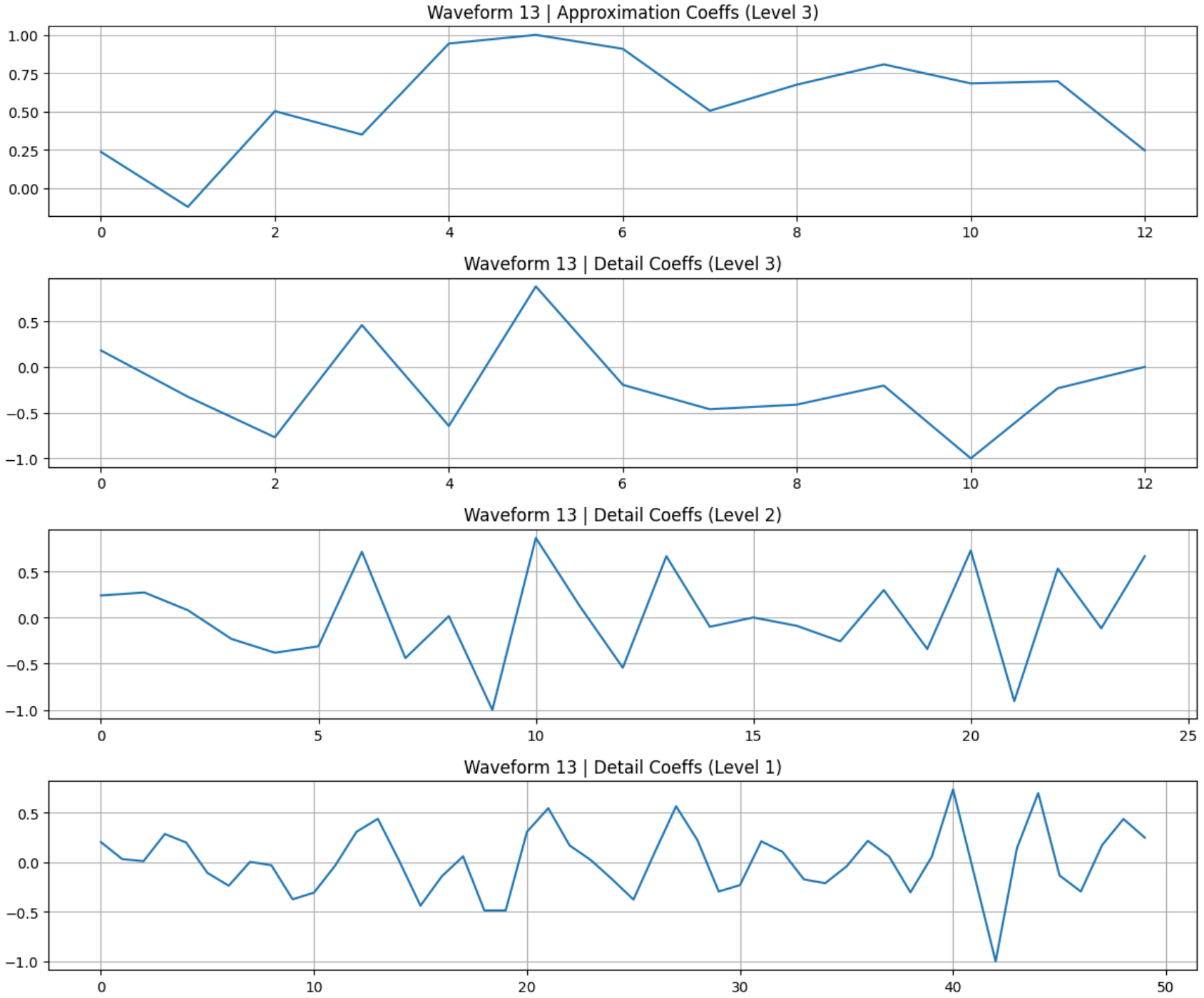

Waveform 8 | Approximation Coeffs (Level 3)
Waveform 8 | Detail Coeffs (Level 3)
Waveform 8 | Detail Coeffs (Level 2)
Waveform 8 | Detail Coeffs (Level 1)

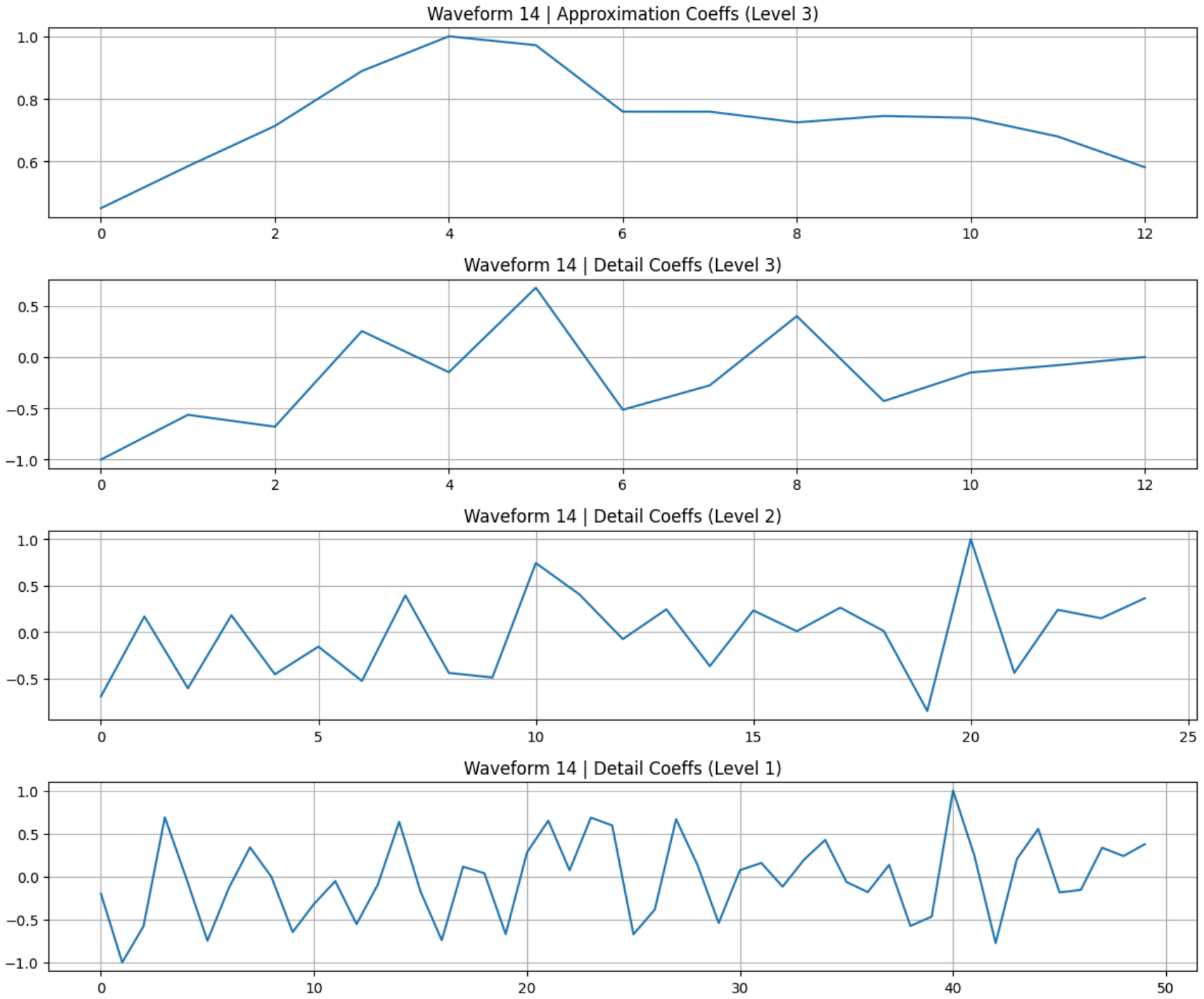

Waveform 9 | Approximation Coeffs (Level 3)
Waveform 9 | Detail Coeffs (Level 3)
Waveform 9 | Detail Coeffs (Level 2)
Waveform 9 | Detail Coeffs (Level 1)

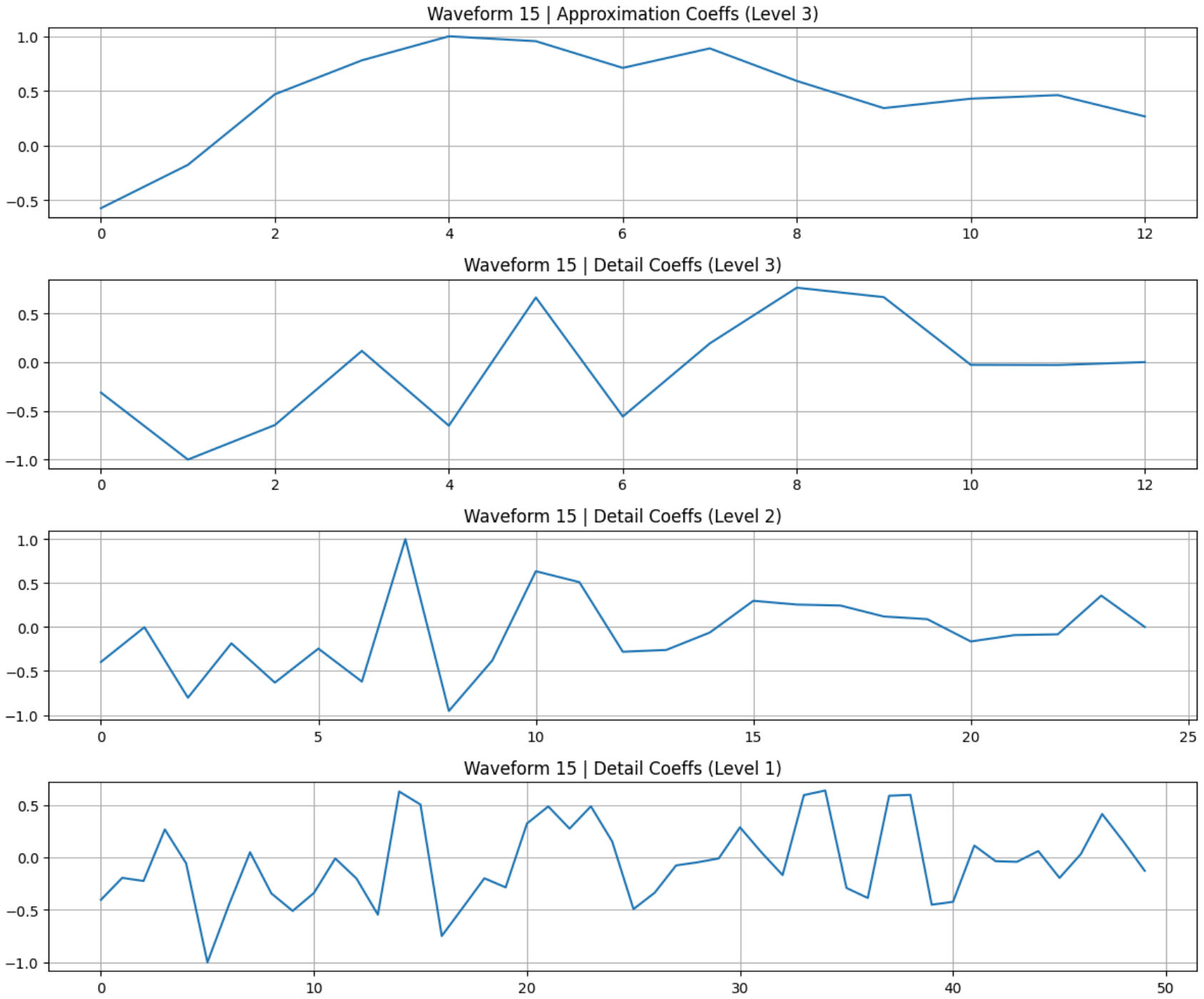

Waveform 10 | Approximation Coeffs (Level 3)
Waveform 10 | Detail Coeffs (Level 3)
Waveform 10 | Detail Coeffs (Level 2)
Waveform 10 | Detail Coeffs (Level 1)

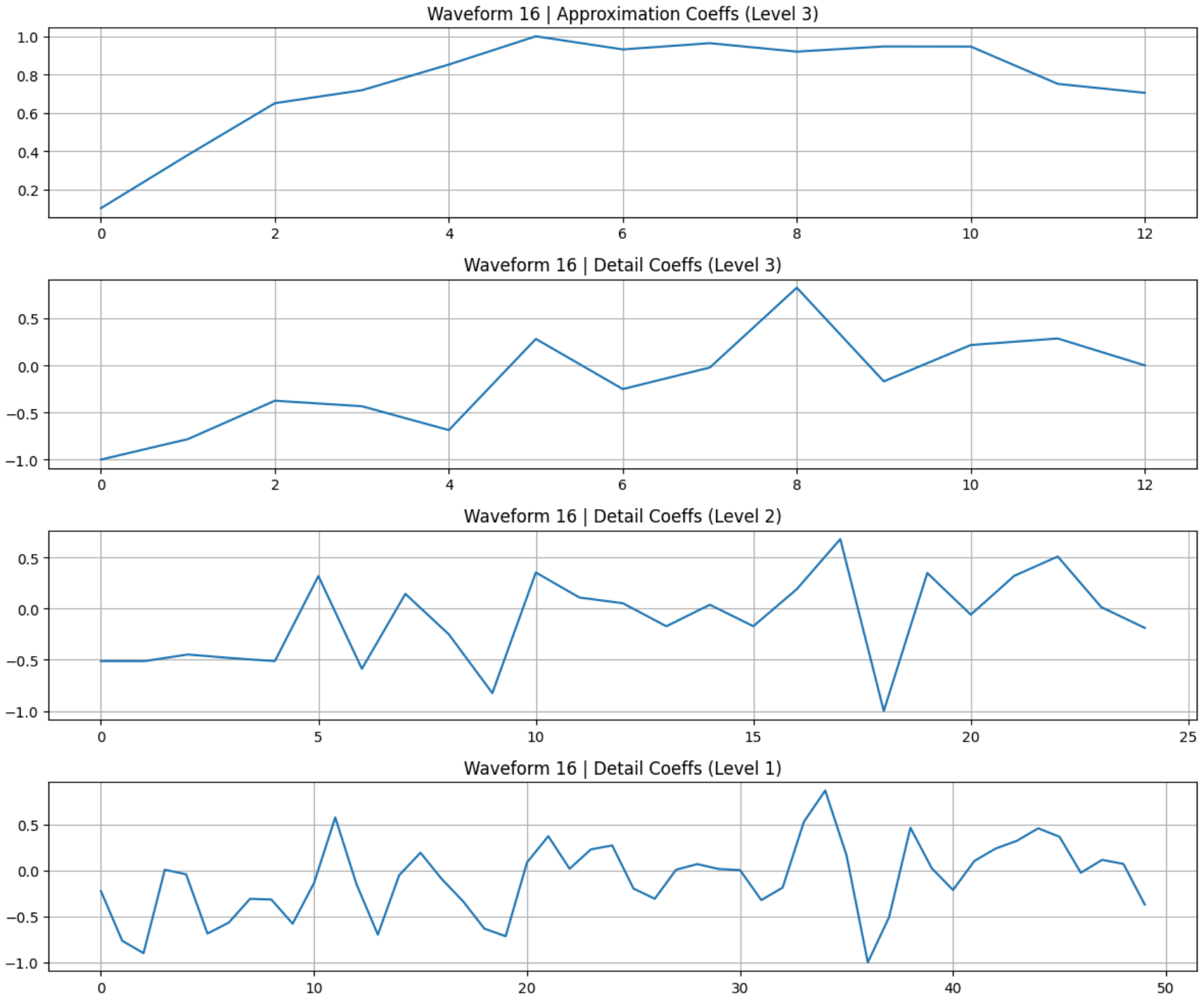

Waveform 11 | Approximation Coeffs (Level 3)
Waveform 11 | Detail Coeffs (Level 3)
Waveform 11 | Detail Coeffs (Level 2)
Waveform 11 | Detail Coeffs (Level 1)

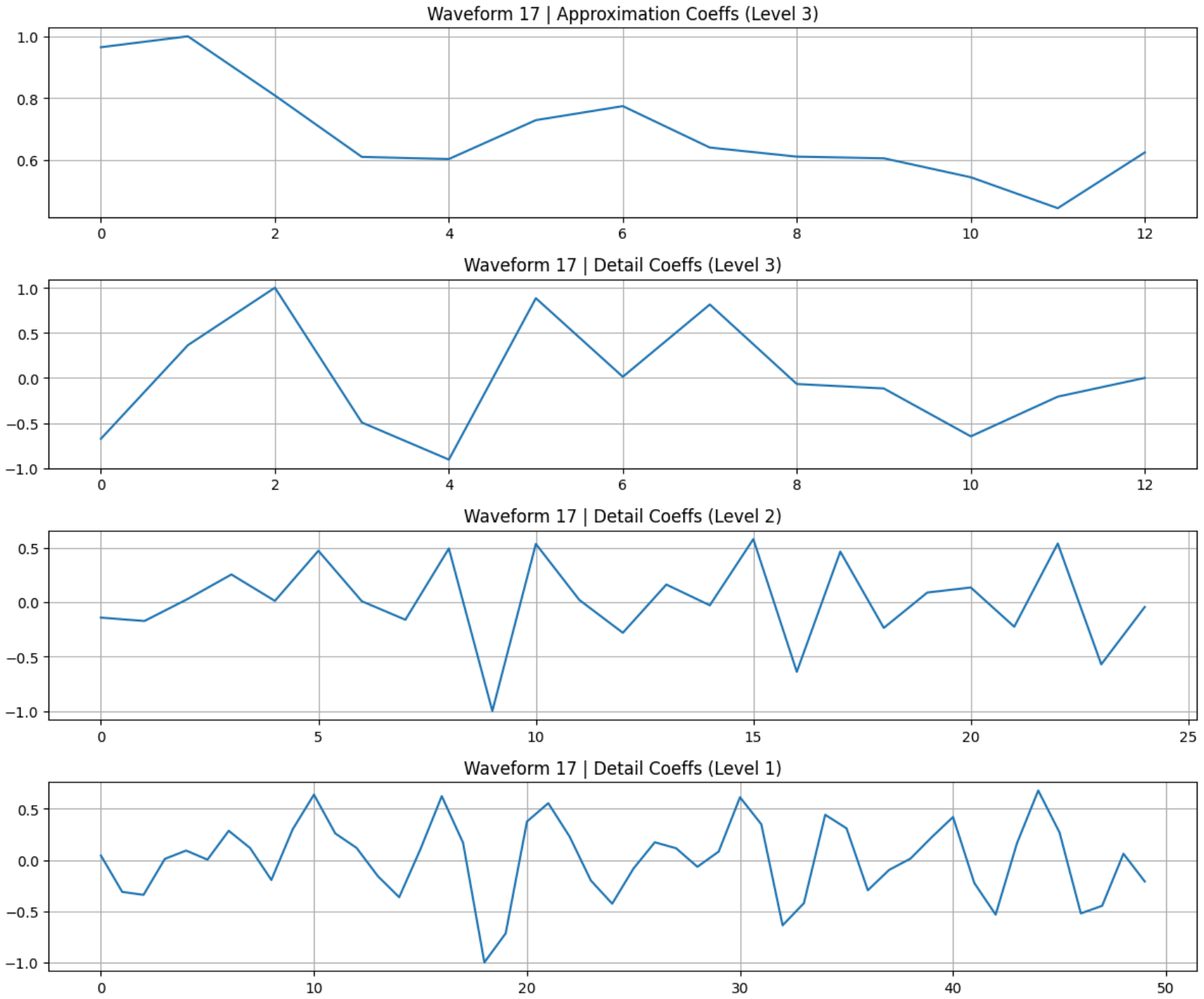

Waveform 12 | Approximation Coeffs (Level 3)
Waveform 12 | Detail Coeffs (Level 3)
Waveform 12 | Detail Coeffs (Level 2)
Waveform 12 | Detail Coeffs (Level 1)

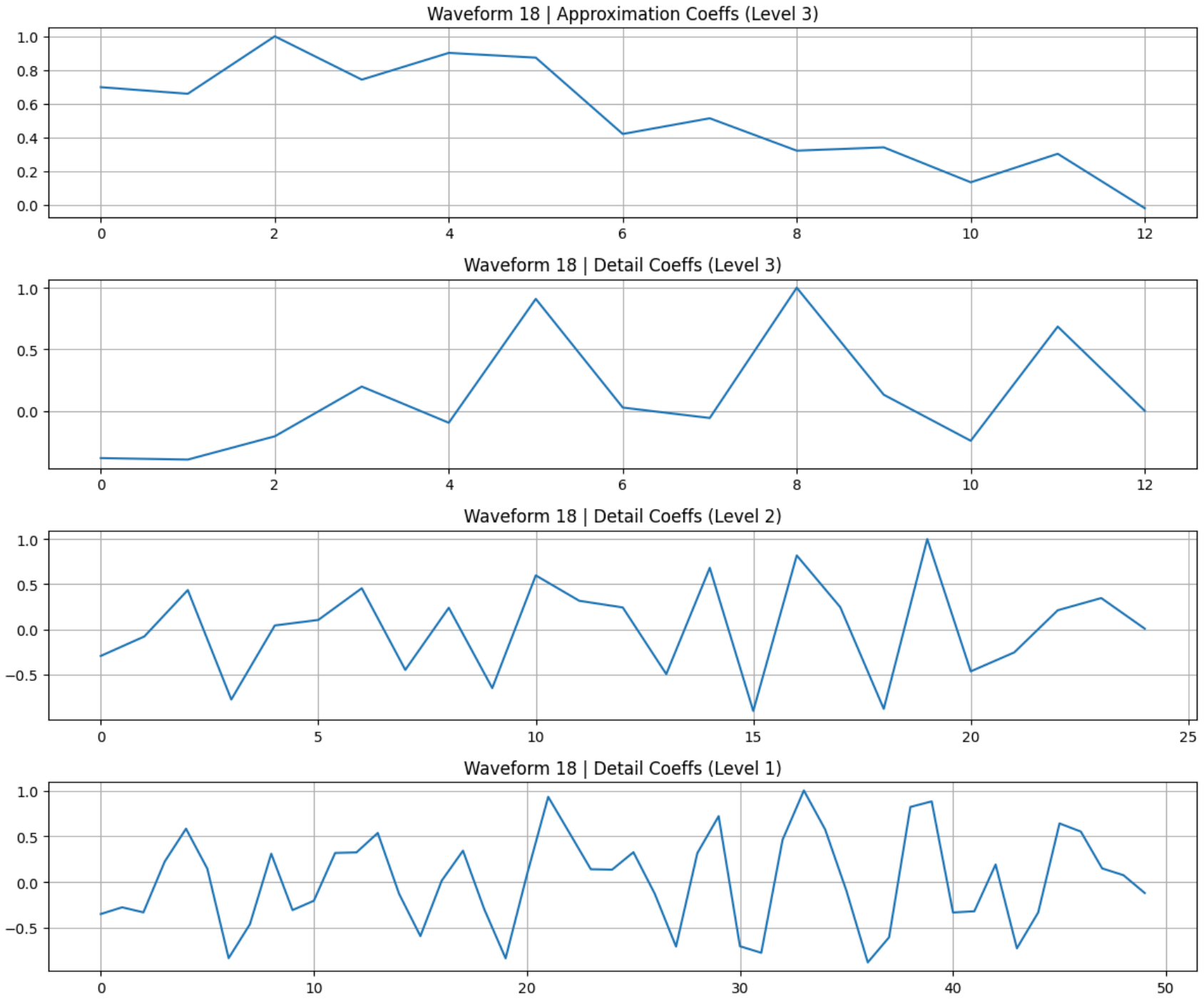

Waveform 13 | Approximation Coeffs (Level 3)
Waveform 13 | Detail Coeffs (Level 3)
Waveform 13 | Detail Coeffs (Level 2)
Waveform 13 | Detail Coeffs (Level 1)

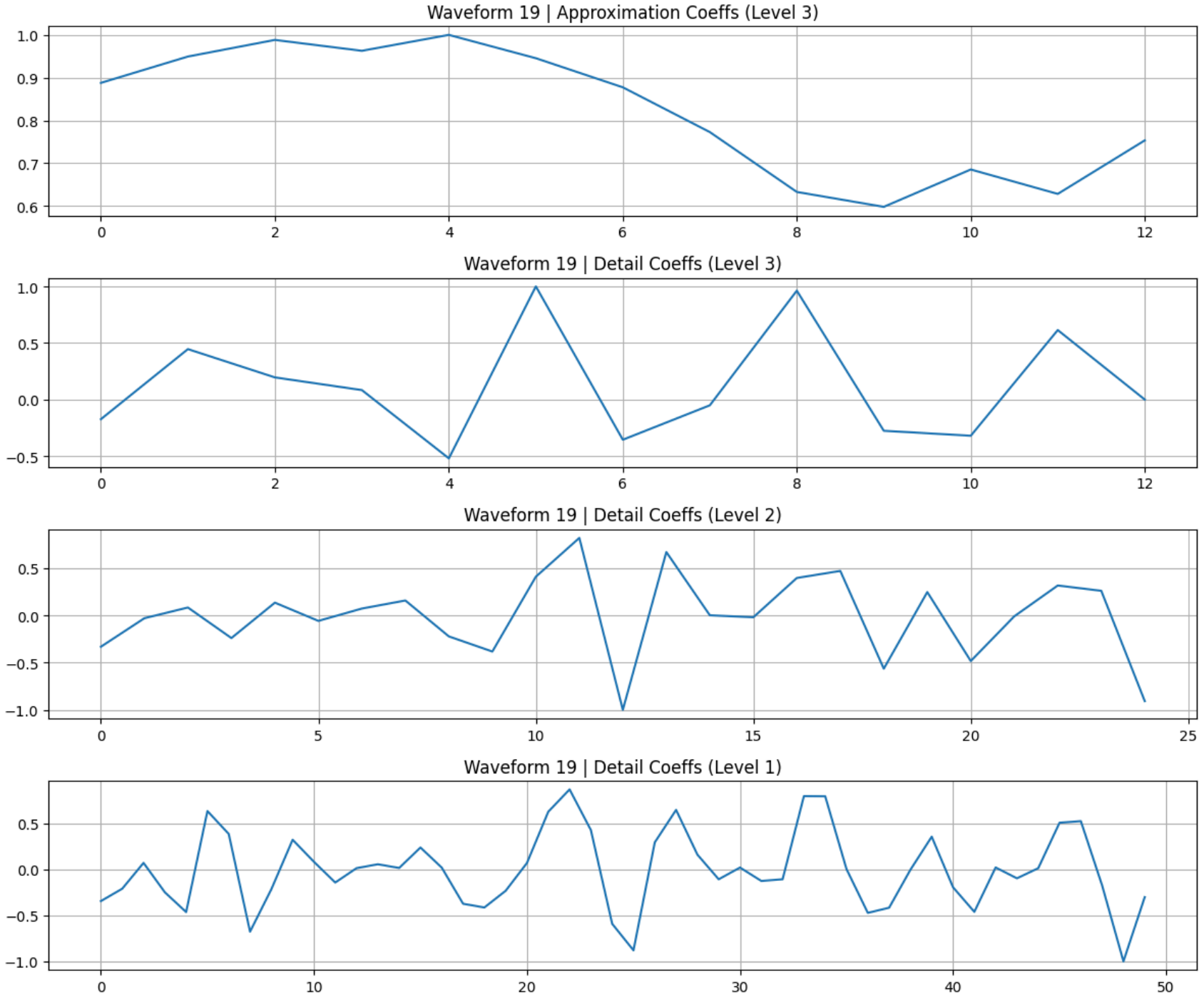

Waveform 14 | Approximation Coeffs (Level 3)
Waveform 14 | Detail Coeffs (Level 3)
Waveform 14 | Detail Coeffs (Level 2)
Waveform 14 | Detail Coeffs (Level 1)

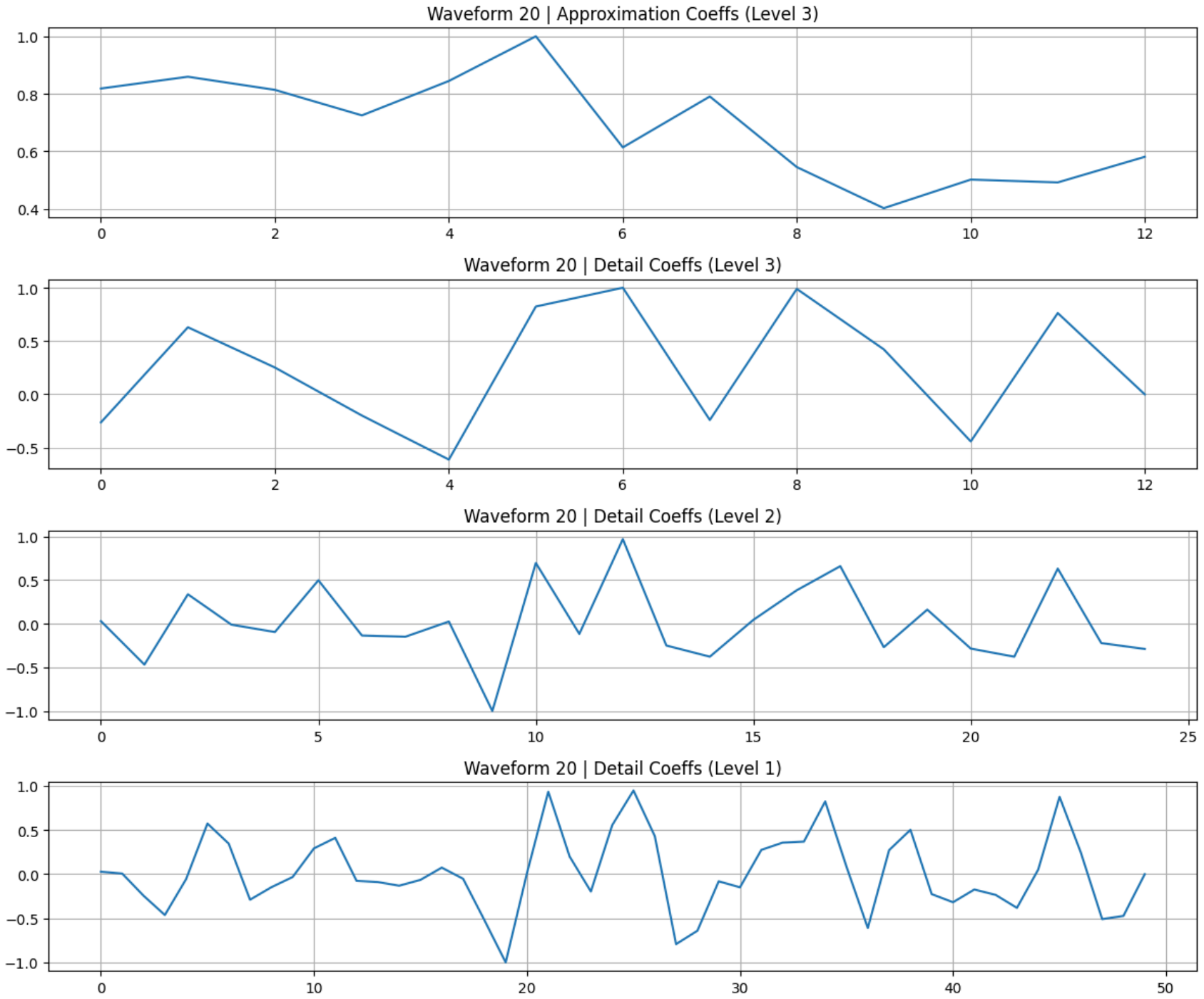

Waveform 15 | Approximation Coeffs (Level 3)
Waveform 15 | Detail Coeffs (Level 3)
Waveform 15 | Detail Coeffs (Level 2)
Waveform 15 | Detail Coeffs (Level 1)

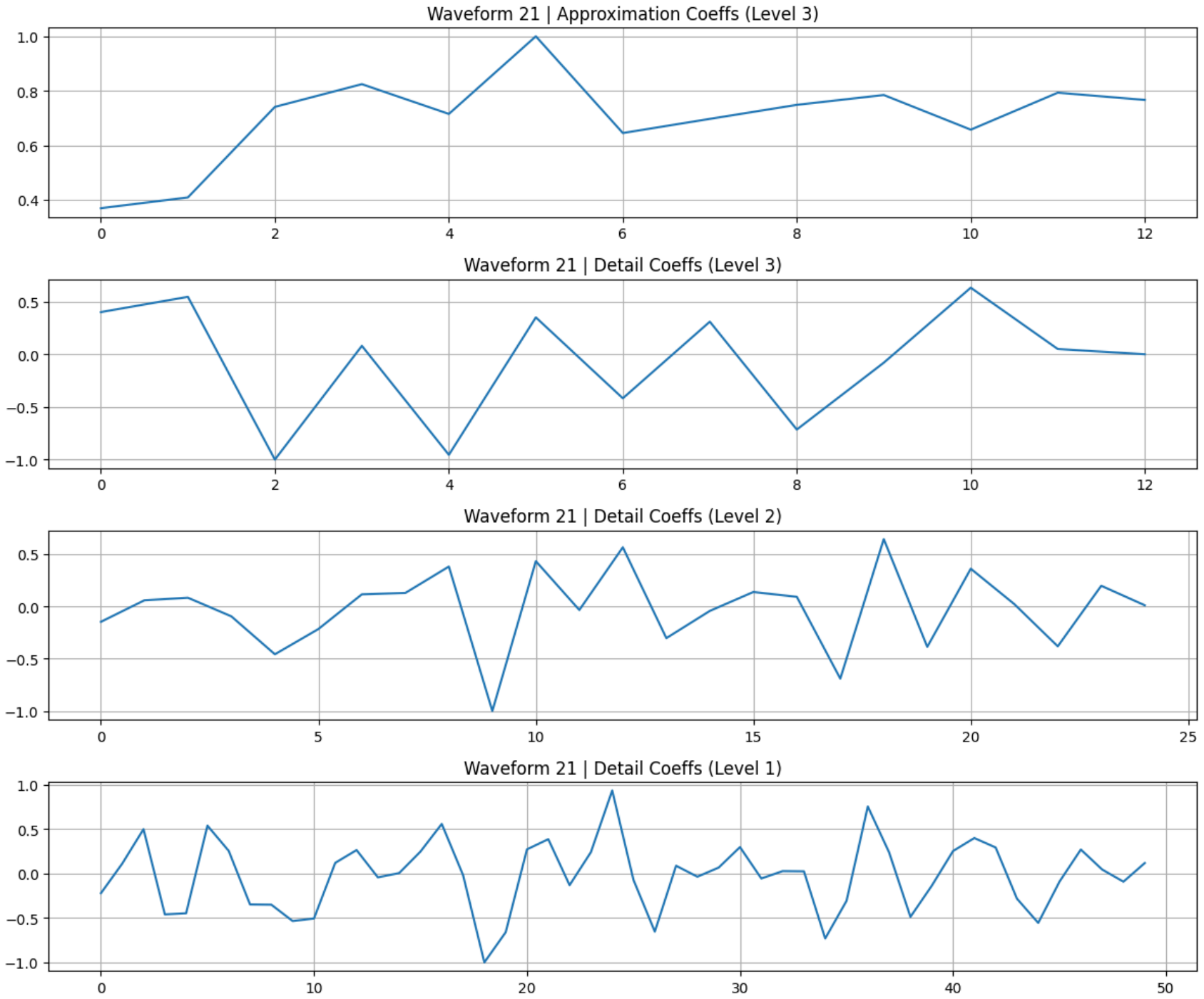

Waveform 16 | Approximation Coeffs (Level 3)
Waveform 16 | Detail Coeffs (Level 3)
Waveform 16 | Detail Coeffs (Level 2)
Waveform 16 | Detail Coeffs (Level 1)

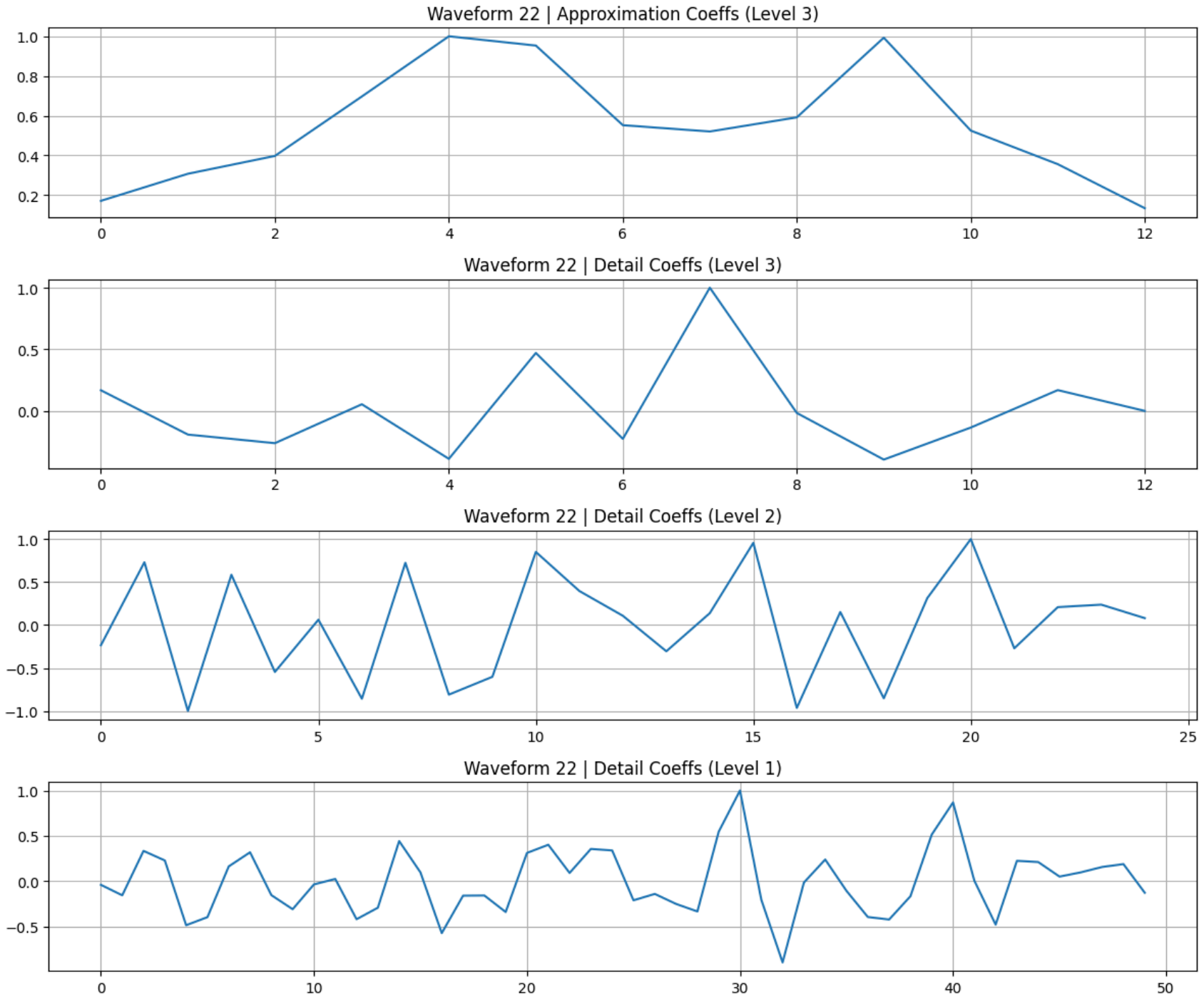

Waveform 17 | Approximation Coeffs (Level 3)
Waveform 17 | Detail Coeffs (Level 3)
Waveform 17 | Detail Coeffs (Level 2)
Waveform 17 | Detail Coeffs (Level 1)


Waveform 18 | Approximation Coeffs (Level 3)
Waveform 18 | Detail Coeffs (Level 3)
Waveform 18 | Detail Coeffs (Level 2)
Waveform 18 | Detail Coeffs (Level 1)


Waveform 19 | Approximation Coeffs (Level 3)
Waveform 19 | Detail Coeffs (Level 3)
Waveform 19 | Detail Coeffs (Level 2)
Waveform 19 | Detail Coeffs (Level 1)

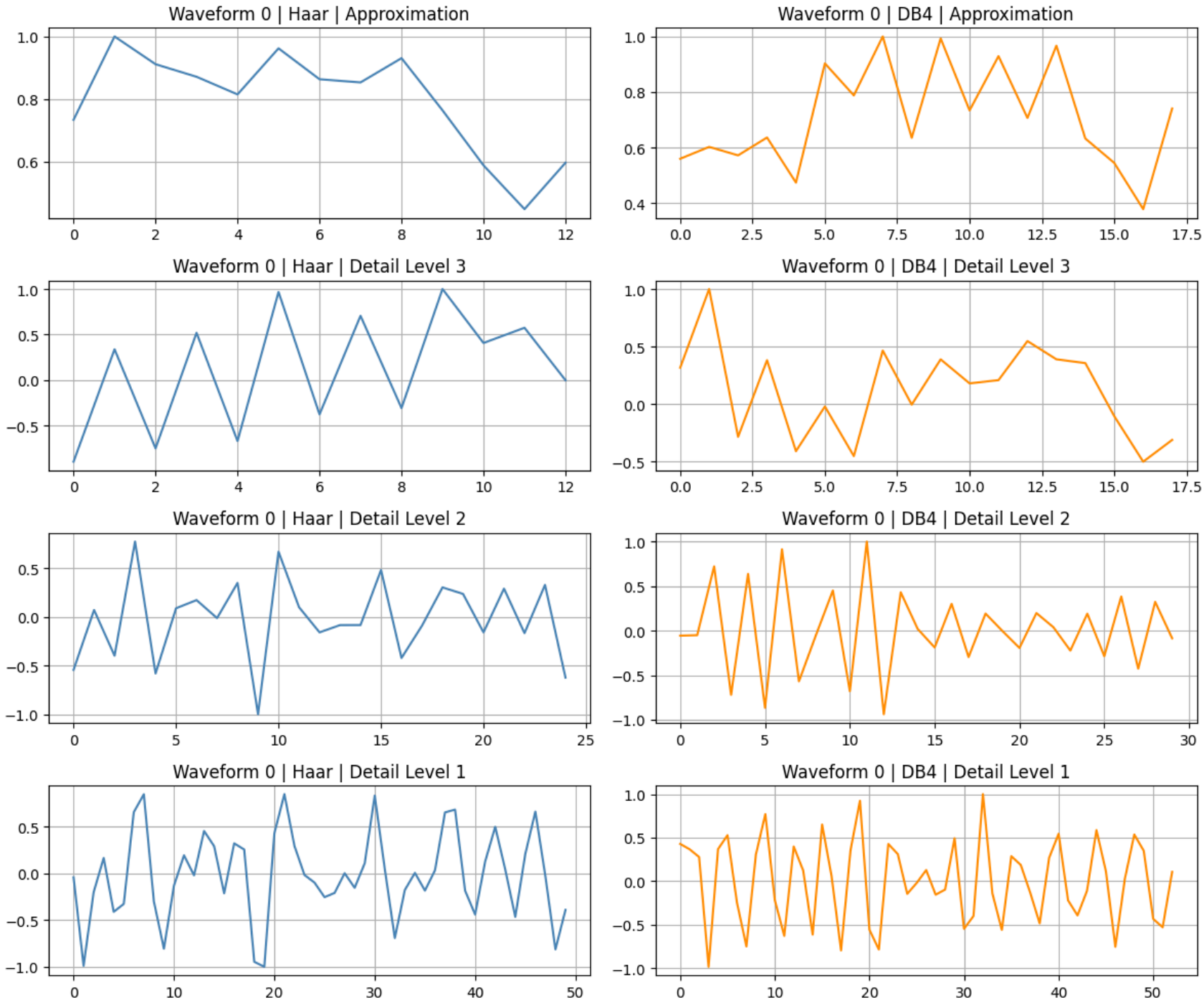

Waveform 20 | Approximation Coeffs (Level 3)
Waveform 20 | Detail Coeffs (Level 3)
Waveform 20 | Detail Coeffs (Level 2)
Waveform 20 | Detail Coeffs (Level 1)

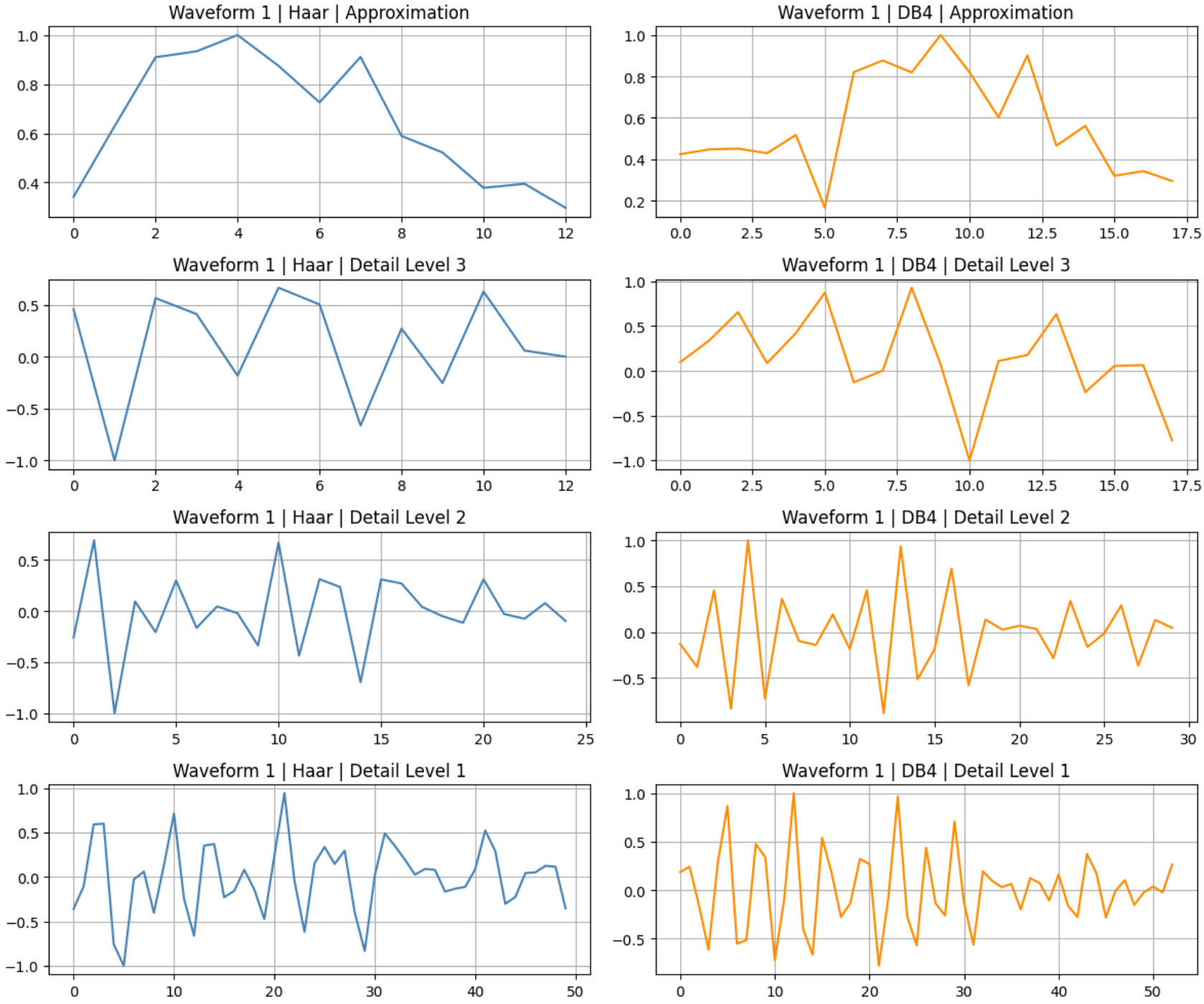

Waveform 21 | Approximation Coeffs (Level 3)
Waveform 21 | Detail Coeffs (Level 3)
Waveform 21 | Detail Coeffs (Level 2)
Waveform 21 | Detail Coeffs (Level 1)

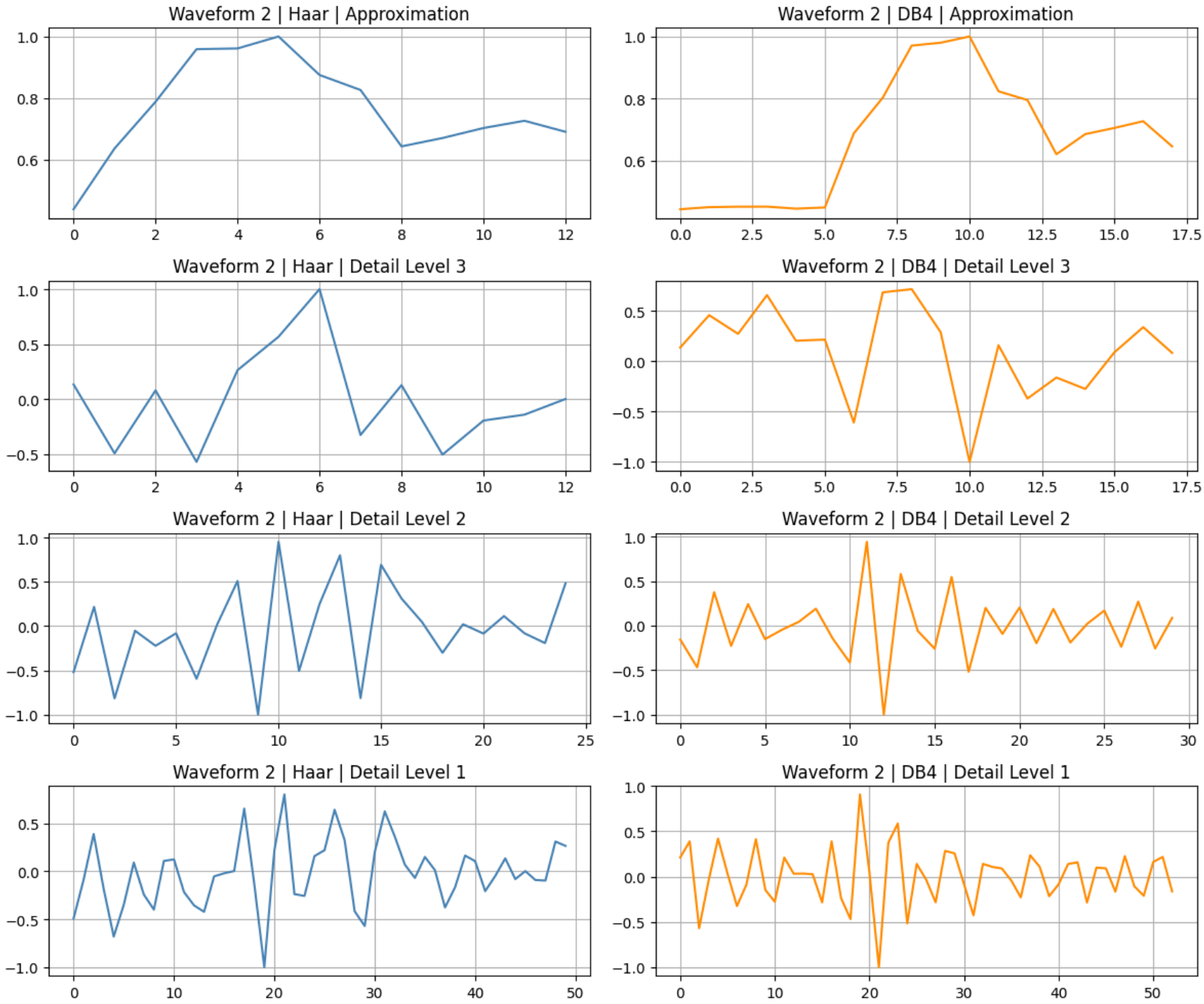

Waveform 22 | Approximation Coeffs (Level 3)
Waveform 22 | Detail Coeffs (Level 3)
Waveform 22 | Detail Coeffs (Level 2)
Waveform 22 | Detail Coeffs (Level 1)

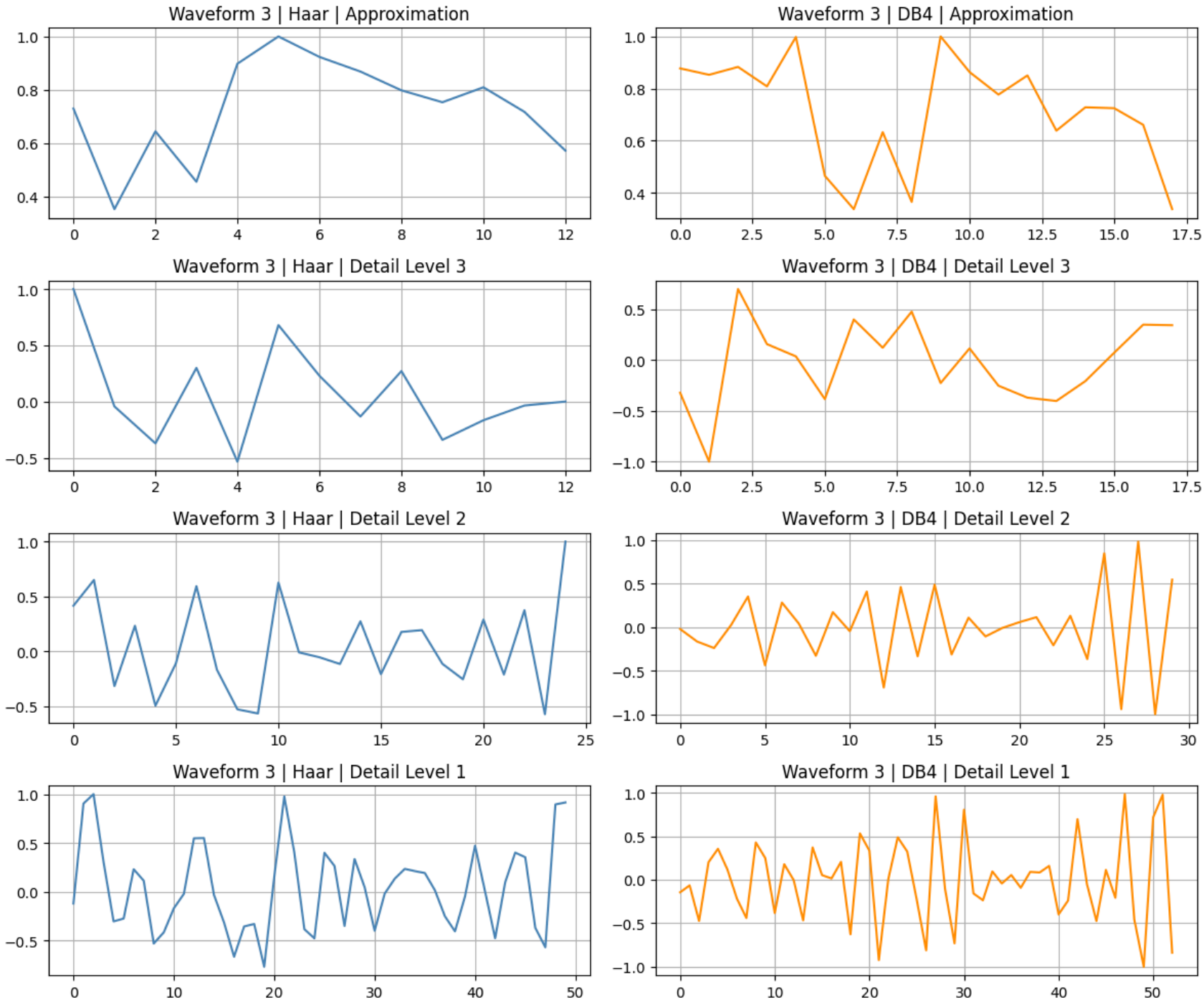
In [ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pywt

def compare_wavelet_coeffs_side_by_side(waveform, level=3, normalize_coeffs=True, title_prefix="Waveform"):
    """
    Visualize Haar vs DB4 wavelet decomposition coefficients side-by-side for each level.
    """

    # Perform decomposition with Haar and db4  → returns [A3, D3, D2, D1]
    coeffs_haar = pywt.wavedec(waveform, wavelet='haar', level=level)
    coeffs_db4 = pywt.wavedec(waveform, wavelet='db4', level=level)

    n_levels = len(coeffs_haar)  # Should be level + 1 (includes Approximation)
    fig, axes = plt.subplots(n_levels, 2, figsize=(12, 2.5 * n_levels), sharex=False)

    for i in range(n_levels):
        c_haar = coeffs_haar[i]
        c_db4 = coeffs_db4[i]

        if normalize_coeffs:
            c_haar = c_haar / np.max(np.abs(c_haar)) if np.max(np.abs(c_haar)) != 0 else c_haar
            c_db4 = c_db4 / np.max(np.abs(c_db4)) if np.max(np.abs(c_db4)) != 0 else c_db4

        level_name = "Approximation" if i == 0 else f"Detail Level {level - i + 1}"

        # Haar on the left
        axes[i, 0].plot(c_haar, color='steelblue')
        axes[i, 0].set_title(f"{title_prefix} | Haar | {level_name}")
        axes[i, 0].grid(True)

        # DB4 on the right
        axes[i, 1].plot(c_db4, color='darkorange')
        axes[i, 1].set_title(f"{title_prefix} | DB4 | {level_name}")
        axes[i, 1].grid(True)

    plt.tight_layout()
    plt.show()

for idx in range(10):  # Compare first 5 waveforms
    compare_wavelet_coeffs_side_by_side(
```

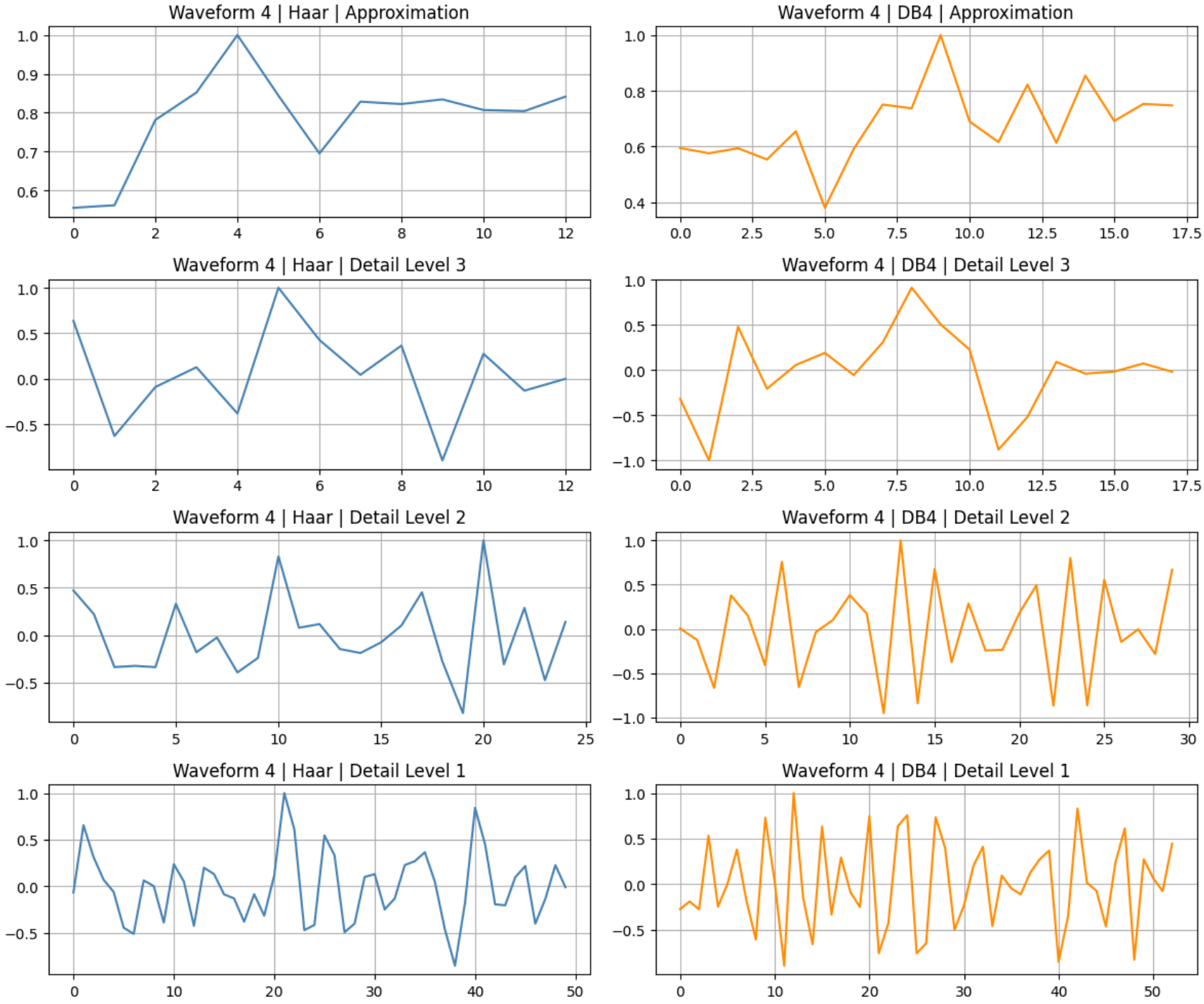
```
        waveform=normalized_waveforms[idx],
        level=3,
        normalize_coeffs=True,
        title_prefix=f"Waveform {idx}"
    )
```

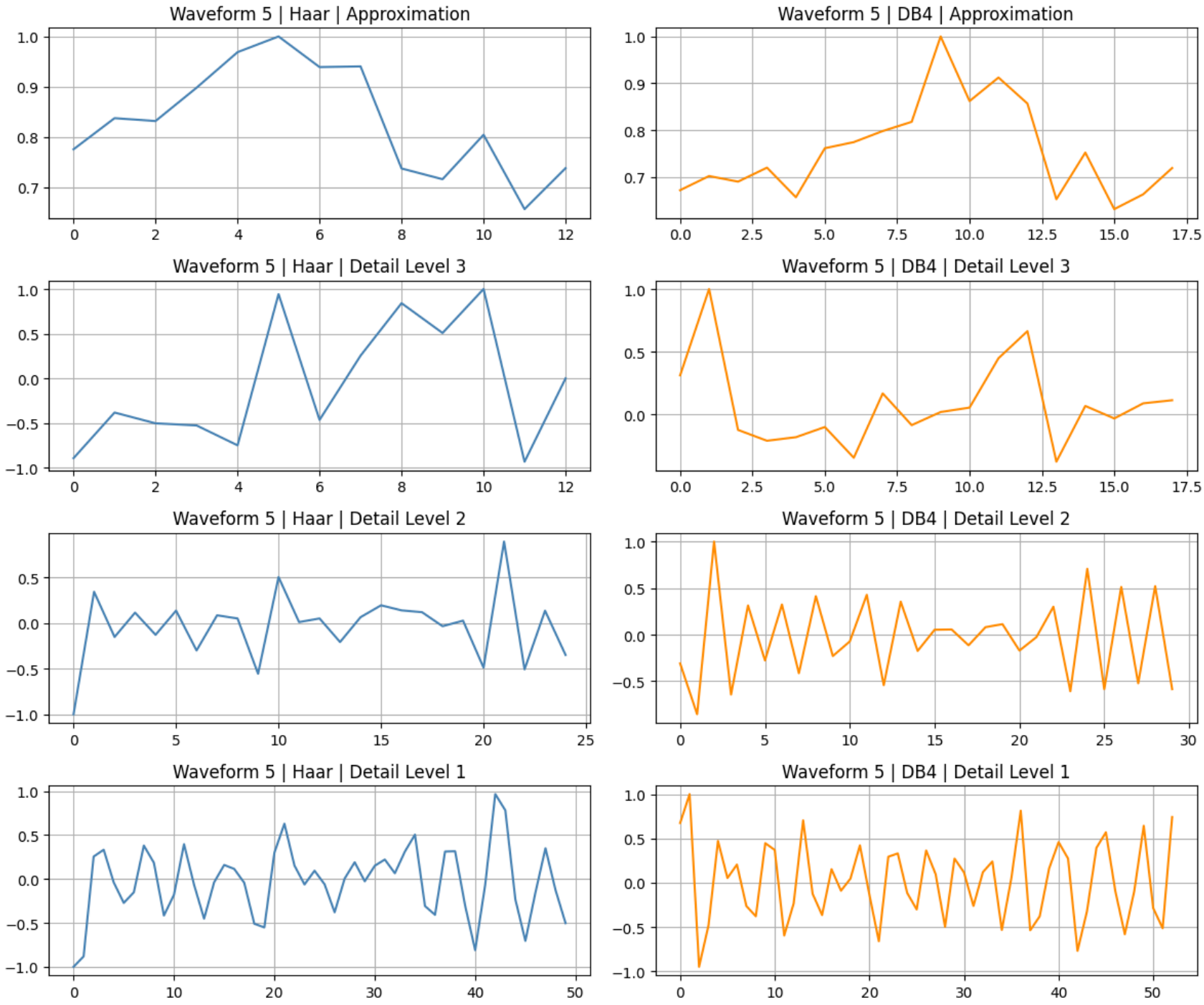

Waveform 0 | Haar | Approximation
Waveform 0 | DB4 | Approximation
Waveform 0 | Haar | Detail Level 3
Waveform 0 | DB4 | Detail Level 3
Waveform 0 | Haar | Detail Level 2
Waveform 0 | DB4 | Detail Level 2
Waveform 0 | Haar | Detail Level 1
Waveform 0 | DB4 | Detail Level 1

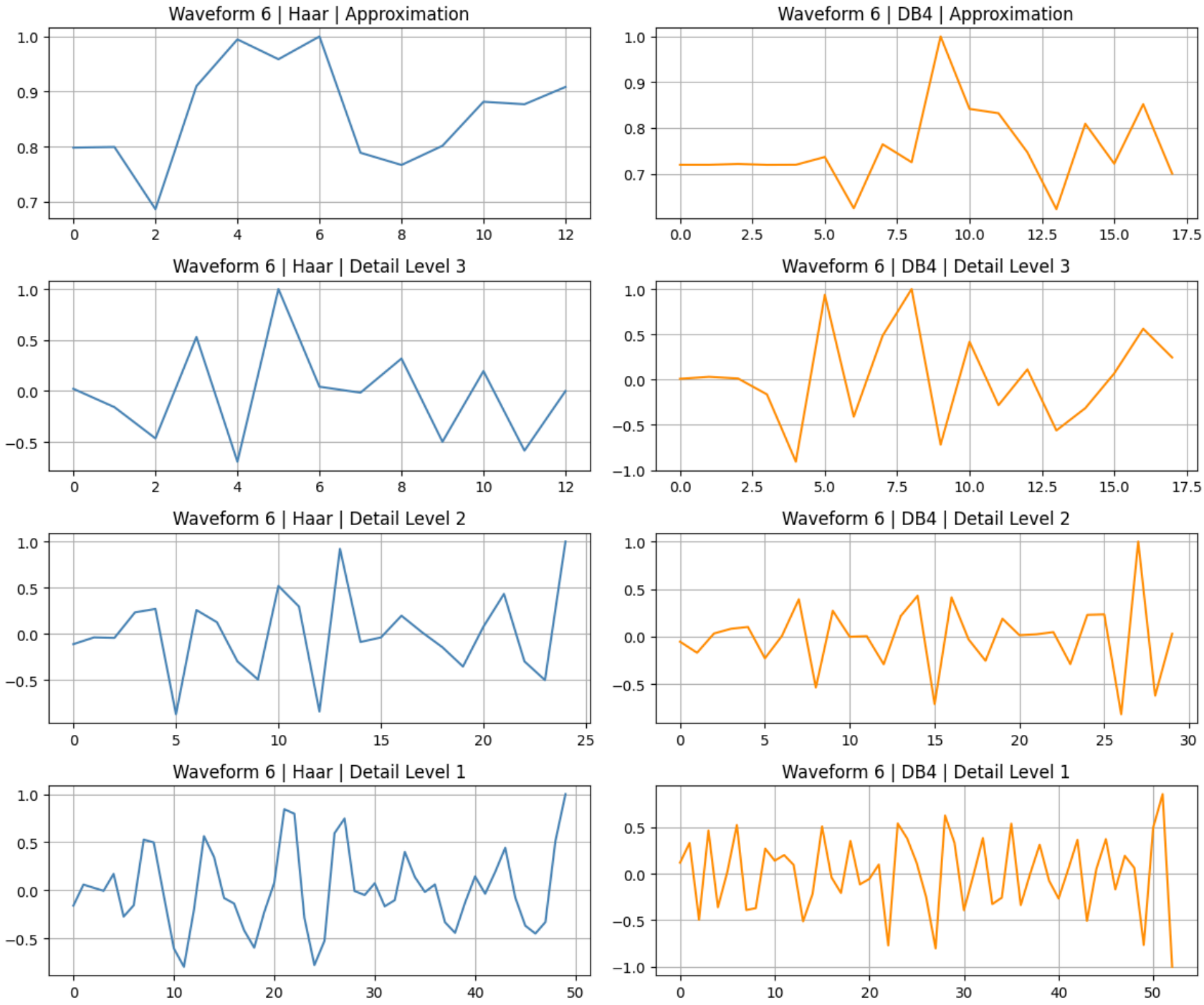

Waveform 1 | Haar | Approximation
Waveform 1 | DB4 | Approximation
Waveform 1 | Haar | Detail Level 3
Waveform 1 | DB4 | Detail Level 3
Waveform 1 | Haar | Detail Level 2
Waveform 1 | DB4 | Detail Level 2
Waveform 1 | Haar | Detail Level 1
Waveform 1 | DB4 | Detail Level 1

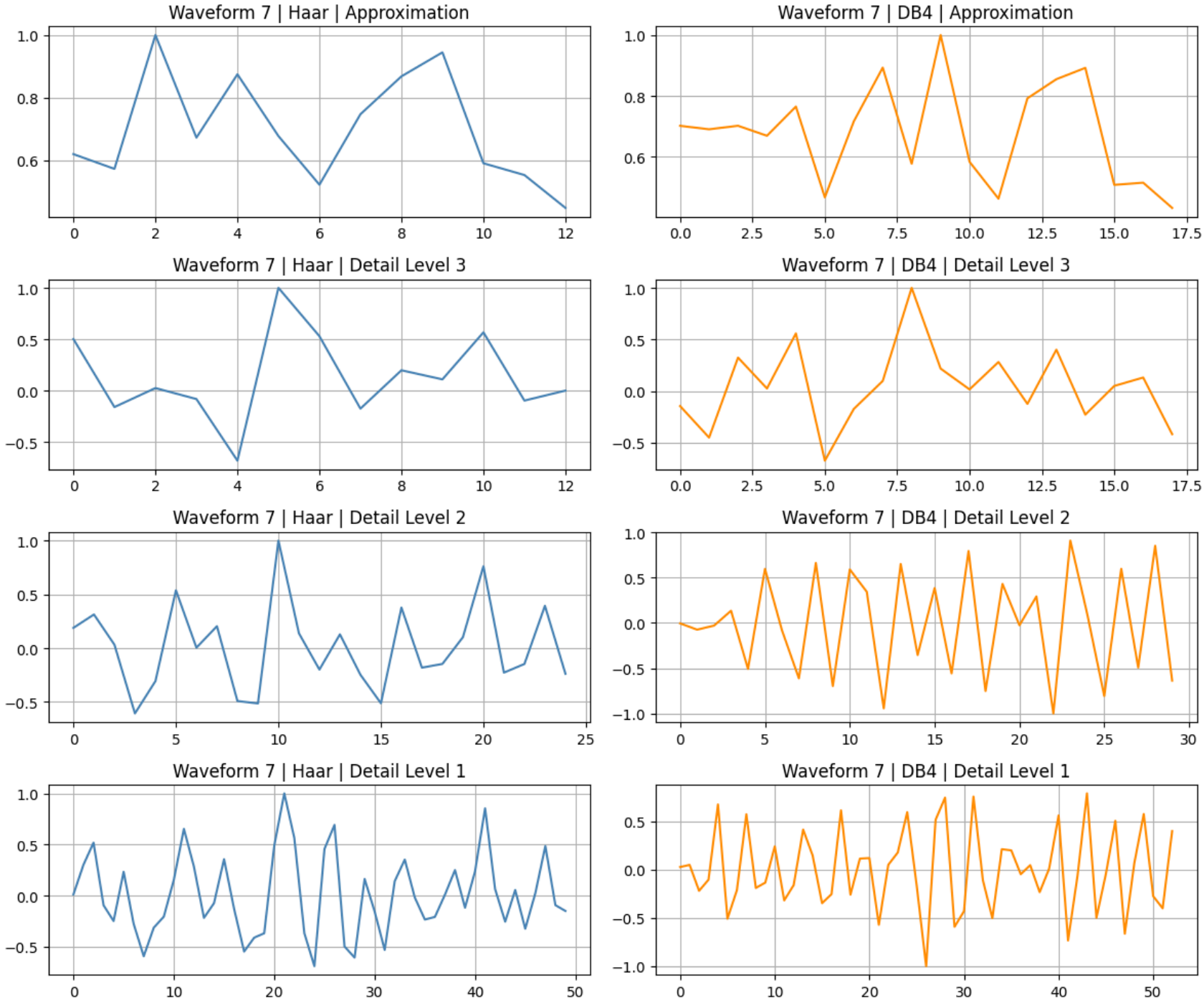

Waveform 2 | Haar | Approximation
Waveform 2 | DB4 | Approximation
Waveform 2 | Haar | Detail Level 3
Waveform 2 | DB4 | Detail Level 3
Waveform 2 | Haar | Detail Level 2
Waveform 2 | DB4 | Detail Level 2
Waveform 2 | Haar | Detail Level 1
Waveform 2 | DB4 | Detail Level 1

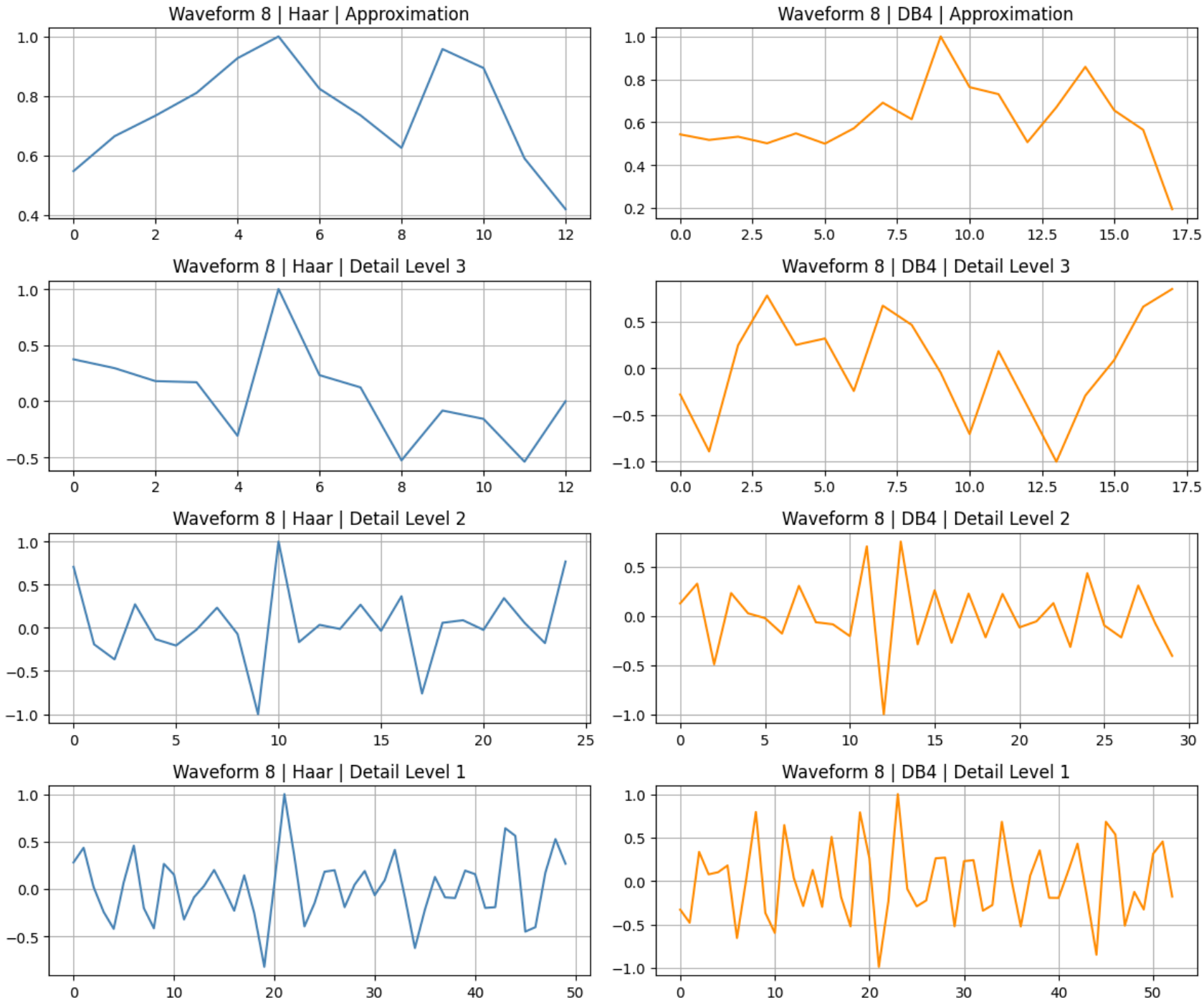

Waveform 3 | Haar | Approximation
Waveform 3 | DB4 | Approximation
Waveform 3 | Haar | Detail Level 3
Waveform 3 | DB4 | Detail Level 3
Waveform 3 | Haar | Detail Level 2
Waveform 3 | DB4 | Detail Level 2
Waveform 3 | Haar | Detail Level 1
Waveform 3 | DB4 | Detail Level 1

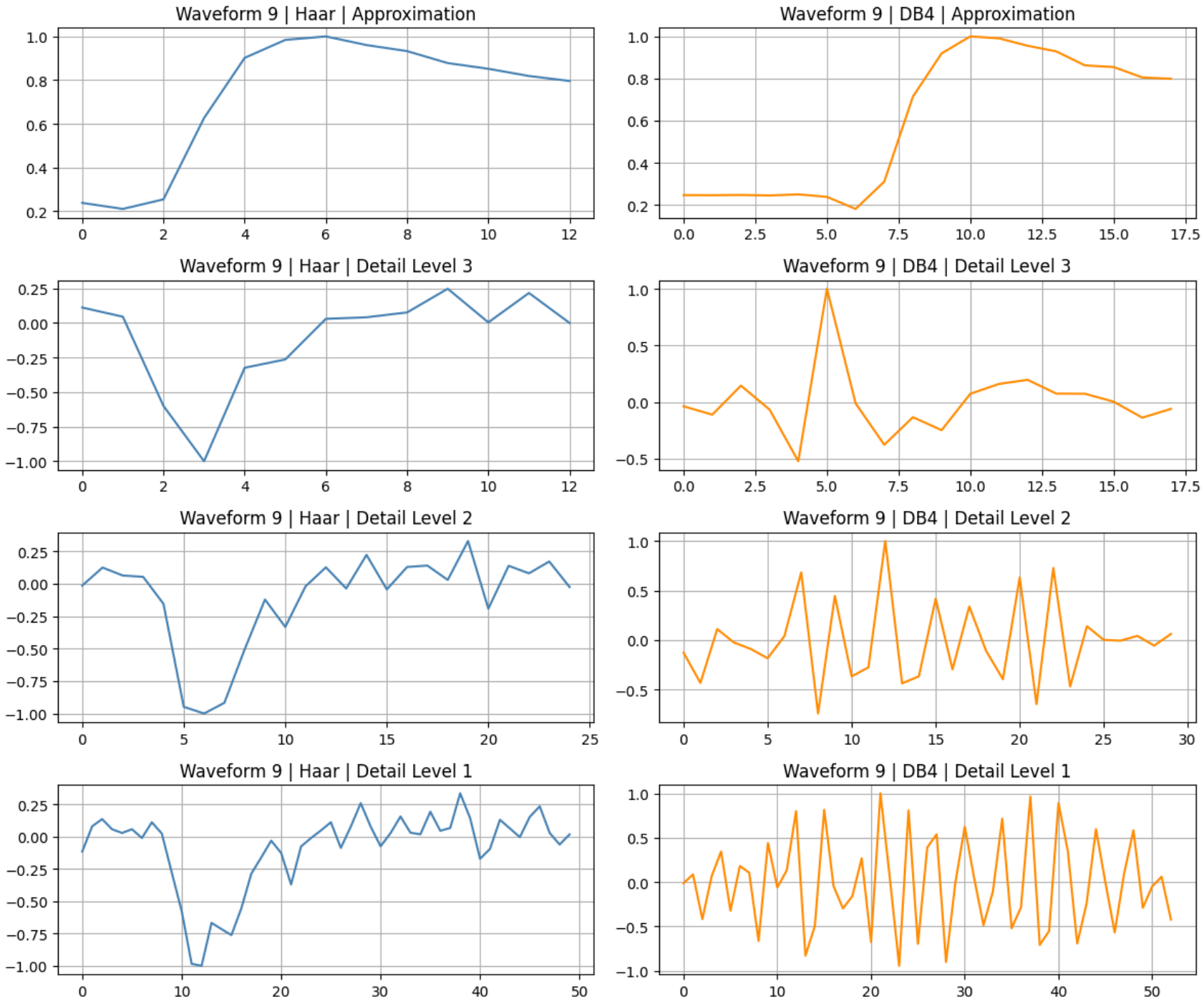

Waveform 4 | Haar | Approximation
Waveform 4 | DB4 | Approximation
Waveform 4 | Haar | Detail Level 3
Waveform 4 | DB4 | Detail Level 3
Waveform 4 | Haar | Detail Level 2
Waveform 4 | DB4 | Detail Level 2
Waveform 4 | Haar | Detail Level 1
Waveform 4 | DB4 | Detail Level 1


Waveform 5 | Haar | Approximation
Waveform 5 | DB4 | Approximation
Waveform 5 | Haar | Detail Level 3
Waveform 5 | DB4 | Detail Level 3
Waveform 5 | Haar | Detail Level 2
Waveform 5 | DB4 | Detail Level 2
Waveform 5 | Haar | Detail Level 1
Waveform 5 | DB4 | Detail Level 1

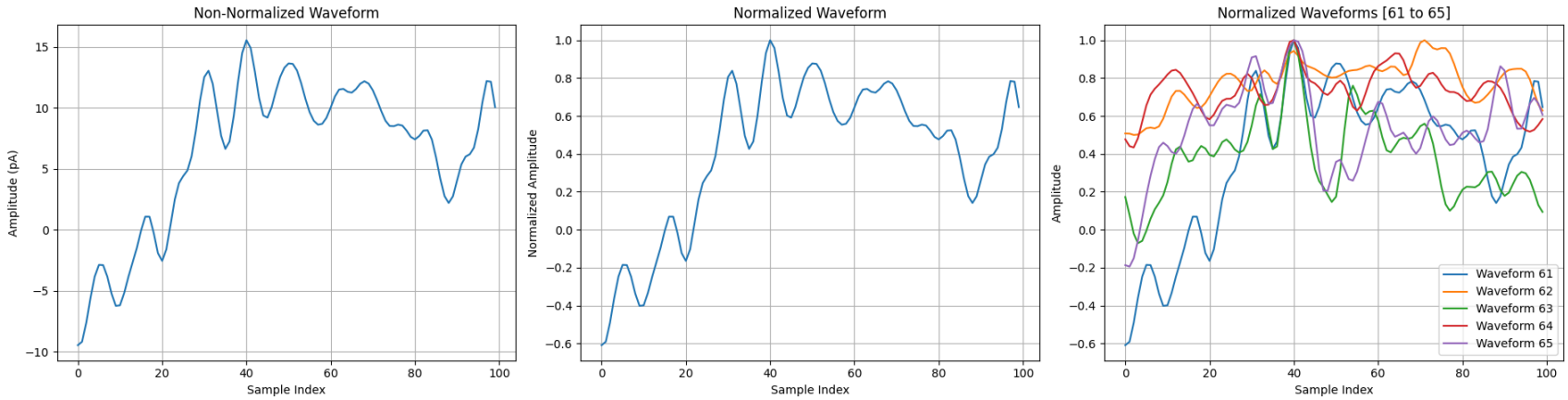

Waveform 6 | Haar | Approximation
Waveform 6 | DB4 | Approximation
Waveform 6 | Haar | Detail Level 3
Waveform 6 | DB4 | Detail Level 3
Waveform 6 | Haar | Detail Level 2
Waveform 6 | DB4 | Detail Level 2
Waveform 6 | Haar | Detail Level 1
Waveform 6 | DB4 | Detail Level 1


Waveform 7 | Haar | Approximation
Waveform 7 | DB4 | Approximation
Waveform 7 | Haar | Detail Level 3
Waveform 7 | DB4 | Detail Level 3
Waveform 7 | Haar | Detail Level 2
Waveform 7 | DB4 | Detail Level 2
Waveform 7 | Haar | Detail Level 1
Waveform 7 | DB4 | Detail Level 1


Waveform 8 | Haar | Approximation
Waveform 8 | DB4 | Approximation
Waveform 8 | Haar | Detail Level 3
Waveform 8 | DB4 | Detail Level 3
Waveform 8 | Haar | Detail Level 2
Waveform 8 | DB4 | Detail Level 2
Waveform 8 | Haar | Detail Level 1
Waveform 8 | DB4 | Detail Level 1


Waveform 9 | Haar | Approximation
Waveform 9 | DB4 | Approximation
Waveform 9 | Haar | Detail Level 3
Waveform 9 | DB4 | Detail Level 3
Waveform 9 | Haar | Detail Level 2
Waveform 9 | DB4 | Detail Level 2
Waveform 9 | Haar | Detail Level 1
Waveform 9 | DB4 | Detail Level 1

In [ ]:
```
# -- Run waveform inspection for sanity check
plot_waveform_inspection(waveforms, normalized_waveforms, index_wave=61, n_examples=5)
#waveforms
```



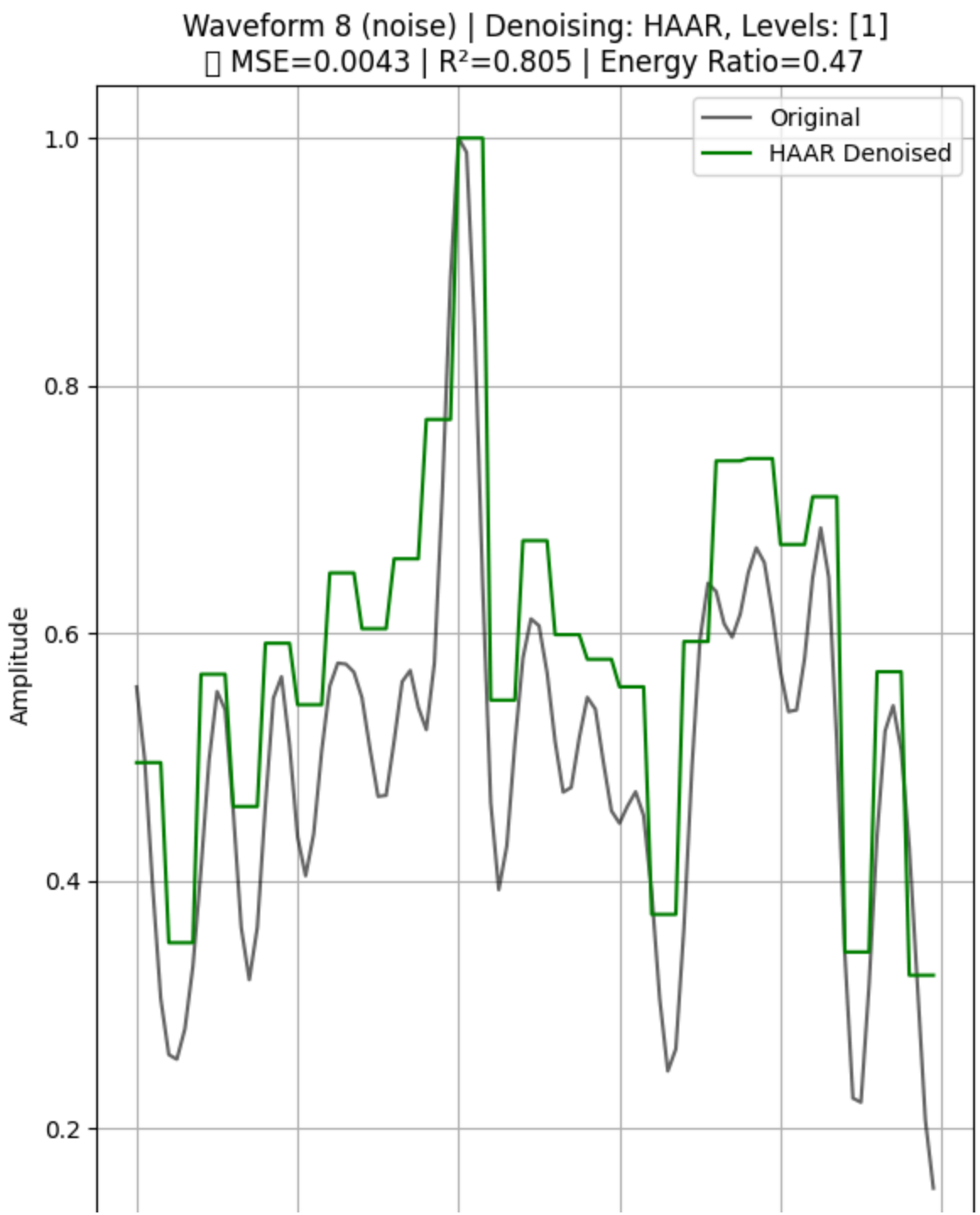
In [ ]:
```
from sklearn.metrics import mean_squared_error, r2_score

def denoise_and_reconstruct_signal(signal, wavelet='db4', level=3, threshold_ratio=0.2, keep_levels=[1, 2], original=
    """
    Denoises a signal using wavelet decomposition. Tracks which levels contribute most.

    Returns:
    - reconstructed_signal
    - dict of metrics: {MSE, R2, Energy Ratio, Kept Levels}
    """
    # Perform multi-level wavelet decomposition
    coeffs = pywt.wavedec(signal, wavelet=wavelet, level=level)
    total_energy = sum(np.sum(c**2) for c in coeffs[1:])  # Compute total detail energy

    # Threshold and keep selected detail levels only
    # i = 1 is Level N (e.g., D3), i = 2 is Level N-1 (e.g., D2), ..., i = N is Level 1 (high-freq)
    kept_energy = 0
    for i in range(1, len(coeffs)):  # Skip Approximation
        if i in keep_levels:
            sigma = np.std(coeffs[i])
            threshold = threshold_ratio * sigma # Apply soft thresholding if level is selected
            coeffs[i] = pywt.threshold(coeffs[i], threshold, mode='soft')
            kept_energy += np.sum(coeffs[i]**2)
```

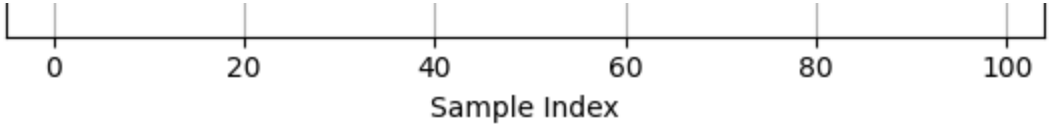
```
        else:
            coeffs[i] = np.zeros_like(coeffs[i])  # zero out and discard this level

    # Reconstruct the signal from the modified coefficients
    reconstructed = pywt.waverec(coeffs, wavelet)

    # Match lengths (Crop all signals to the same length)
    min_len = min(len(signal), len(reconstructed))
    signal = signal[:min_len]
    reconstructed = reconstructed[:min_len]

    if original is not None:
        original = original[:min_len]
    else:
        original = signal

    # Calculate error metrics between original and reconstructed signal
    # MSE: How different is reconstructed from original?
    # R²: How much variance is preserved?
    # Energy Ratio: What percent of original detail energy was retained?
    metrics = {
        "MSE": mean_squared_error(original, reconstructed),
        "R2": r2_score(original, reconstructed),
        "EnergyRatio": kept_energy / total_energy if total_energy != 0 else 0,
        "KeptLevels": keep_levels,
        "Wavelet": wavelet
    }

    return reconstructed, metrics

###
def compare_recon_and_denoise(waveforms, level=3, threshold_ratio=0.2, keep_levels=[1, 2],
                              normalize=True, wavelet='db4', title_prefix="Waveform"):

    recon, metrics = denoise_and_reconstruct_signal(
        signal=waveforms,
        wavelet=wavelet,
        level=level,
        threshold_ratio=threshold_ratio,
        keep_levels=keep_levels,
        original=waveforms
    )
```

```
    if normalize:
        waveforms = waveforms / np.max(np.abs(waveforms))
        recon = recon / np.max(np.abs(recon))

    plt.figure(figsize=(6, 8))
    plt.plot(waveforms, label="Original", color="black", alpha=0.6)
    plt.plot(recon, label=f"{wavelet.upper()} Denoised", color="green", linewidth=1.5)
    plt.title(f"{title_prefix} | Denoising: {wavelet.upper()}, Levels: {keep_levels}\n"
              f"📉 MSE={metrics['MSE']:.4f} | R²={metrics['R2']:.3f} | Energy Ratio={metrics['EnergyRatio']:.2f}")
    plt.xlabel("Sample Index")
    plt.ylabel("Amplitude")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    return metrics

##

results = []
interest_waveforms = {
    "Waveform 8 (noise)": normalized_waveforms[8], # change for which waveform is noise
    "Waveform 9 (event)": normalized_waveforms[9], # change for which waveform is event
    "Waveform 61 (event)": normalized_waveforms[61] # change for which waveform is event
}

# Try wavelets: db4, coif5, sym5, bior3.5
for wavelet in ['haar', 'db4', 'sym5', 'bior3.5']:
    for keep_levels in [[1], [2], [3], [1, 2], [2, 3], [1,2,3]]:
        for name, signal in interest_waveforms.items():
            metrics = compare_recon_and_denoise(
                waveforms=signal,
                wavelet=wavelet,
                level=3,
                threshold_ratio=0.2,
                keep_levels=keep_levels,
                title_prefix=f"{name}"
            )
            metrics["Waveform"] = name
            results.append(metrics)
```

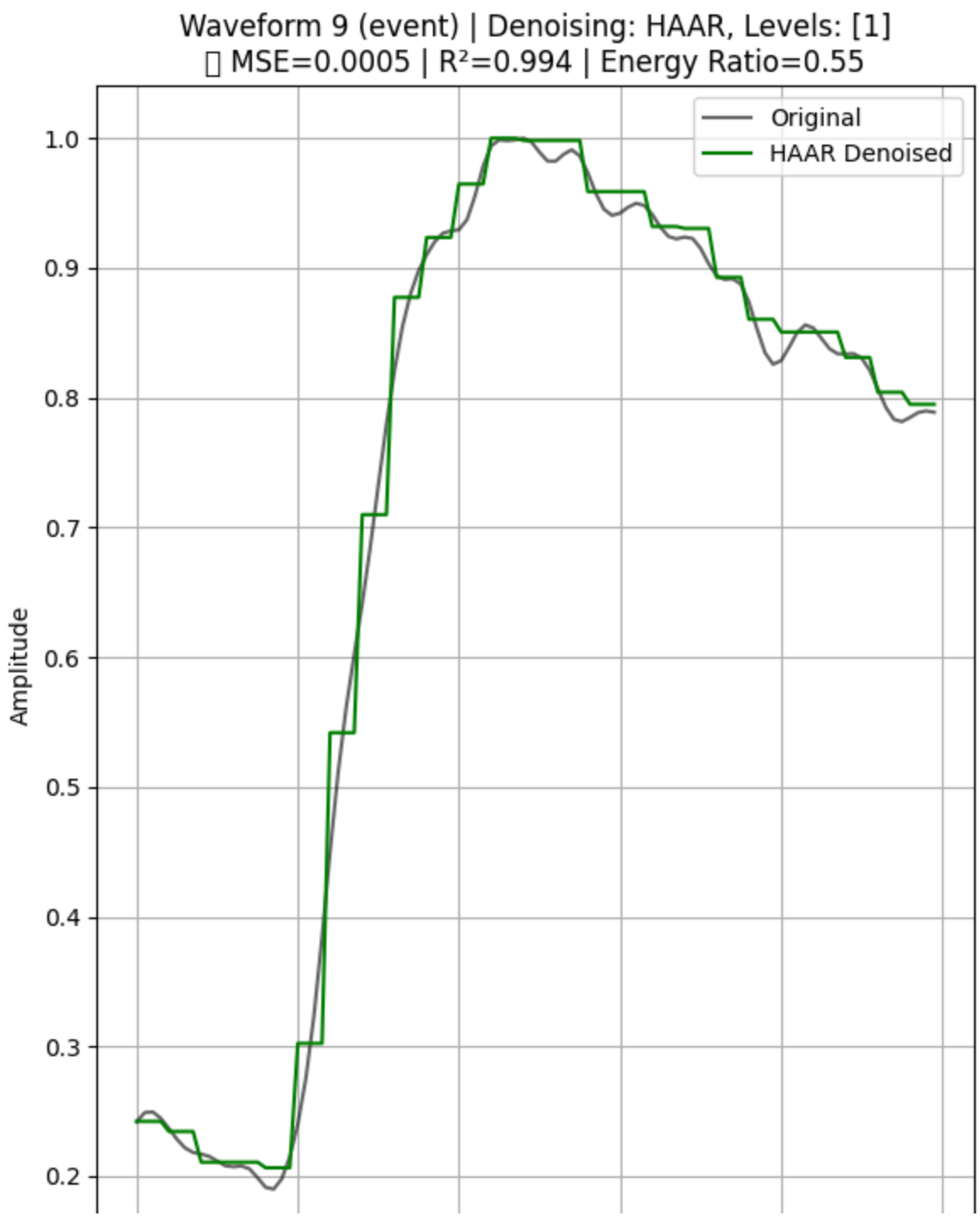
```
# Haar is a step function wavelet, meaning:
# It jumps between values rather than transitioning smoothly.
# Very fast to compute.
# Great for detecting abrupt changes, but bad for smooth slopes (e.g., waveform 9).
# Thus, DB4 and Coif5 give smoother reconstructions.
```

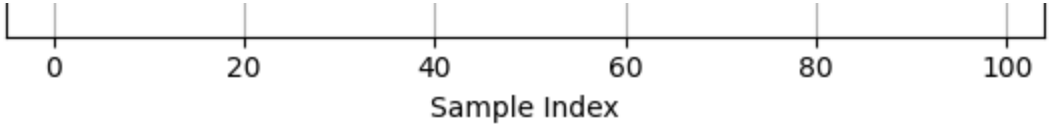
```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s)
DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWN
WARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 8 (noise) | Denoising: HAAR, Levels: [1]
 MSE=0.0043 | R²=0.805 | Energy Ratio=0.47
Original
HAAR Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2

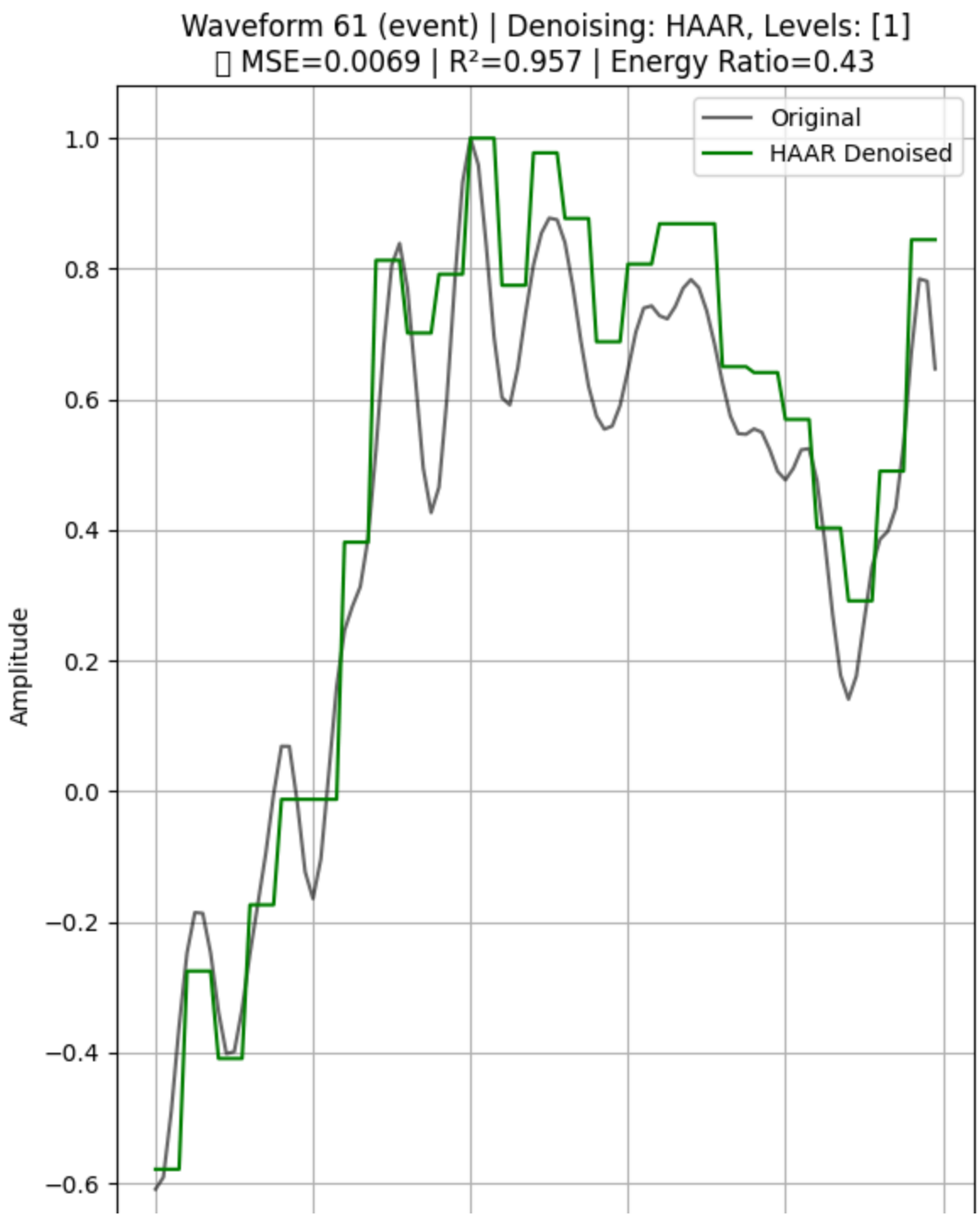


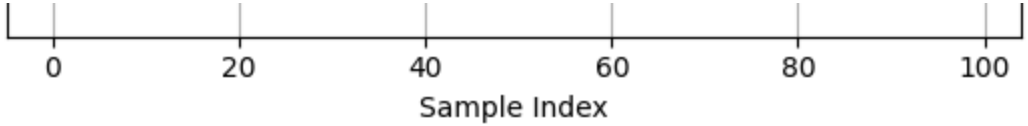

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 9 (event) | Denoising: HAAR, Levels: [1]
 MSE=0.0005 | R²=0.994 | Energy Ratio=0.55
Original
HAAR Denoised
Amplitude
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

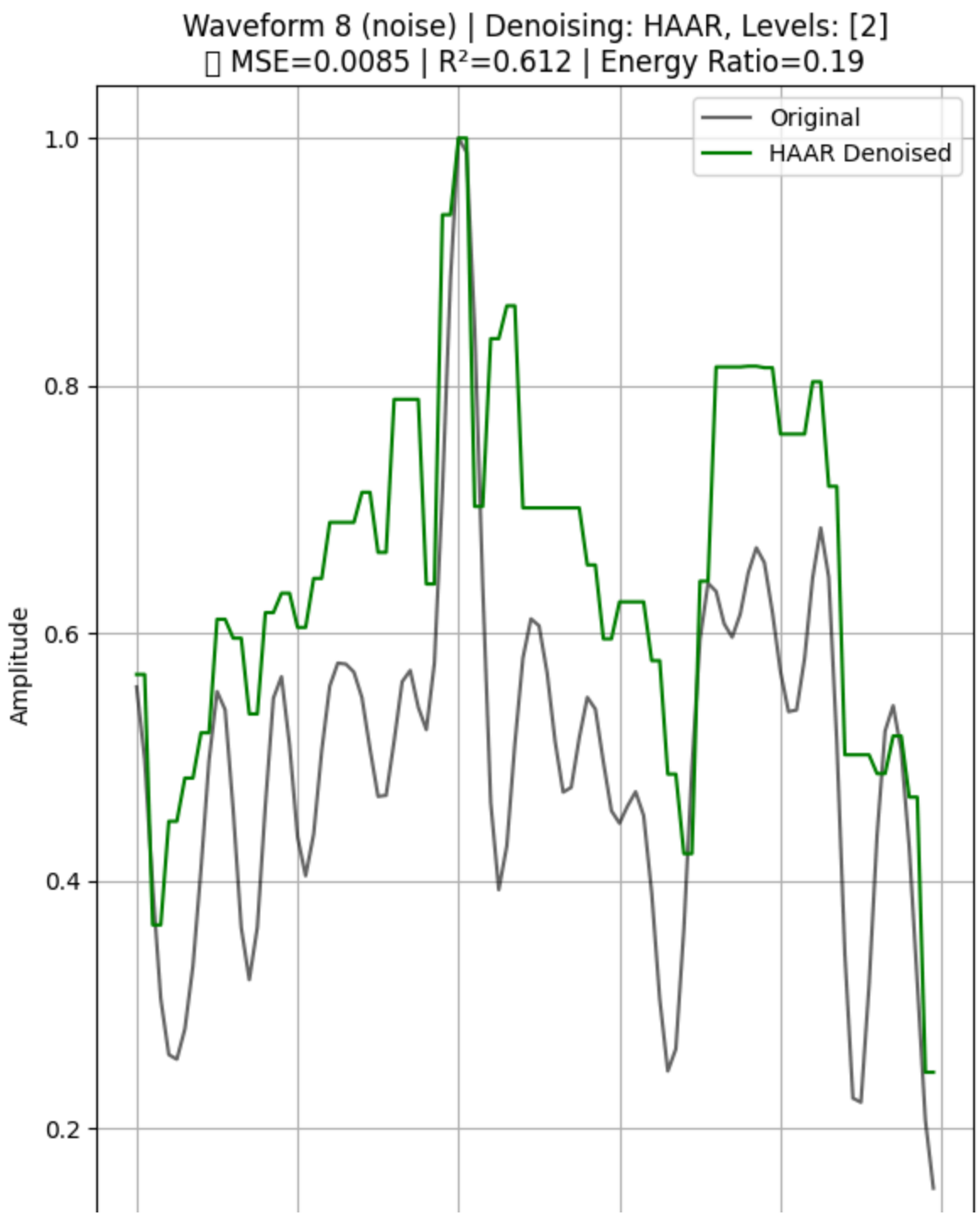


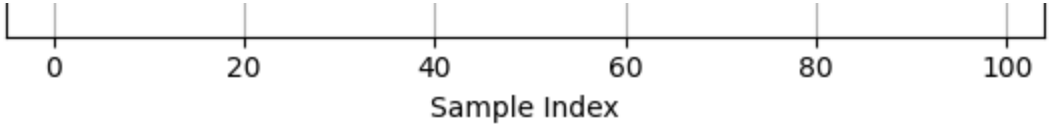

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 61 (event) | Denoising: HAAR, Levels: [1]
 MSE=0.0069 | R²=0.957 | Energy Ratio=0.43
Original
HAAR Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

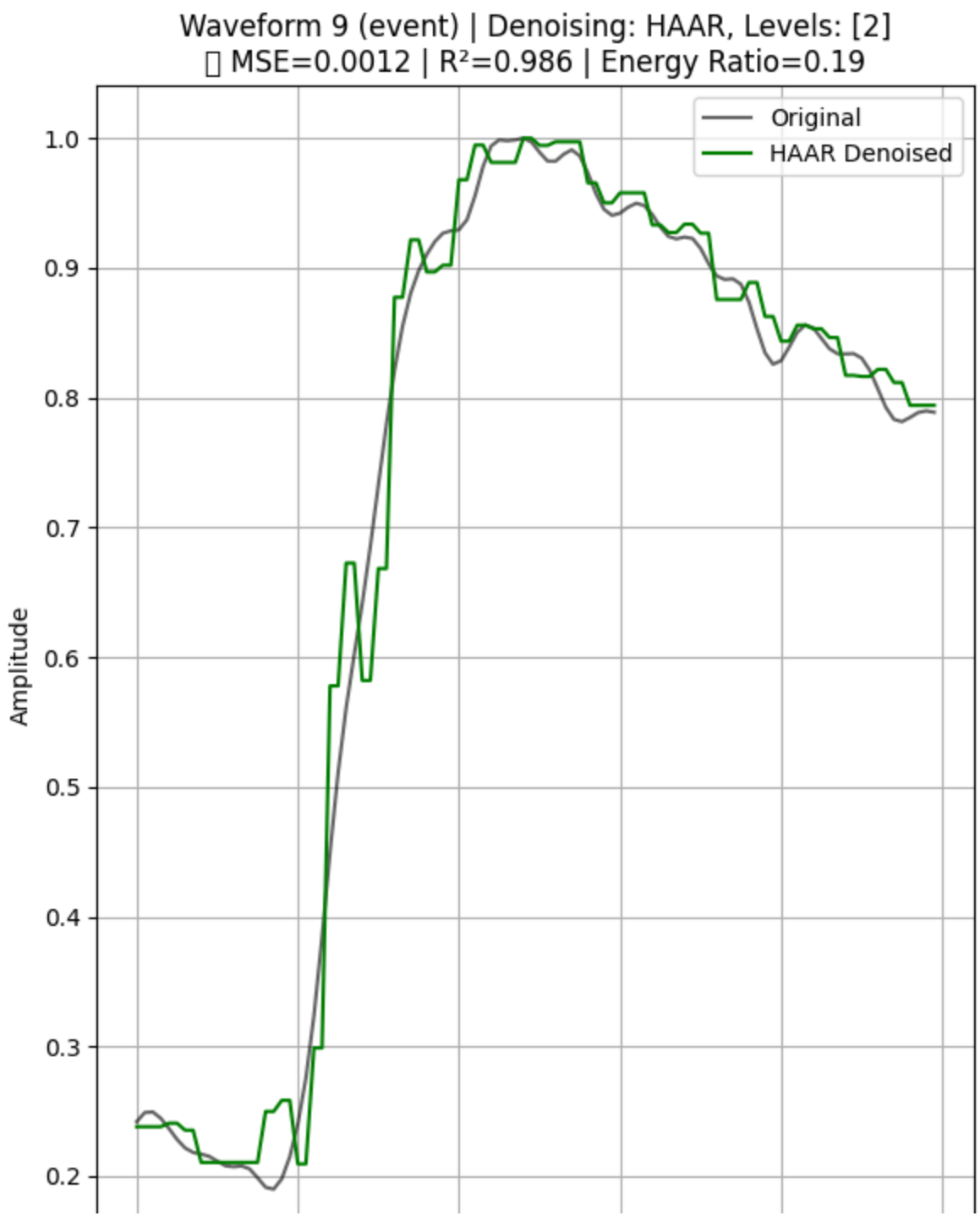


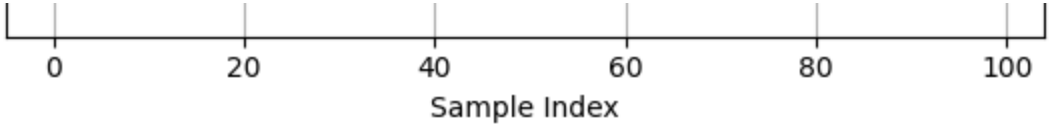

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 8 (noise) | Denoising: HAAR, Levels: [2]
 MSE=0.0085 | R²=0.612 | Energy Ratio=0.19
Original
HAAR Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2

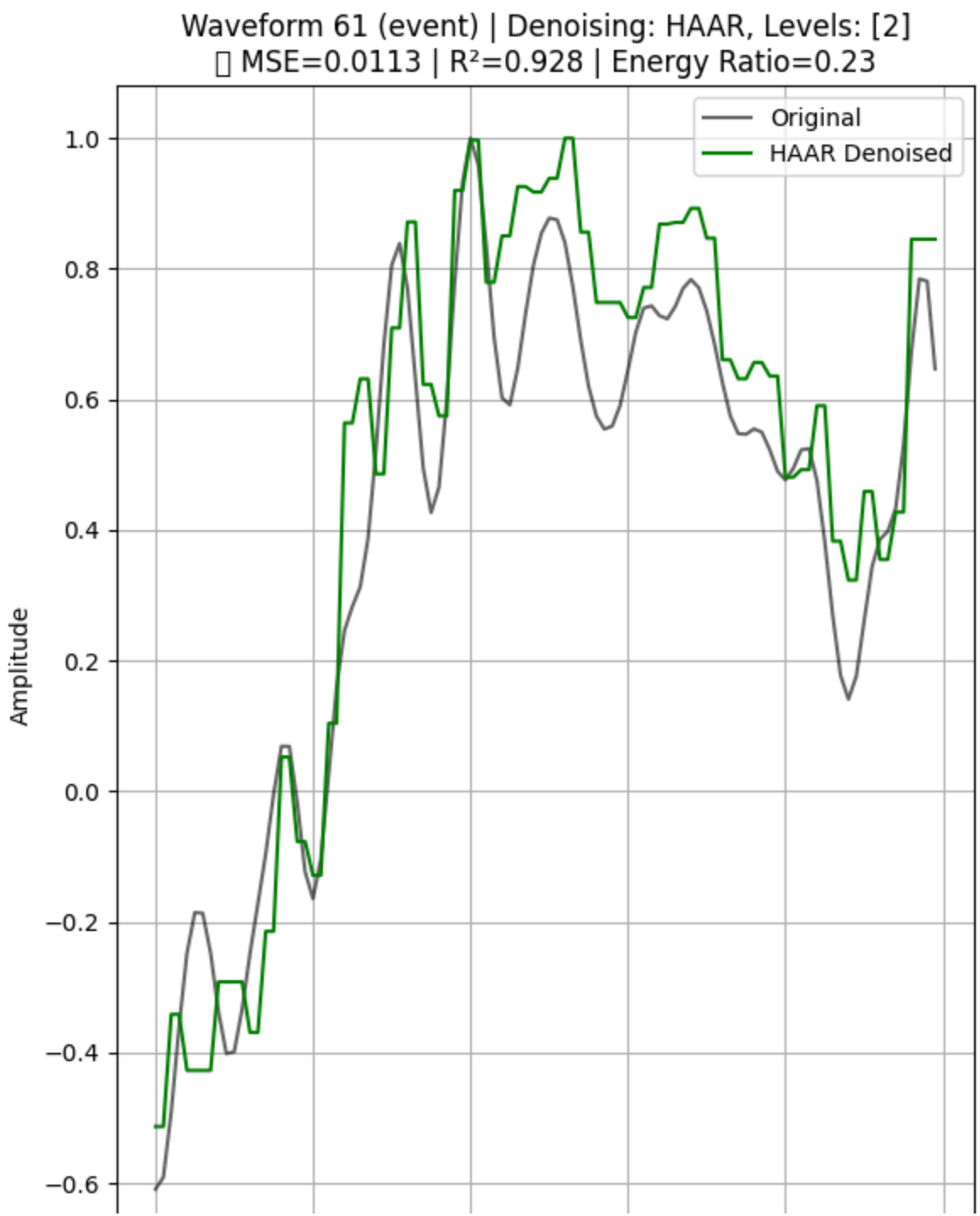


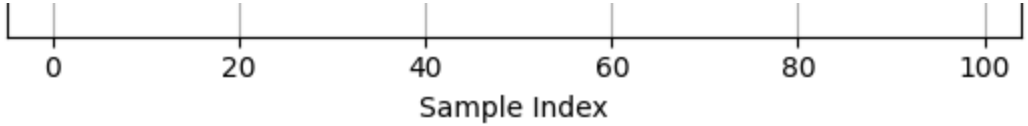

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 9 (event) | Denoising: HAAR, Levels: [2]
 MSE=0.0012 | R²=0.986 | Energy Ratio=0.19
Original
HAAR Denoised
Amplitude
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

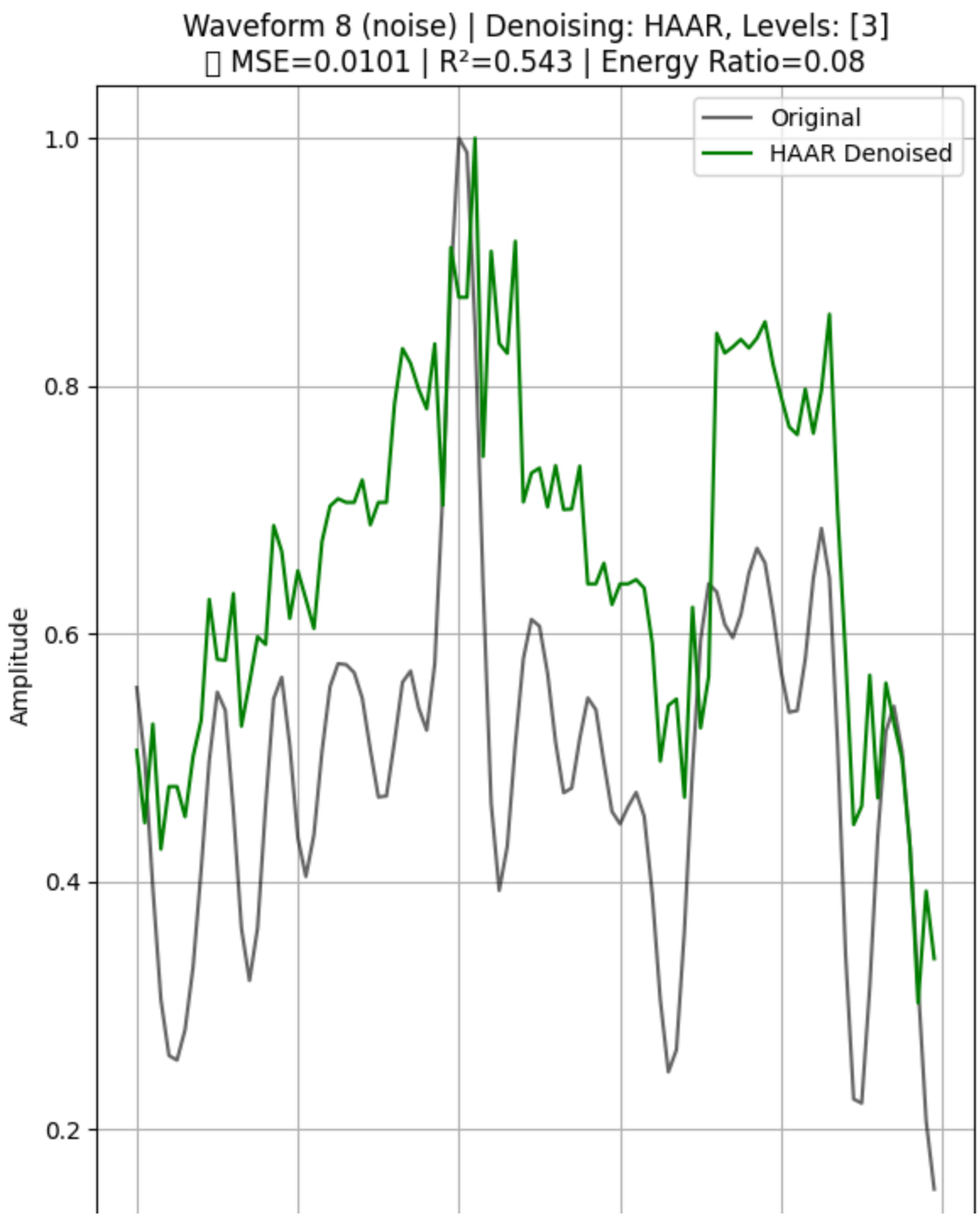


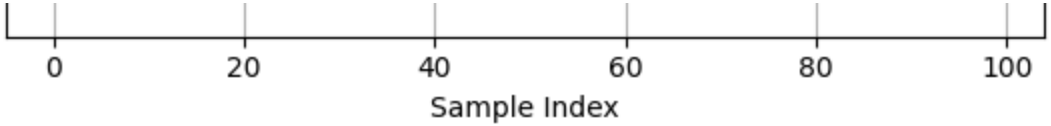

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 61 (event) | Denoising: HAAR, Levels: [2]
 MSE=0.0113 | R²=0.928 | Energy Ratio=0.23
Original
HAAR Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

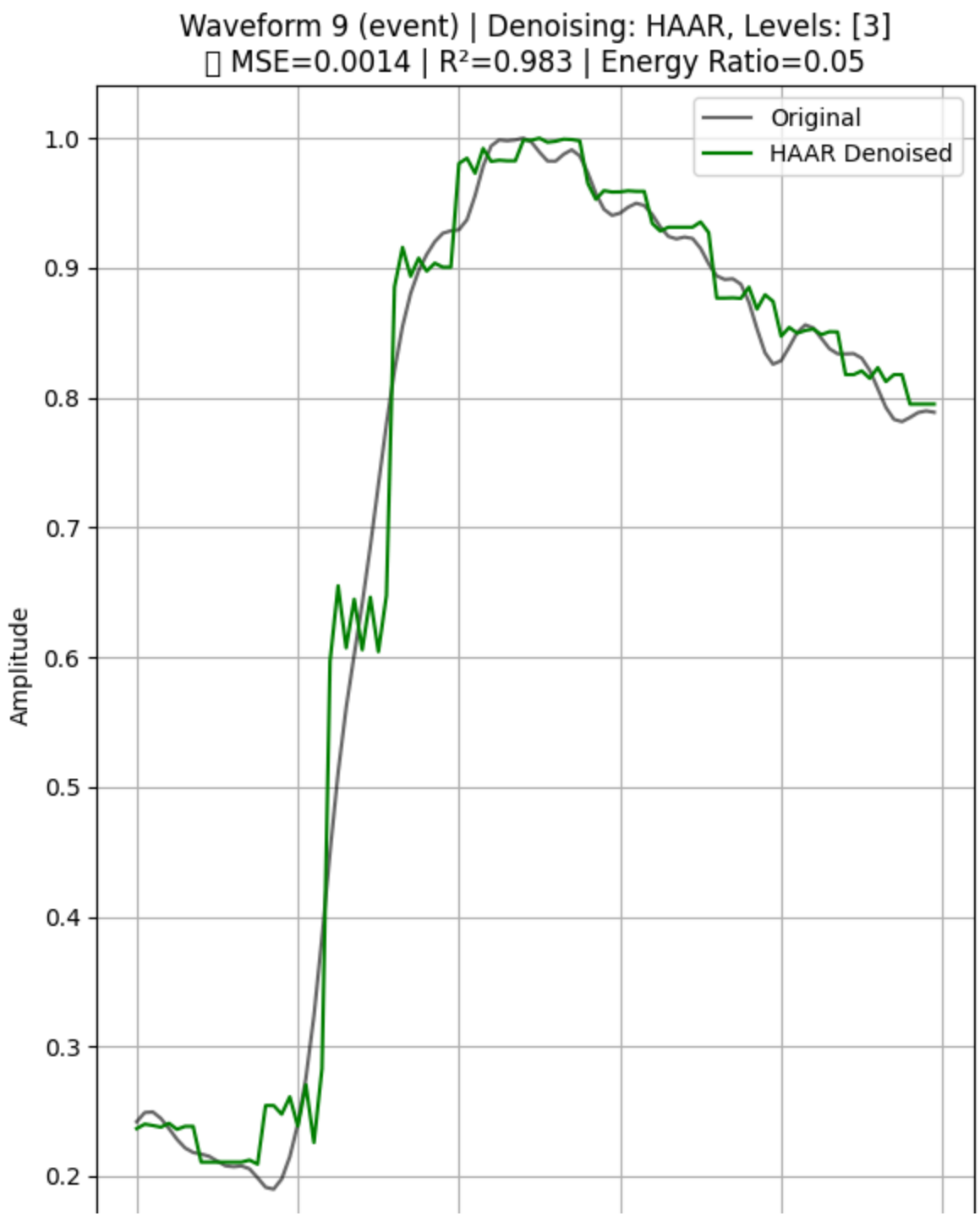


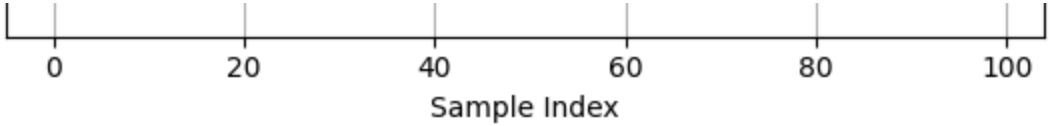

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 8 (noise) | Denoising: HAAR, Levels: [3]
 MSE=0.0101 | R²=0.543 | Energy Ratio=0.08
Original
HAAR Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2

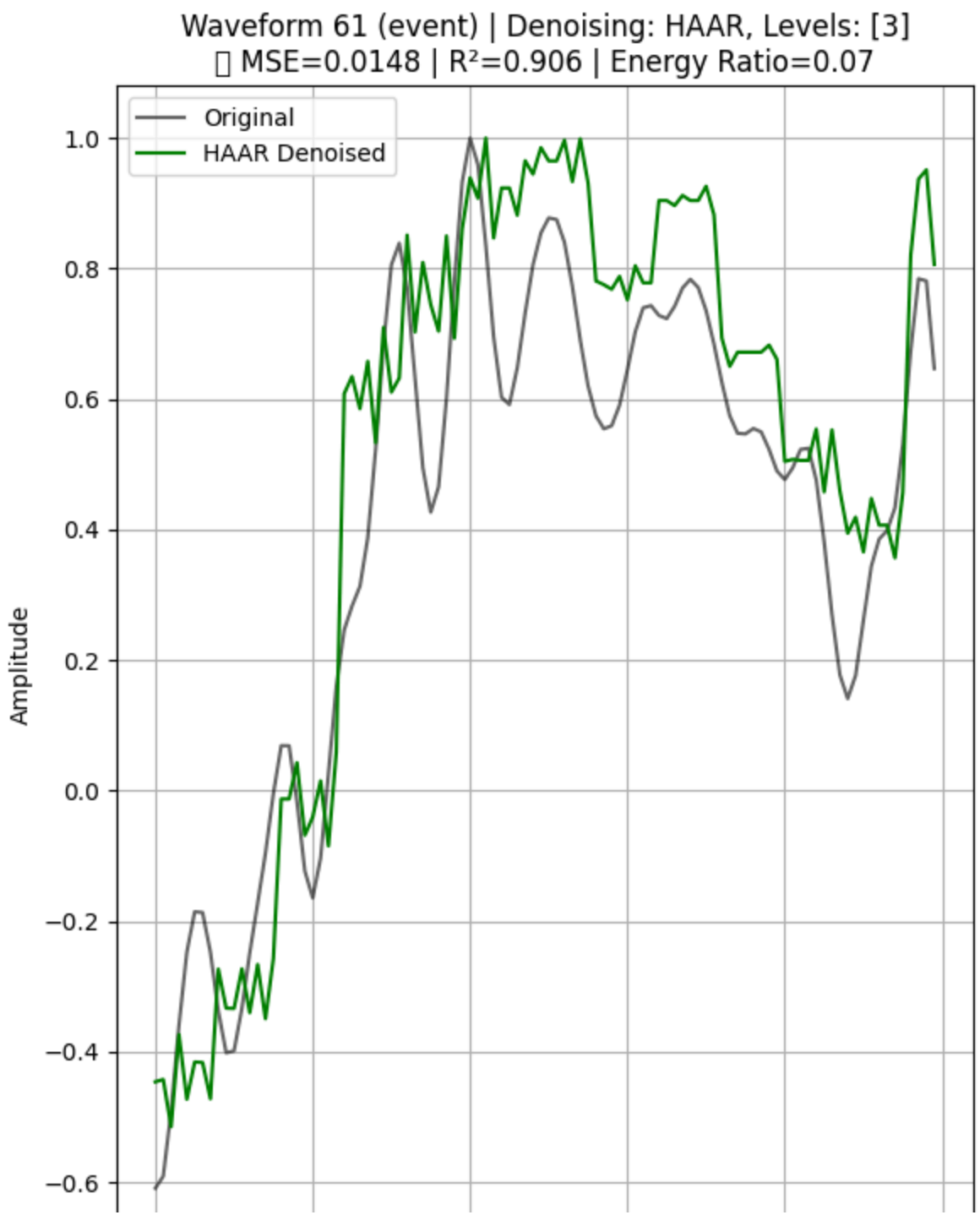


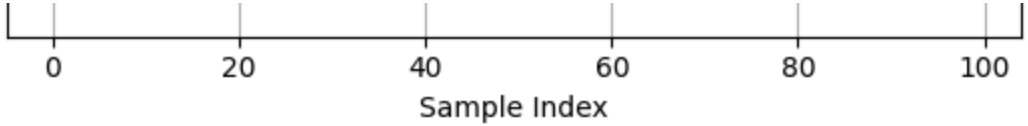

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s)
DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWN
WARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 9 (event) | Denoising: HAAR, Levels: [3]
 MSE=0.0014 | R²=0.983 | Energy Ratio=0.05
Original
HAAR Denoised
Amplitude
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

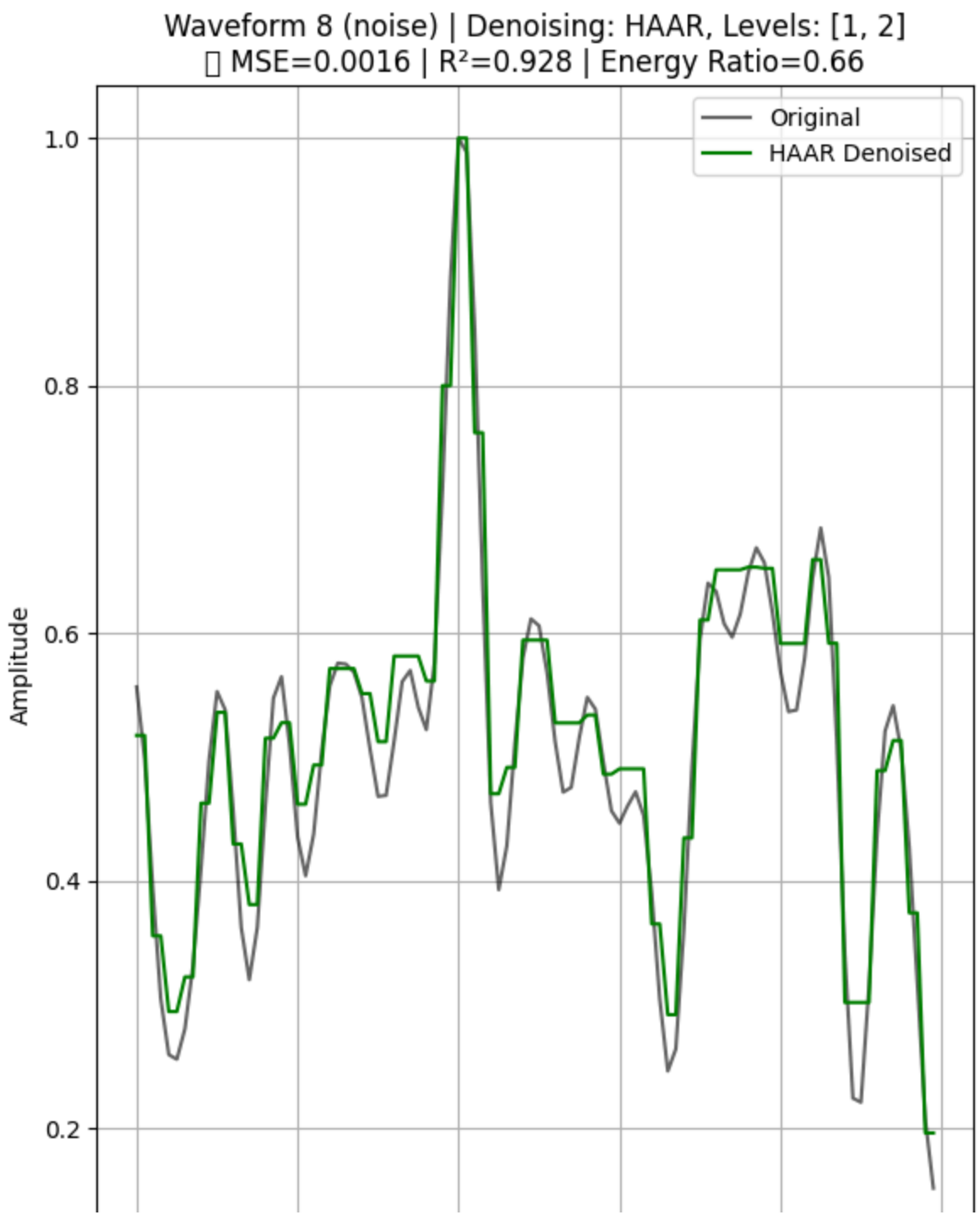


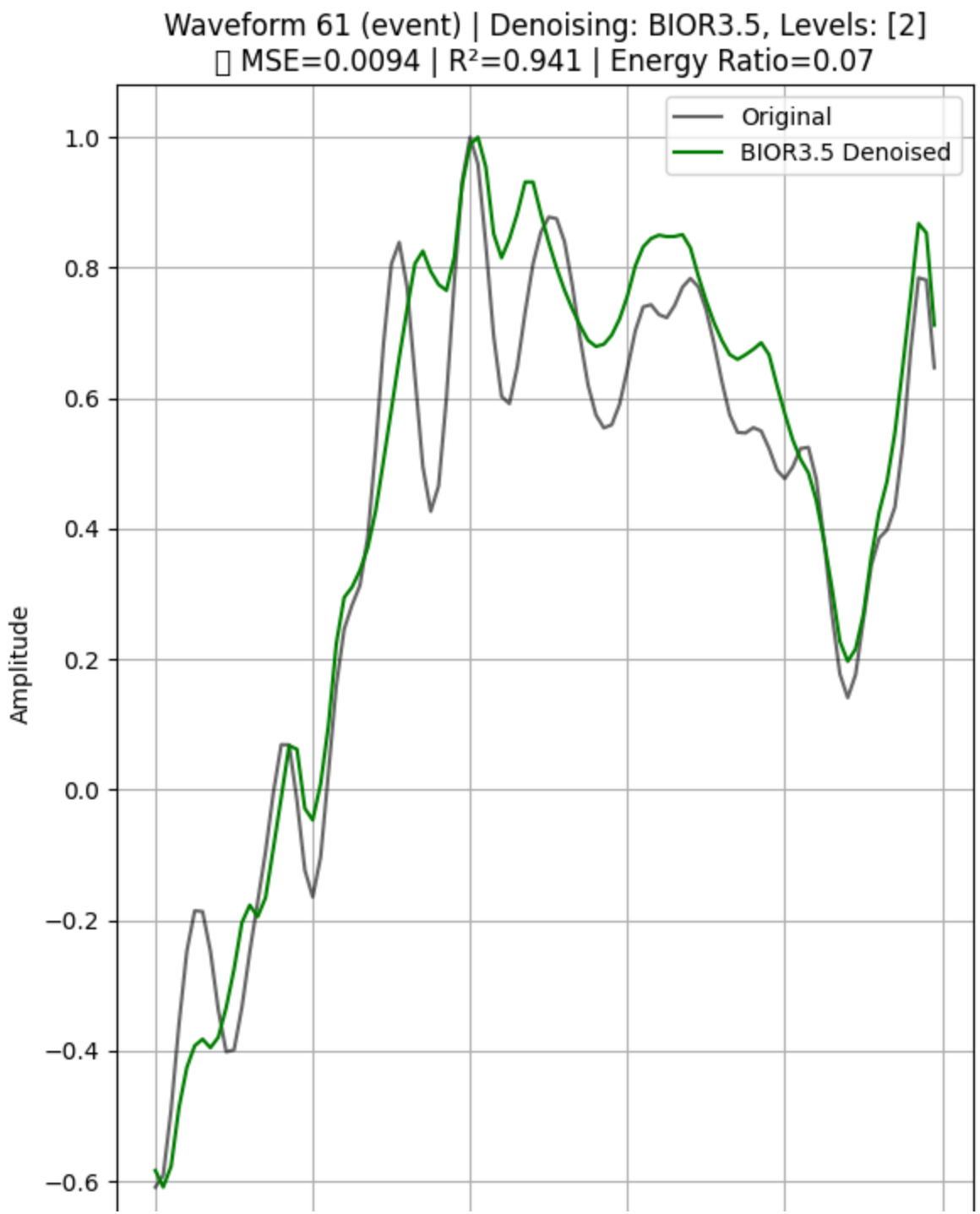

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```

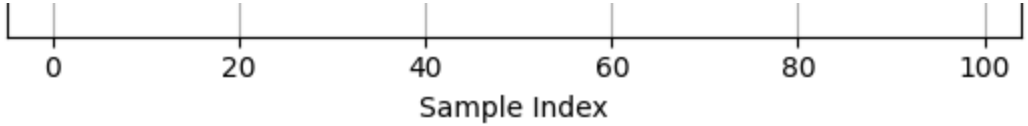

Waveform 61 (event) | Denoising: HAAR, Levels: [3]
 MSE=0.0148 | R²=0.906 | Energy Ratio=0.07
Original
HAAR Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6



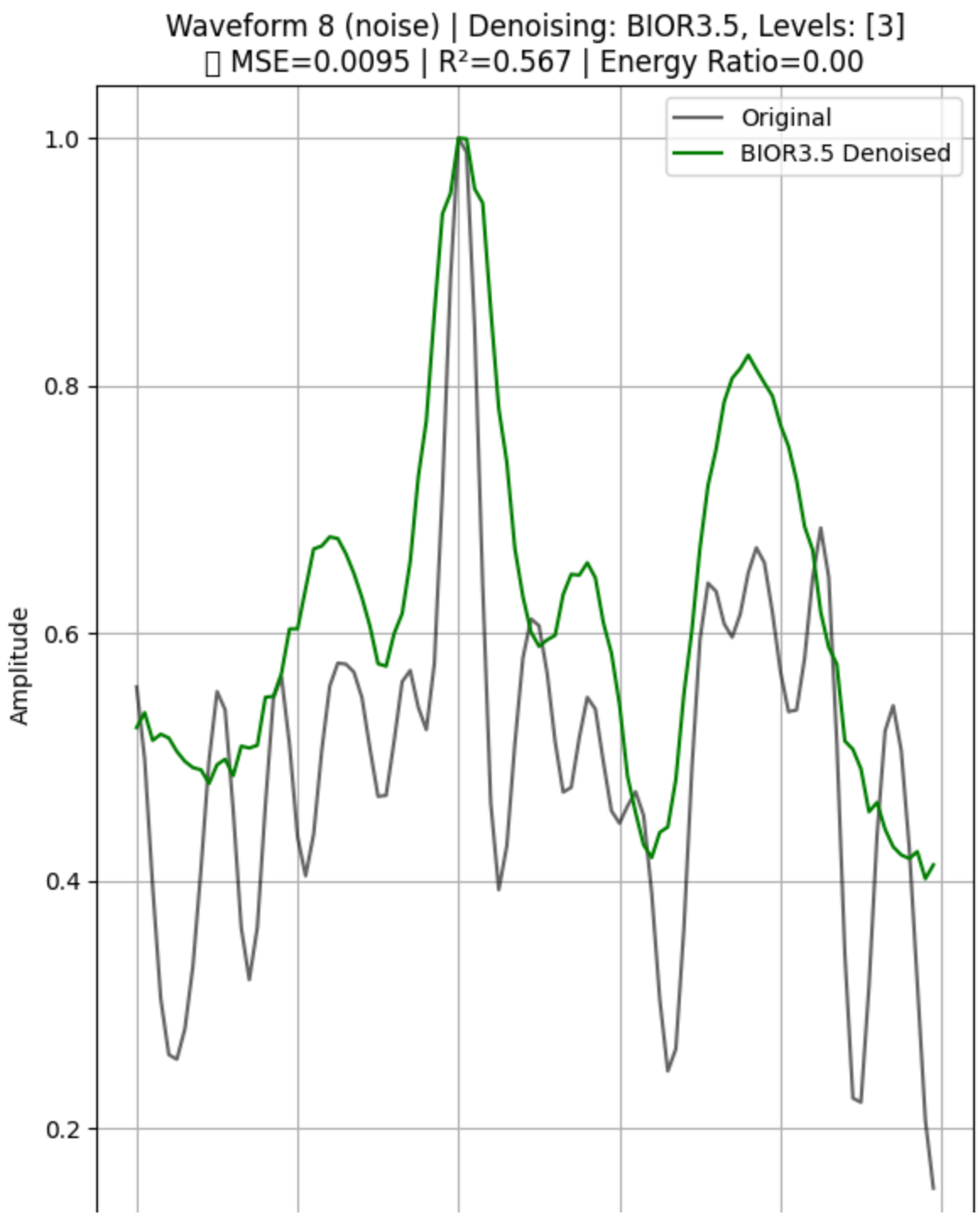

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s)
DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWN
WARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```

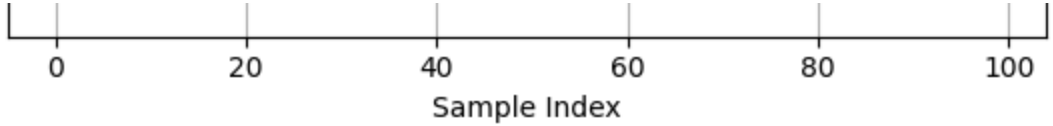

Waveform 8 (noise) | Denoising: HAAR, Levels: [1, 2]
 MSE=0.0016 | R²=0.928 | Energy Ratio=0.66
Original
HAAR Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2


Waveform 61 (event) | Denoising: BIOR3.5, Levels: [2]
 MSE=0.0094 | R²=0.941 | Energy Ratio=0.07
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

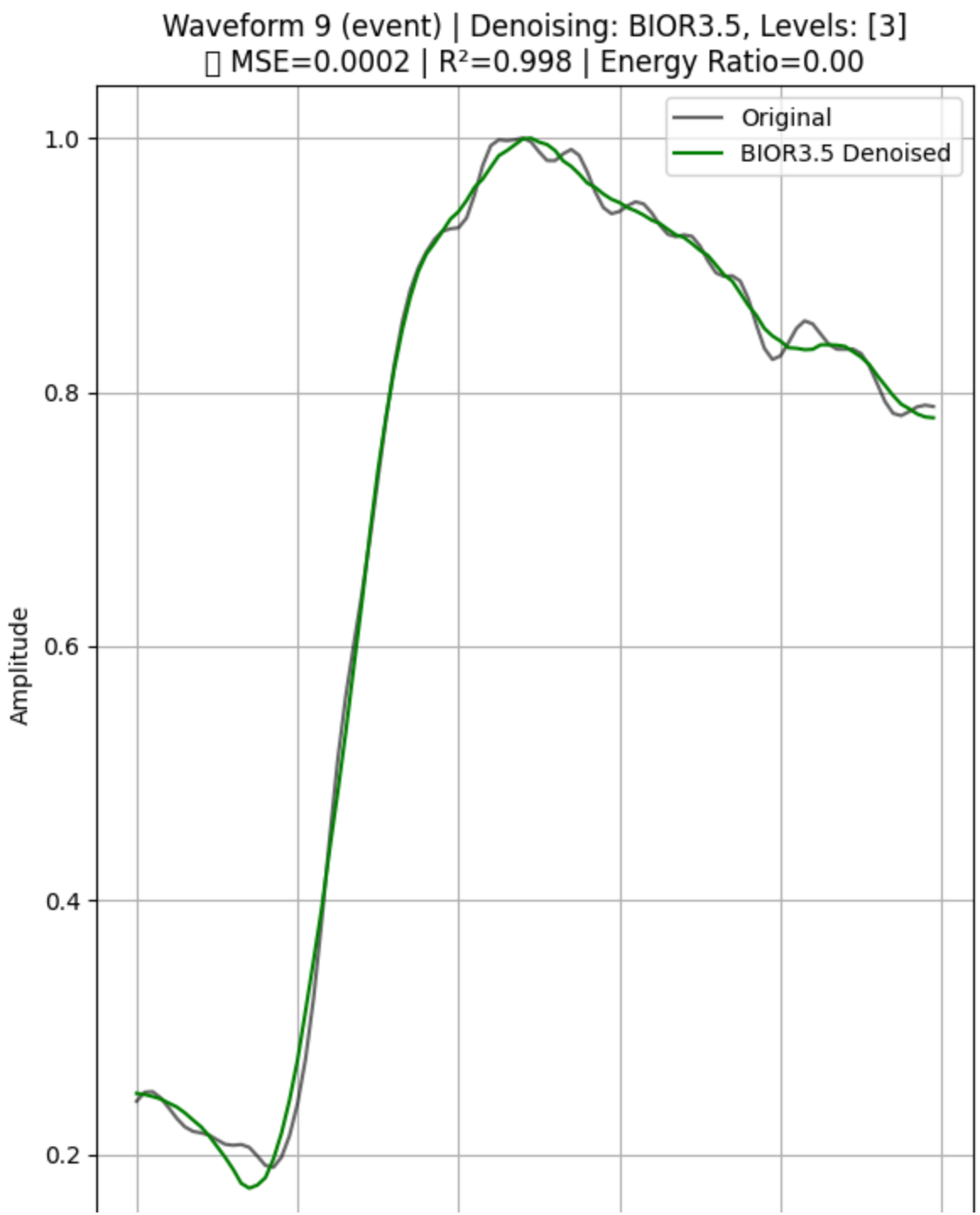


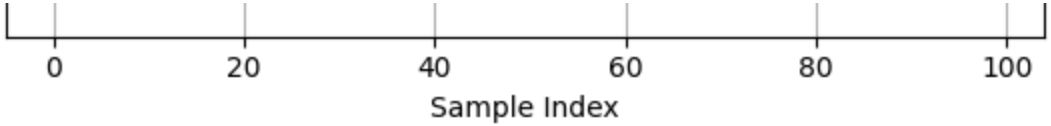

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s)
DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWN
WARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 8 (noise) | Denoising: BIOR3.5, Levels: [3]
 MSE=0.0095 | R²=0.567 | Energy Ratio=0.00
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2

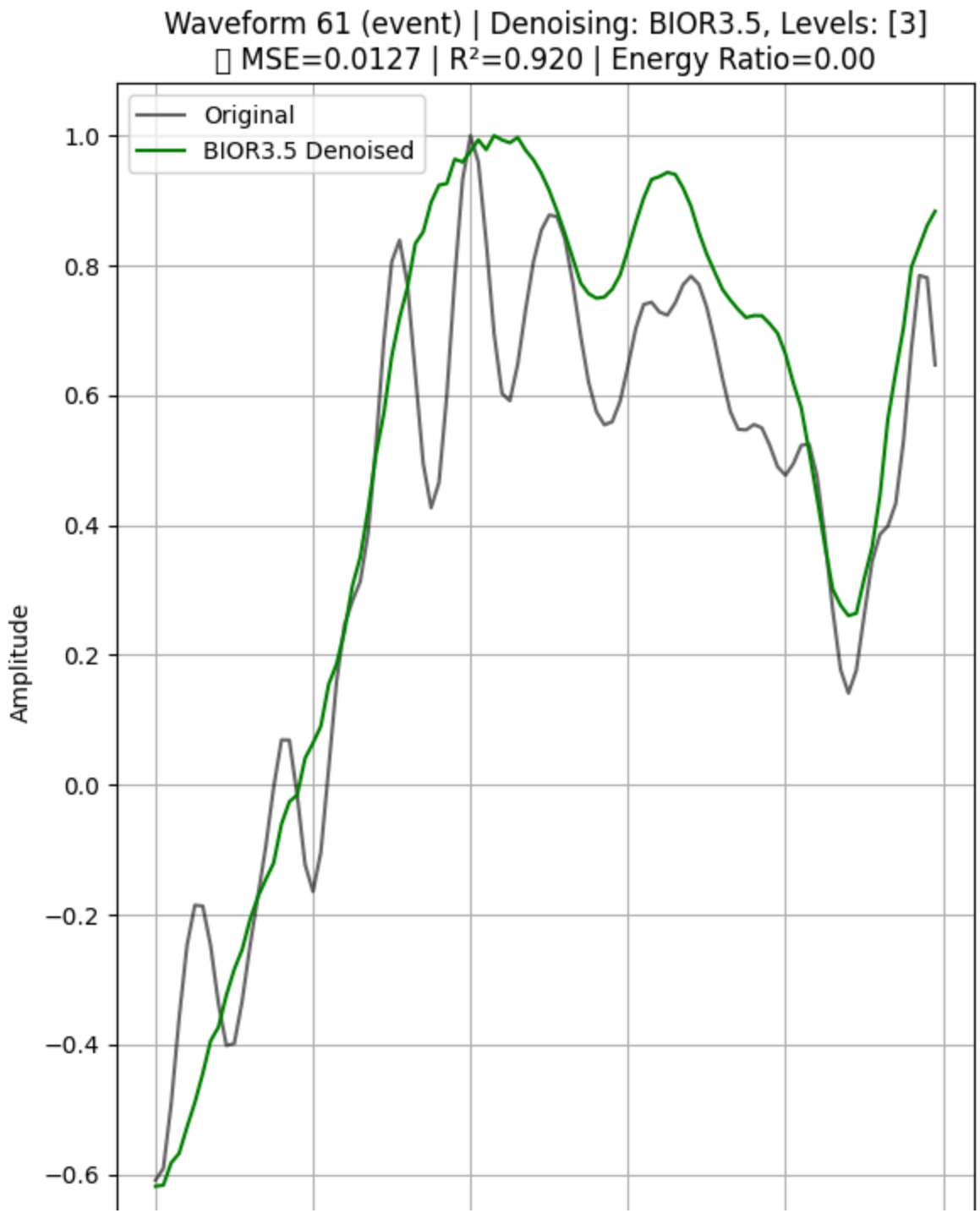


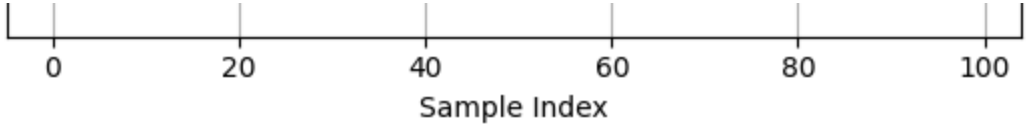

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 9 (event) | Denoising: BIOR3.5, Levels: [3]
 MSE=0.0002 | R²=0.998 | Energy Ratio=0.00
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2

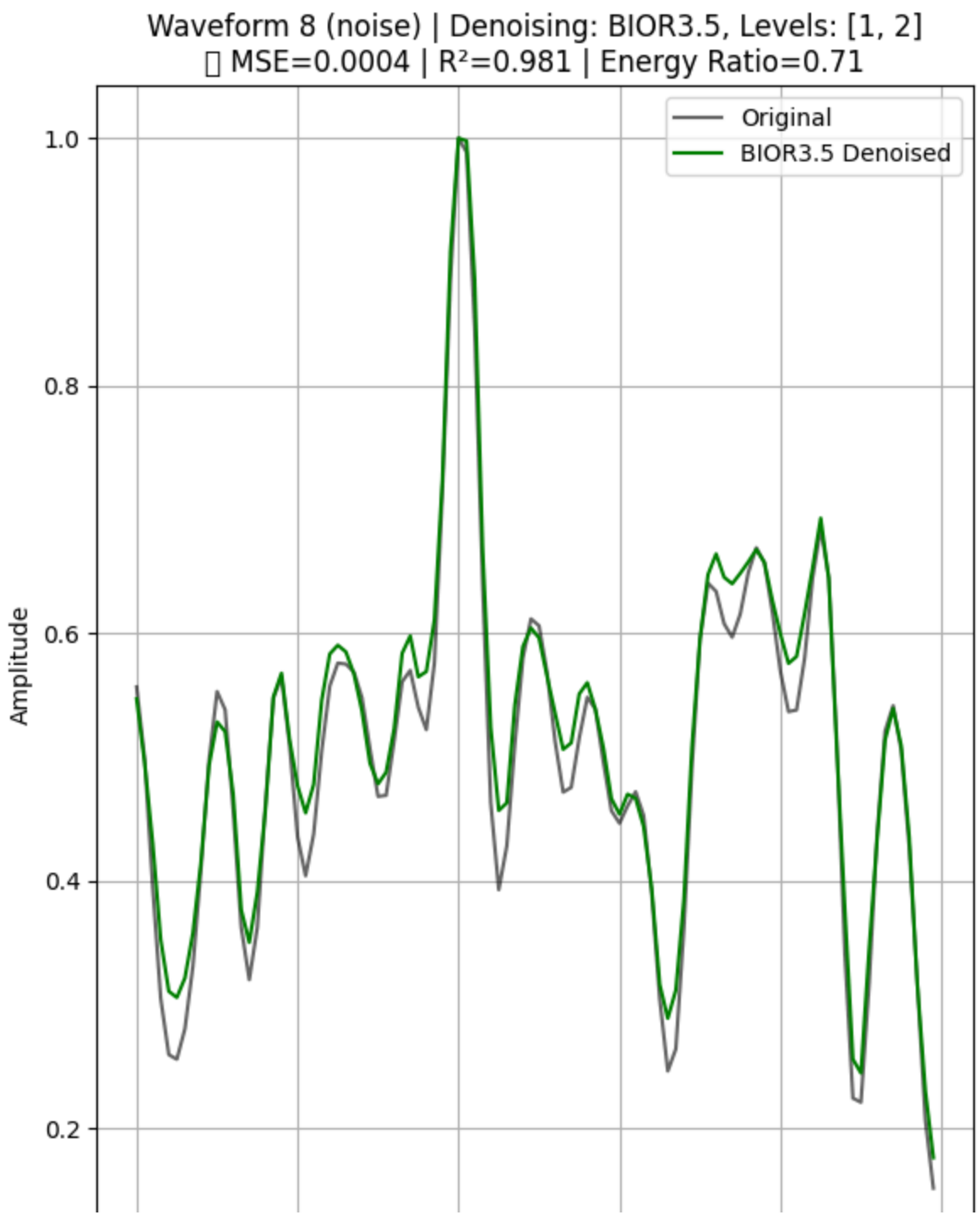


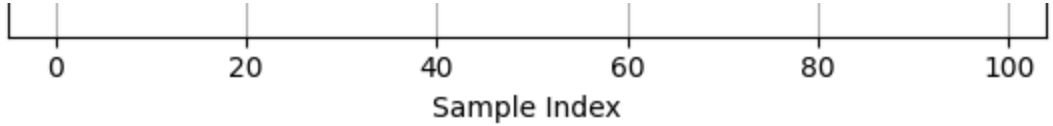

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 61 (event) | Denoising: BIOR3.5, Levels: [3]
 MSE=0.0127 | R²=0.920 | Energy Ratio=0.00
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

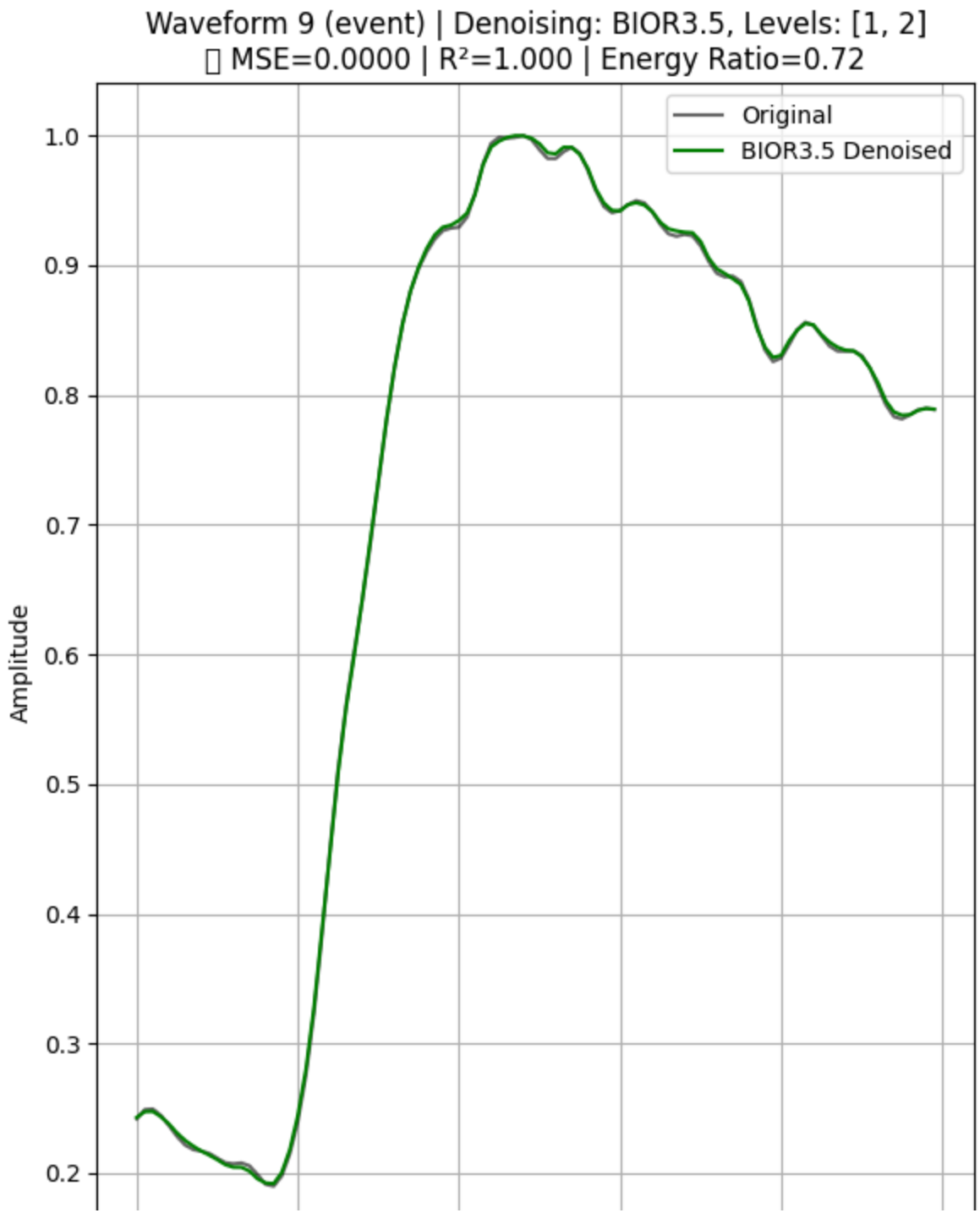


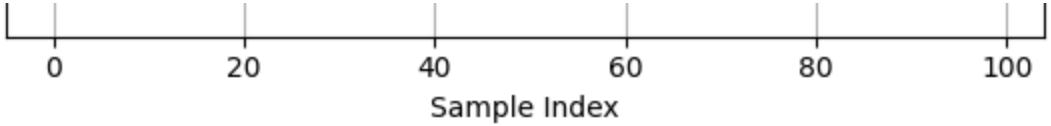

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 8 (noise) | Denoising: BIOR3.5, Levels: [1, 2]
 MSE=0.0004 | R²=0.981 | Energy Ratio=0.71
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2

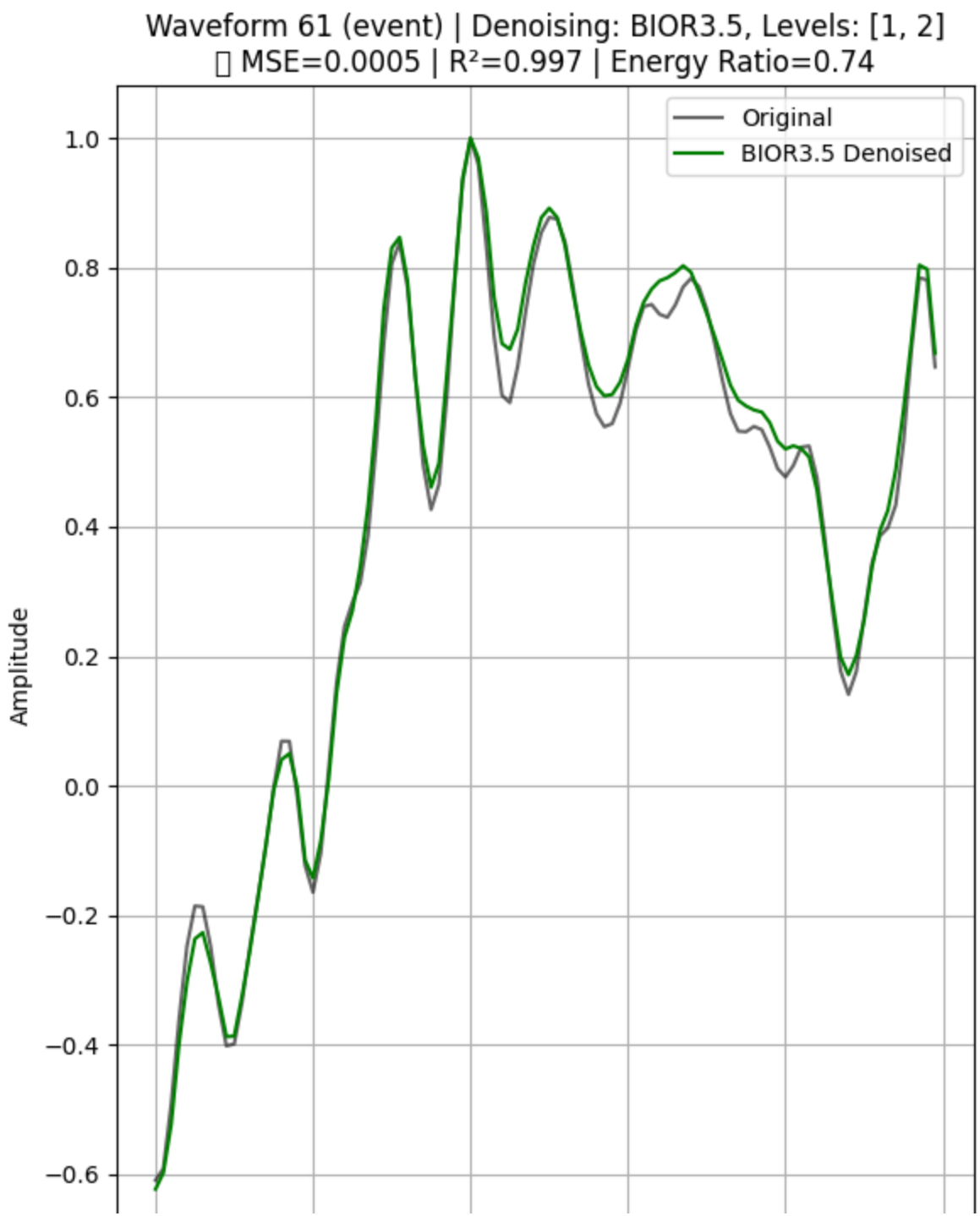


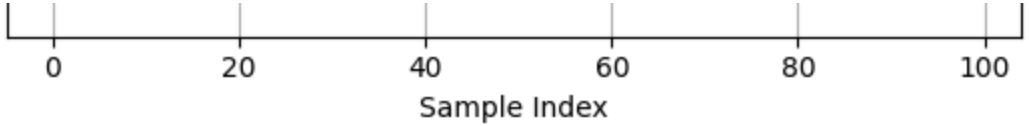

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 9 (event) | Denoising: BIOR3.5, Levels: [1, 2]
 MSE=0.0000 | R²=1.000 | Energy Ratio=0.72
Original
BIOR3.5 Denoised
Amplitude
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2

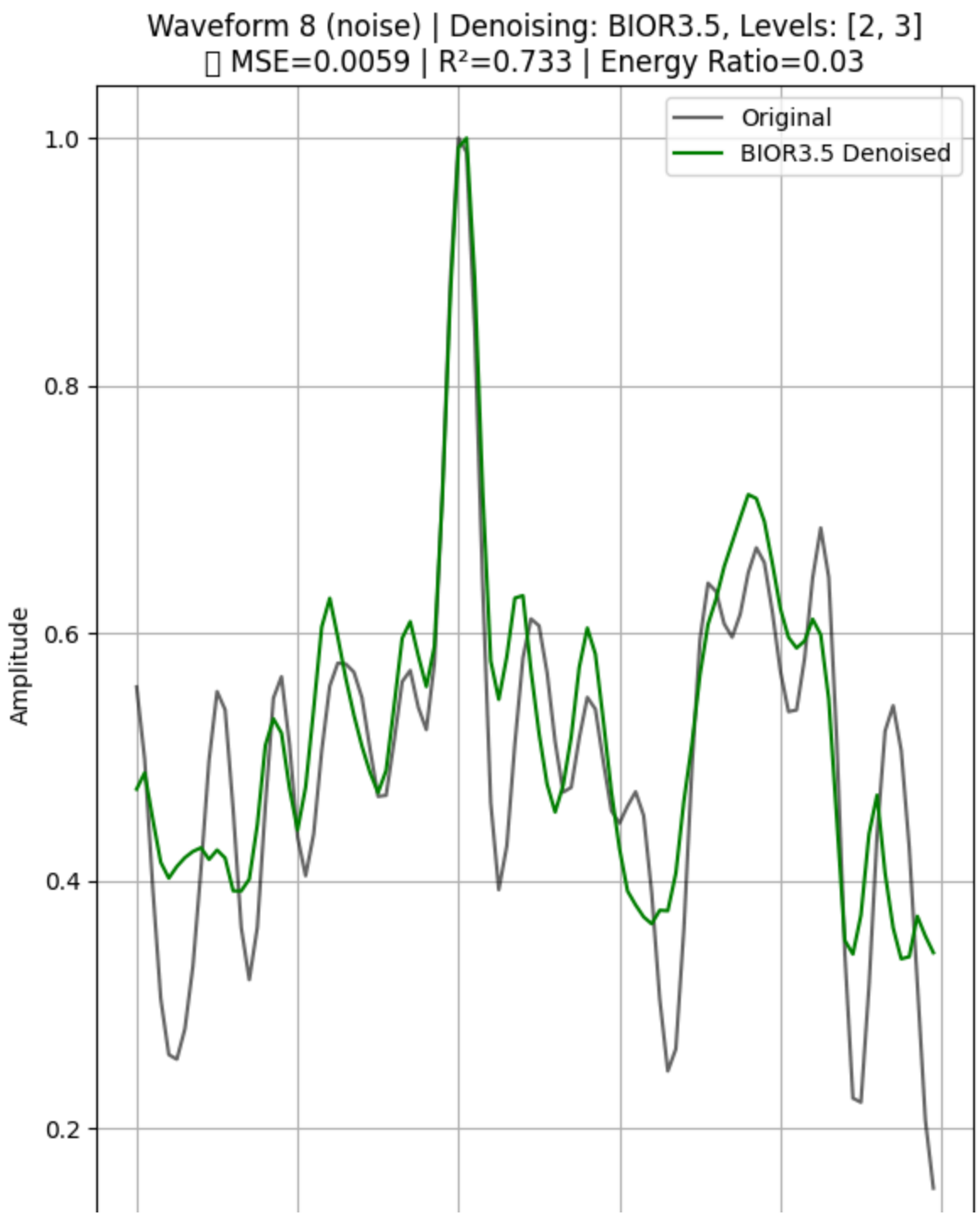


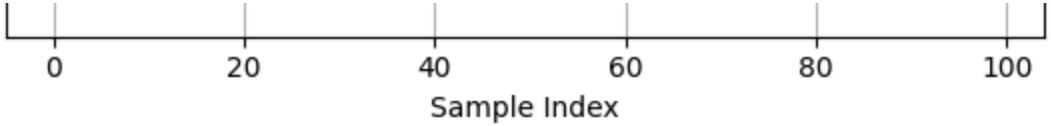

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 61 (event) | Denoising: BIOR3.5, Levels: [1, 2]
 MSE=0.0005 | R²=0.997 | Energy Ratio=0.74
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

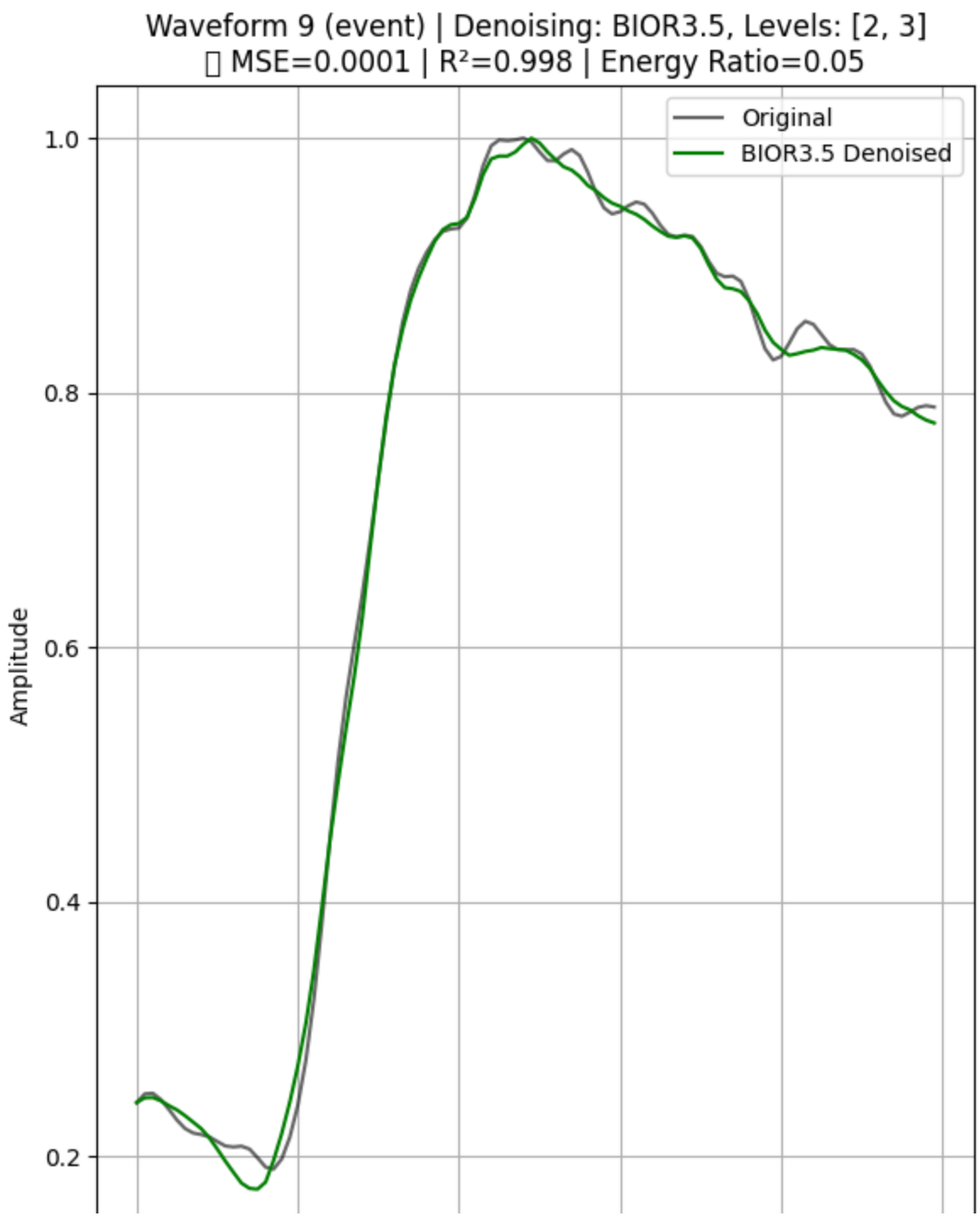


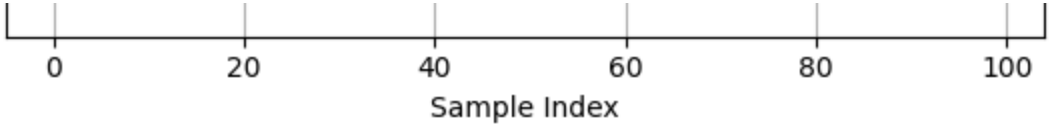

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s)
DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWN
WARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 8 (noise) | Denoising: BIOR3.5, Levels: [2, 3]
 MSE=0.0059 | R²=0.733 | Energy Ratio=0.03
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2

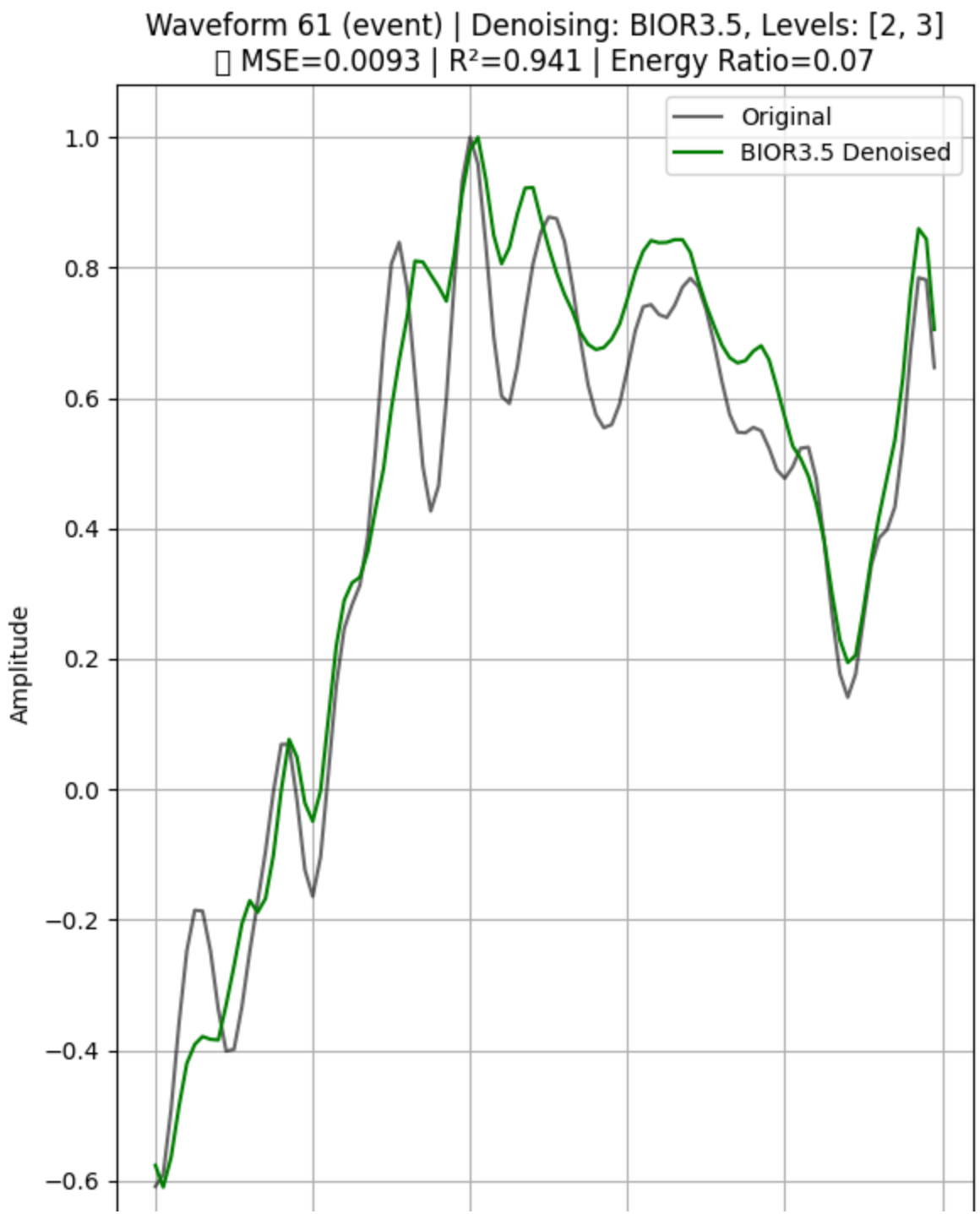


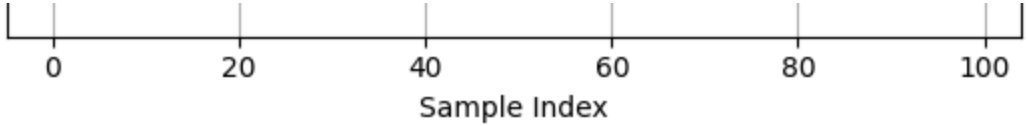

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 9 (event) | Denoising: BIOR3.5, Levels: [2, 3]
 MSE=0.0001 | R²=0.998 | Energy Ratio=0.05
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2

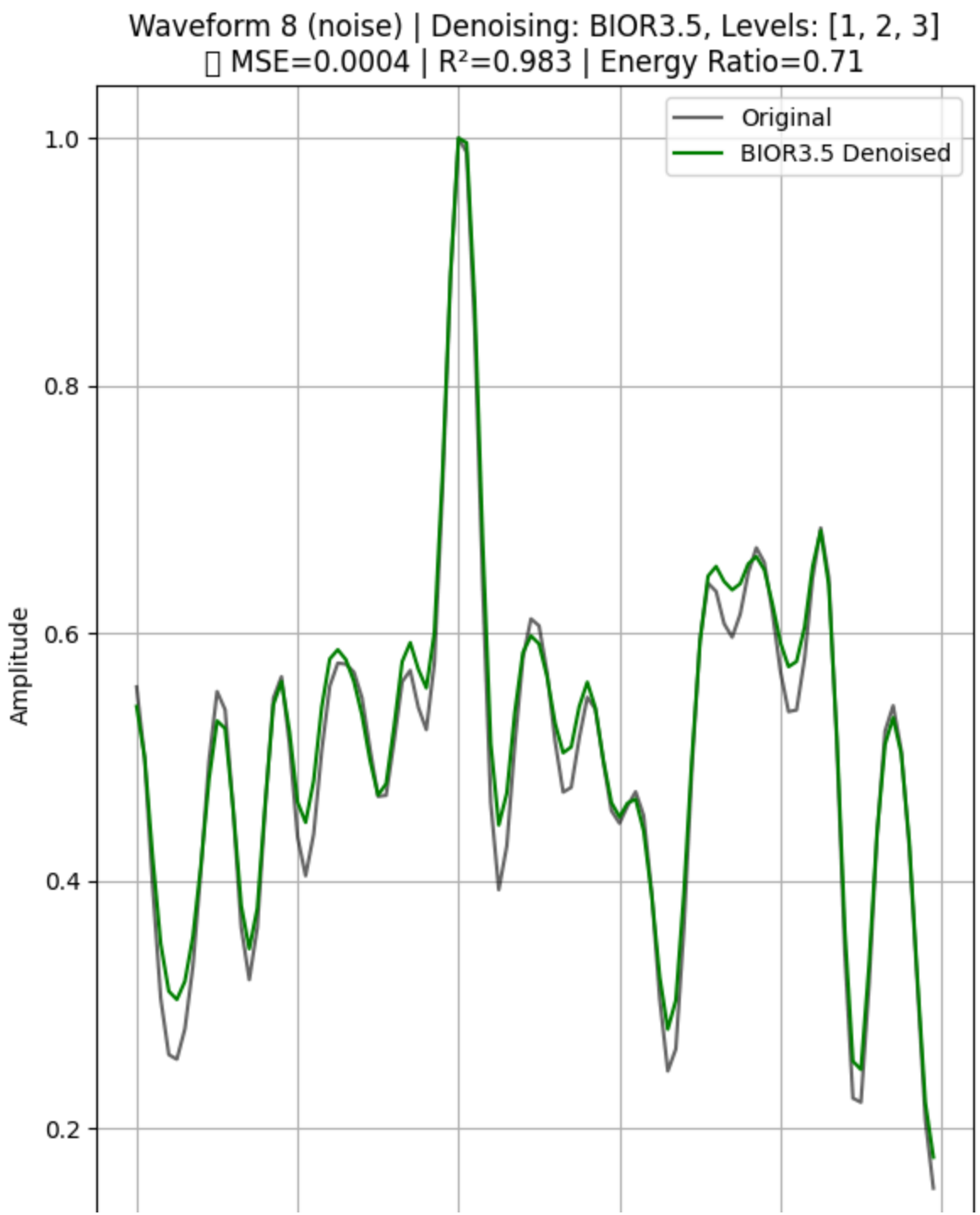


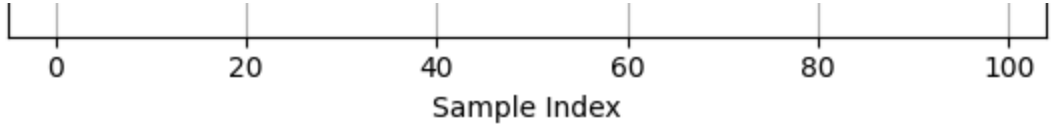

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 61 (event) | Denoising: BIOR3.5, Levels: [2, 3]
 MSE=0.0093 | R²=0.941 | Energy Ratio=0.07
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6

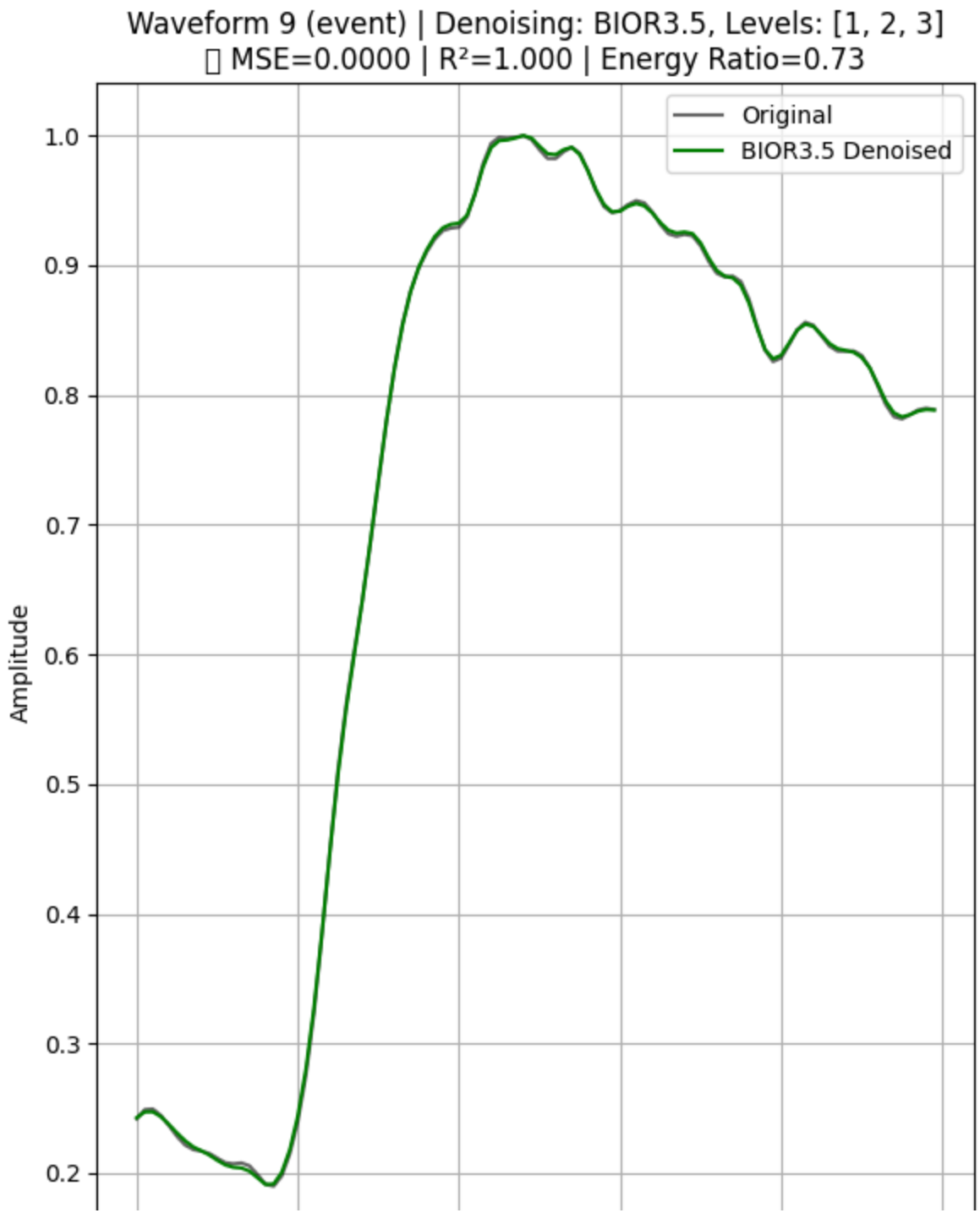


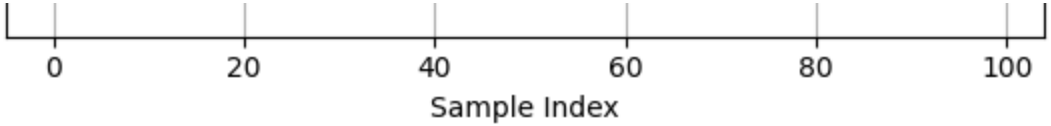

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s)
DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWN
WARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 8 (noise) | Denoising: BIOR3.5, Levels: [1, 2, 3]
 MSE=0.0004 | R²=0.983 | Energy Ratio=0.71
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2

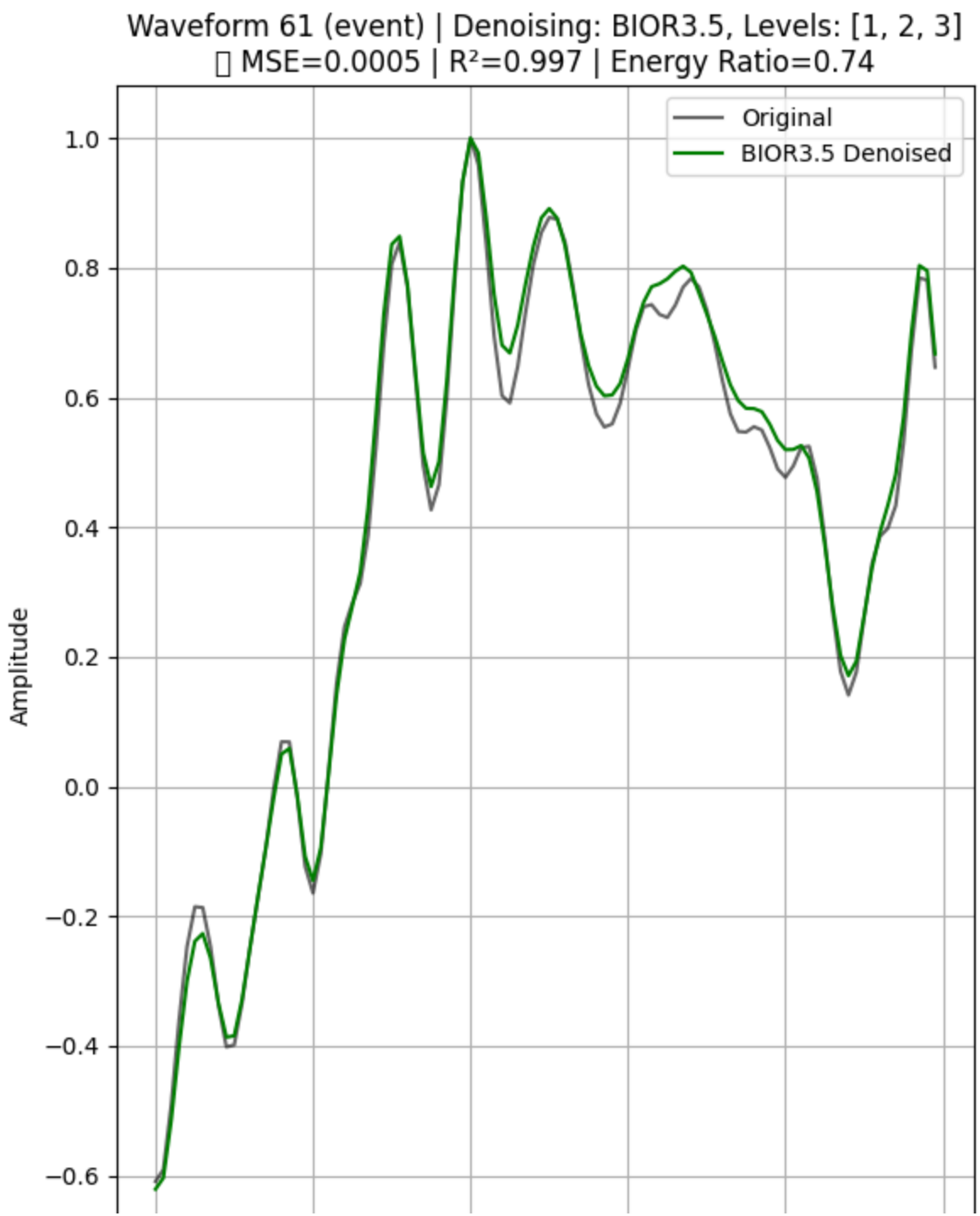


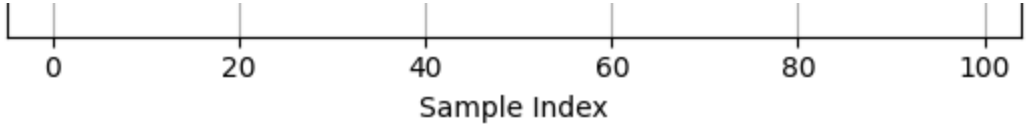

```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 9 (event) | Denoising: BIOR3.5, Levels: [1, 2, 3]
 MSE=0.0000 | R²=1.000 | Energy Ratio=0.73
Original
BIOR3.5 Denoised
Amplitude
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2




```
<ipython-input-121-924e4f272f01>:80: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  plt.tight_layout()
/usr/local/lib/python3.11/dist-packages/IPython/core/pylabtools.py:151: UserWarning: Glyph 128201 (\N{CHART WITH DOWNWARDS TREND}) missing from font(s) DejaVu Sans.
  fig.canvas.print_figure(bytes_io, **kw)
```


Waveform 61 (event) | Denoising: BIOR3.5, Levels: [1, 2, 3]
 MSE=0.0005 | R²=0.997 | Energy Ratio=0.74
Original
BIOR3.5 Denoised
Amplitude
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−0.6



In [ ]:
```python
df_waveform_results = pd.DataFrame(results)
df_waveform_results
#df_waveform_results.sort_values(by="R2", ascending=False)

#  MSE -   Error between original and denoise signal (lower is better)
#  R2 - Proportion of variance in the signal preserved (Closer to 1 is better)
#  Energy Ratio - % of wavelet detail energy retained in reconstruction. Tells you how much structure you kept vs dis
#  Kept levels - Which wavelet detail levels were retained for reconstruction
#  Wavelet - The wavelet family used
#  Waveform - Label the input waveform

# Level Frequency Band  Interpretation
# Level 1 |     Highest frequency |     Captures sharp, fast events (spikes, noise)
# Level 2 |     Mid frequency   Curved | transient features
# Level 3 |     Low-mid frequency       | Smoother, more global signal variation
# Approximation |       Lowest freq |   Long-term trends or baseline
```

Out[ ]:

	MSE	R2	EnergyRatio	KeptLevels	Wavelet	Waveform
0	0.004300	0.804841	0.468641	[1]	haar	Waveform 8 (noise)
1	0.000476	0.994341	0.553475	[1]	haar	Waveform 9 (event)
2	0.003883	0.967851	0.510295	[1]	haar	Waveform 25 (event)
3	0.008545	0.612228	0.188492	[2]	haar	Waveform 8 (noise)
4	0.001169	0.986106	0.185486	[2]	haar	Waveform 9 (event)
5	0.009469	0.921606	0.158165	[2]	haar	Waveform 25 (event)
6	0.010074	0.542829	0.079195	[3]	haar	Waveform 8 (noise)
7	0.001423	0.983085	0.049111	[3]	haar	Waveform 9 (event)
8	0.011255	0.906824	0.052056	[3]	haar	Waveform 25 (event)
9	0.001582	0.928226	0.657133	[1, 2]	haar	Waveform 8 (noise)
10	0.000131	0.998446	0.738961	[1, 2]	haar	Waveform 9 (event)
11	0.001258	0.989584	0.668460	[1, 2]	haar	Waveform 25 (event)
12	0.007355	0.666214	0.267687	[2, 3]	haar	Waveform 8 (noise)
13	0.001078	0.987189	0.234597	[2, 3]	haar	Waveform 9 (event)
14	0.008630	0.928557	0.210221	[2, 3]	haar	Waveform 25 (event)
15	0.000392	0.982212	0.736328	[1, 2, 3]	haar	Waveform 8 (noise)
16	0.000040	0.999530	0.788073	[1, 2, 3]	haar	Waveform 9 (event)
17	0.000418	0.996535	0.720517	[1, 2, 3]	haar	Waveform 25 (event)
18	0.002716	0.876758	0.558260	[1]	db4	Waveform 8 (noise)
19	0.000032	0.999620	0.678908	[1]	db4	Waveform 9 (event)
20	0.001128	0.990658	0.549782	[1]	db4	Waveform 25 (event)
21	0.006874	0.688038	0.141648	[2]	db4	Waveform 8 (noise)

	MSE	R2	EnergyRatio	KeptLevels	Wavelet	Waveform
22	0.000179	0.997868	0.092696	[2]	db4	Waveform 9 (event)
23	0.003549	0.970616	0.151899	[2]	db4	Waveform 25 (event)
24	0.009101	0.586970	0.004834	[3]	db4	Waveform 8 (noise)
25	0.000204	0.997578	0.003961	[3]	db4	Waveform 9 (event)
26	0.004424	0.963377	0.008765	[3]	db4	Waveform 25 (event)
27	0.000451	0.979533	0.699908	[1, 2]	db4	Waveform 8 (noise)
28	0.000006	0.999923	0.771604	[1, 2]	db4	Waveform 9 (event)
29	0.000207	0.998284	0.701681	[1, 2]	db4	Waveform 25 (event)
30	0.006788	0.691943	0.146483	[2, 3]	db4	Waveform 8 (noise)
31	0.000178	0.997882	0.096657	[2, 3]	db4	Waveform 9 (event)
32	0.003483	0.971168	0.160664	[2, 3]	db4	Waveform 25 (event)
33	0.000364	0.983485	0.704742	[1, 2, 3]	db4	Waveform 8 (noise)
34	0.000005	0.999936	0.775565	[1, 2, 3]	db4	Waveform 9 (event)
35	0.000142	0.998827	0.710446	[1, 2, 3]	db4	Waveform 25 (event)
36	0.002516	0.885805	0.599587	[1]	coif5	Waveform 8 (noise)
37	0.000023	0.999729	0.587638	[1]	coif5	Waveform 9 (event)
38	0.001207	0.990011	0.627421	[1]	coif5	Waveform 25 (event)
39	0.008546	0.612155	0.109209	[2]	coif5	Waveform 8 (noise)
40	0.000107	0.998732	0.122982	[2]	coif5	Waveform 9 (event)
41	0.005586	0.953752	0.099229	[2]	coif5	Waveform 25 (event)
42	0.010608	0.518604	0.000212	[3]	coif5	Waveform 8 (noise)
43	0.000126	0.998507	0.001118	[3]	coif5	Waveform 9 (event)

	MSE	R2	EnergyRatio	KeptLevels	Wavelet	Waveform
44	0.006567	0.945631	0.000944	[3]	coif5	Waveform 25 (event)
45	0.000414	0.981195	0.708796	[1, 2]	coif5	Waveform 8 (noise)
46	0.000003	0.999962	0.710620	[1, 2]	coif5	Waveform 9 (event)
47	0.000184	0.998476	0.726649	[1, 2]	coif5	Waveform 25 (event)
48	0.008540	0.612416	0.109422	[2, 3]	coif5	Waveform 8 (noise)
49	0.000106	0.998736	0.124100	[2, 3]	coif5	Waveform 9 (event)
50	0.005572	0.953873	0.100172	[2, 3]	coif5	Waveform 25 (event)
51	0.000408	0.981469	0.709008	[1, 2, 3]	coif5	Waveform 8 (noise)
52	0.000003	0.999964	0.711737	[1, 2, 3]	coif5	Waveform 9 (event)
53	0.000173	0.998564	0.727593	[1, 2, 3]	coif5	Waveform 25 (event)

In [ ]:
```
#
```