

## da\_Mushroom\_25-05-08\_0326-sound\_simulation3

May 14, 2025

```
[1]: import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os

# Set file path
file_path = '../data/Mushroom_25-05-08_0326.lvm'

# Check if file exists
if not os.path.exists(file_path):
    print(f"Error: File {file_path} does not exist")
else:
    # Read LVM file
    # LVM files are tab-separated text files without header
    data = pd.read_csv(file_path, sep='\t', header=None)

    # Display basic information about the data
    print(f"Data shape: {data.shape}")
    print("\nFirst 5 rows of data:")
    print(data.head())

    # Based on file content, we need to name the columns
    # Assuming first column is timestamp, others are sensor data
    columns = ['Timestamp'] + [f'Sensor_{i}' for i in range(1, data.shape[1])]
    data.columns = columns

    print("\nData after renaming columns:")
    print(data.head())
```

Data shape: (1084420, 10)

First 5 rows of data:

	0	1	2	3	4	5	6	\
0	120386.537600	-0.017416	-0.015052	-0.035177	-0.024526	-0.022283	-0.014307	
1	120386.714606	-0.017413	-0.015028	-0.035177	-0.024510	-0.022269	-0.014292	
2	120386.889620	-0.017420	-0.015043	-0.035157	-0.024524	-0.022270	-0.014293	
3	120387.088626	-0.017404	-0.015036	-0.035172	-0.024527	-0.022294	-0.014290	
4	120387.273636	-0.017437	-0.015036	-0.035183	-0.024523	-0.022269	-0.014280	

	7	8	9
0	-0.035494	0.001486	NaN
1	-0.035491	0.001480	NaN
2	-0.035494	0.001500	NaN
3	-0.035498	0.001483	NaN
4	-0.035490	0.001495	NaN

Data after renaming columns:

	Timestamp	Sensor_1	Sensor_2	Sensor_3	Sensor_4	Sensor_5	Sensor_6	\
0	120386.537600	-0.017416	-0.015052	-0.035177	-0.024526	-0.022283	-0.014307	
1	120386.714606	-0.017413	-0.015028	-0.035177	-0.024510	-0.022269	-0.014292	
2	120386.889620	-0.017420	-0.015043	-0.035157	-0.024524	-0.022270	-0.014293	
3	120387.088626	-0.017404	-0.015036	-0.035172	-0.024527	-0.022294	-0.014290	
4	120387.273636	-0.017437	-0.015036	-0.035183	-0.024523	-0.022269	-0.014280	

	Sensor_7	Sensor_8	Sensor_9
0	-0.035494	0.001486	NaN
1	-0.035491	0.001480	NaN
2	-0.035494	0.001500	NaN
3	-0.035498	0.001483	NaN
4	-0.035490	0.001495	NaN

```
[2]: # Extract date and time information from the filename
file_name = os.path.basename(file_path) # Get the filename
date_time_str = file_name.split('_')[1:3] # Extract date and time parts
date_str = date_time_str[0].replace('-', '/') # Format date
time_str = date_time_str[1].replace('.lvm', '') # Format time
# Parse time string, first two digits are hours, last two are minutes
hour = time_str[:2]
minute = time_str[2:]
formatted_time = f"{hour}:{minute}"

# Use actual timestamps and convert to specific times
actual_time = data['Timestamp']
# Calculate seconds relative to start time
start_time = actual_time.iloc[0]
relative_seconds = actual_time - start_time

# Create specific time labels
from datetime import datetime, timedelta
# Assume data recording started at the date and time specified in the filename
base_time = datetime(2025, 5, 12, int(hour), int(minute)) # Date and time
# parsed from filename
time_labels = [base_time + timedelta(seconds=s) for s in relative_seconds]

# Determine the number of sensors in the dataset
```

```

num_sensors = len([col for col in data.columns if 'Sensor_' in col]) - 1

# Create a figure with subplots for all sensors
plt.figure(figsize=(15, 10))

# Plot data for all sensors
for i in range(1, num_sensors + 1):
    sensor_name = f'Sensor_{i}'
    plt.subplot(num_sensors, 1, i)
    plt.plot(time_labels, data[sensor_name], linewidth=1)
    plt.title(f'{sensor_name} Data')
    plt.ylabel(f'{sensor_name} Value')
    plt.grid(True)

    # Only add x-label for the bottom subplot
    if i == num_sensors:
        plt.xlabel('Time')

plt.gcf().autofmt_xdate() # Automatically format x-axis date labels

# Add a main title for the entire figure
plt.suptitle(f'Sensor Data - {date_str} {formatted_time}', fontsize=16)

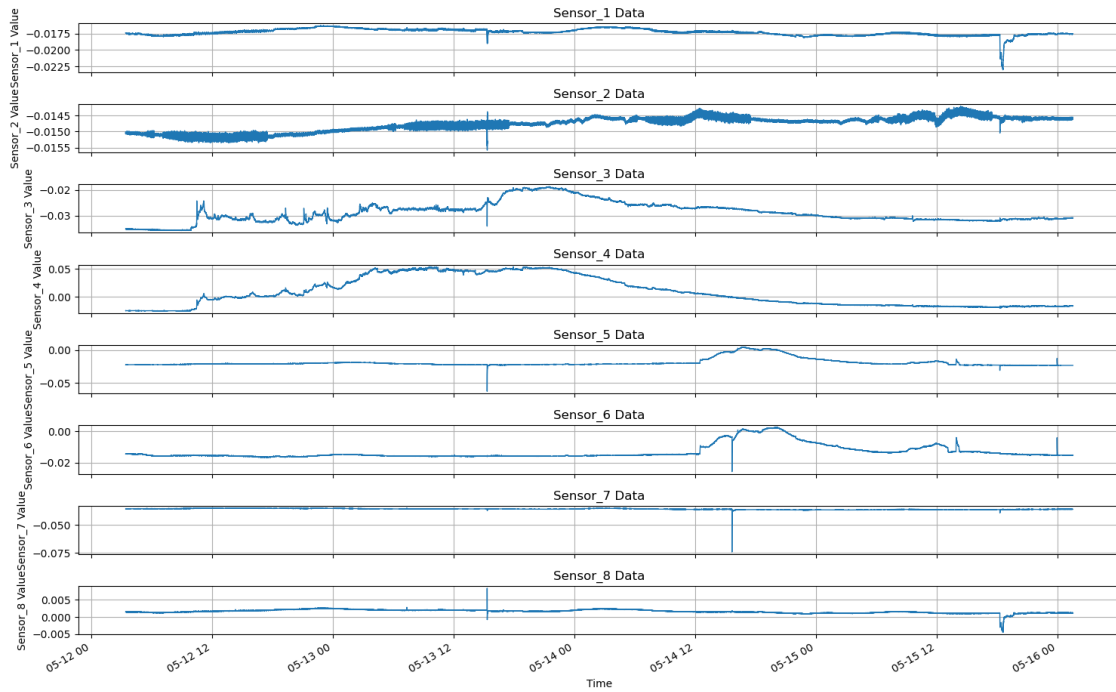
# Adjust layout
plt.tight_layout(rect=[0, 0, 1, 0.97]) # Make room for the supitle

# Display the figure
plt.show()

# Print basic statistics for all sensors
print("Sensor Statistics:")
for i in range(1, num_sensors):
    sensor_name = f'Sensor_{i}'
    print(f"\n{sensor_name}:\n{data[sensor_name].describe()}")

```

Sensor Data - 25/05/08 03:26



## Sensor Statistics:

### Sensor\_1:

```
count    1.084420e+06
mean     -1.727884e-02
std       5.048720e-04
min      -2.300700e-02
25%      -1.762500e-02
50%      -1.728600e-02
75%      -1.690600e-02
max       -1.632500e-02
Name: Sensor_1, dtype: float64
```

### Sensor\_2:

```
count    1.084420e+06
mean     -1.481143e-02
std       2.283351e-04
min      -1.557400e-02
25%      -1.504800e-02
50%      -1.476700e-02
75%      -1.461900e-02
max       -1.421000e-02
Name: Sensor_2, dtype: float64
```

```
Sensor_3:
count      1.084420e+06
mean       -2.902471e-02
std         4.014688e-03
min        -3.576500e-02
25%        -3.148200e-02
50%        -2.978400e-02
75%        -2.700600e-02
max        -1.869200e-02
Name: Sensor_3, dtype: float64
```

```
Sensor_4:
count      1.084420e+06
mean        9.146397e-03
std         2.632990e-02
min        -2.506200e-02
25%        -1.520400e-02
50%         2.054000e-03
75%         3.857700e-02
max         5.387100e-02
Name: Sensor_4, dtype: float64
```

```
Sensor_5:
count      1.084420e+06
mean       -1.927404e-02
std         5.577440e-03
min        -6.273700e-02
25%        -2.202700e-02
50%        -2.094700e-02
75%        -1.956500e-02
max         4.554000e-03
Name: Sensor_5, dtype: float64
```

```
Sensor_6:
count      1.084420e+06
mean       -1.349133e-02
std         4.149826e-03
min        -2.560900e-02
25%        -1.566500e-02
50%        -1.521000e-02
75%        -1.375100e-02
max         2.503000e-03
Name: Sensor_6, dtype: float64
```

```
Sensor_7:
count      1.084420e+06
mean       -3.571238e-02
std         5.212150e-04
```

```
min      -7.406800e-02
25%      -3.605700e-02
50%      -3.561300e-02
75%      -3.531600e-02
max      -3.472400e-02
Name: Sensor_7, dtype: float64
```

```
[3]: # Perform Short-Time Fourier Transform (STFT) analysis
from scipy import signal
import matplotlib.pyplot as plt
import numpy as np

# Create a new figure for STFT analysis
plt.figure(figsize=(15, 10))

# Perform STFT on all sensor data
for i in range(1, 9): # Assuming 8 sensors
    sensor_name = f'Sensor_{i}'

    # Get sensor data
    sensor_data = data[sensor_name].values

    # Calculate sampling rate (based on timestamp differences)
    sampling_rate = 1.0 / np.mean(np.diff(data['Timestamp']))

    # Perform STFT
    f, t, Zxx = signal.stft(sensor_data, fs=sampling_rate, nperseg=256)

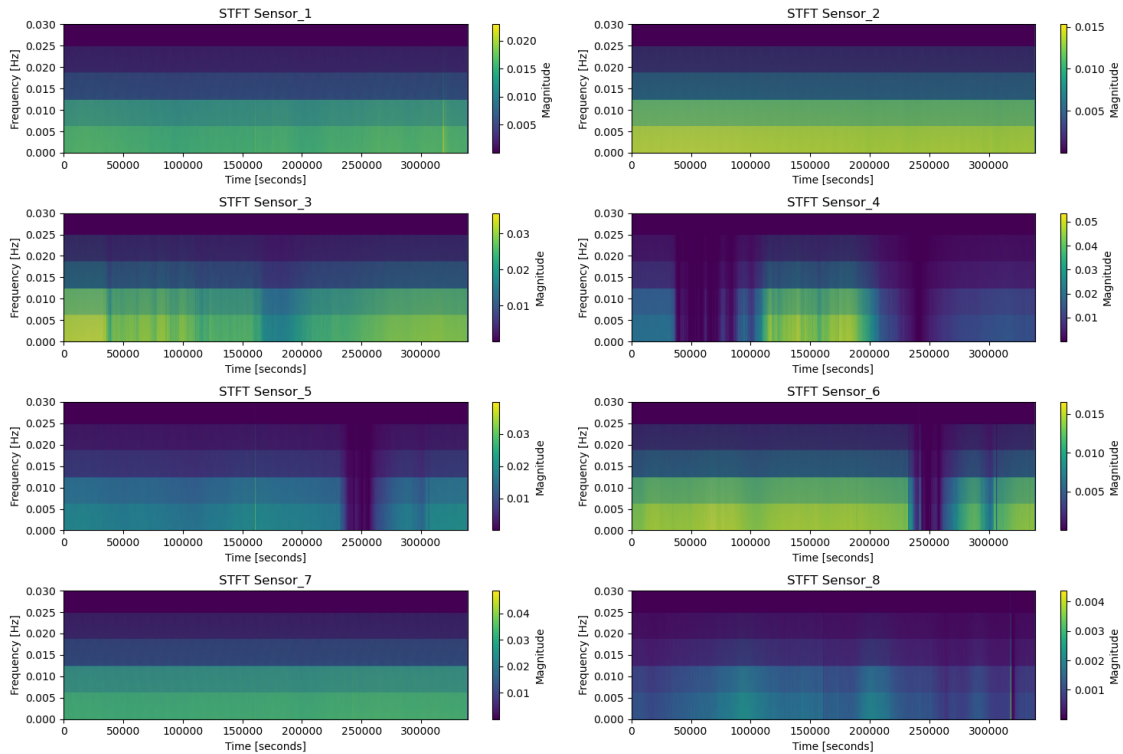
    # Plot STFT results
    plt.subplot(4, 2, i)

    plt.pcolormesh(t, f, np.abs(Zxx), shading='gouraud')

    plt.title(f'STFT {sensor_name}')
    plt.ylabel('Frequency [Hz]')
    plt.xlabel('Time [seconds]')
    plt.colorbar(label='Magnitude')
    plt.ylim(0, 0.03) # Limit y-axis to 0.03Hz

plt.tight_layout()
plt.show()

# Print basic information about the STFT analysis
print(f"STFT analysis completed")
print(f"Sampling rate: {sampling_rate:.2f} Hz")
print(f"Frequency resolution: {f[1]-f[0]:.4f} Hz")
print(f"Time resolution: {t[1]-t[0]:.4f} seconds")
```



STFT analysis completed  
Sampling rate: 3.20 Hz  
Frequency resolution: 0.0125 Hz  
Time resolution: 39.9933 seconds

```
[4]: # Calculate the recording end time based on the timestamp
import datetime
# Extract start time from the filename (Mushroom_25-05-08_0326)
filename = file_path.split('/')[-1]
date_part = filename.split('_')[1] # '25-05-08'
time_part = filename.split('_')[2] # '0326'

# Handle potential file extension in time_part
if '.' in time_part:
    time_part = time_part.split('.')[0] # Remove file extension if present

year = 2000 + int(date_part.split('-')[0]) # '25' -> 2025
month = int(date_part.split('-')[1]) # '05' -> 5
day = int(date_part.split('-')[2]) # '08' -> 8
hour = int(time_part[:2]) # '03' -> 3
minute = int(time_part[2:]) # '26' -> 26

start_time = datetime.datetime(year, month, day, hour, minute)
```

```

# Get the first and last timestamp
first_timestamp = data['Timestamp'].iloc[0]
last_timestamp = data['Timestamp'].iloc[-1]

# Calculate the duration in seconds
duration_seconds = last_timestamp - first_timestamp

# Calculate the end time
end_time = start_time + datetime.timedelta(seconds=duration_seconds)

# Format and print the results
print(f"Recording start time: {start_time.strftime('%Y-%m-%d %H:%M:%S')}")
print(f"Recording end time: {end_time.strftime('%Y-%m-%d %H:%M:%S')}")
print(f"Total recording duration: {duration_seconds:.2f} seconds_
↳({duration_seconds/60:.2f} minutes)")

```

Recording start time: 2025-05-08 03:26:00  
Recording end time: 2025-05-12 01:33:04  
Total recording duration: 338824.48 seconds (5647.07 minutes)

```

[5]: # Function to find the closest timestamp in the data to a given event time
import pytz
import datetime

# Parse the event time string
event_time_str = "2025-05-11T19:58:58.093Z"
event_time = datetime.datetime.strptime(event_time_str, "%Y-%m-%dT%H:%M:%S.%fZ")
event_time = event_time.replace(tzinfo=pytz.UTC) # Make it timezone-aware

# Make start_time timezone-aware as well
start_time = start_time.replace(tzinfo=pytz.UTC)

# Calculate seconds elapsed since recording start
elapsed_seconds = (event_time - start_time).total_seconds()

print(f"Event time: {event_time_str}")
print(f"Recording start time: {start_time.strftime('%Y-%m-%d %H:%M:%S %Z')}")
print(f"Seconds elapsed since recording start: {elapsed_seconds:.2f} seconds")

# Get the first timestamp from the data
first_timestamp = data['Timestamp'].iloc[0]

# Calculate the target timestamp by adding elapsed seconds to the first_
↳timestamp
target_timestamp = first_timestamp + elapsed_seconds

```



```

# Find the closest timestamp in the data
closest_idx = (data['Timestamp'] - target_timestamp).abs().idxmin()
closest_timestamp = data['Timestamp'].iloc[closest_idx]
closest_time_diff = abs(closest_timestamp - target_timestamp)

print(f"First data timestamp: {first_timestamp:.2f} seconds")
print(f"Target timestamp: {target_timestamp:.2f} seconds")
print(f"Closest data timestamp: {closest_timestamp:.2f} seconds")
print(f"Difference from target: {closest_time_diff:.2f} seconds")

# Extract the data at the closest timestamp
event_data = data.iloc[closest_idx]
print("\nSensor readings at event time:")
for column in data.columns:
    if column != 'Timestamp':
        print(f"{column}: {event_data[column]}")

```

Event time: 2025-05-11T19:58:58.093Z  
 Recording start time: 2025-05-08 03:26:00 UTC  
 Seconds elapsed since recording start: 318778.09 seconds  
 First data timestamp: 120386.54 seconds  
 Target timestamp: 439164.63 seconds  
 Closest data timestamp: 439164.77 seconds  
 Difference from target: 0.14 seconds

Sensor readings at event time:

Sensor_1:	-0.017922
Sensor_2:	-0.01462
Sensor_3:	-0.03123
Sensor_4:	-0.01655
Sensor_5:	-0.023128
Sensor_6:	-0.014577
Sensor_7:	-0.036016
Sensor_8:	0.000943
Sensor_9:	nan

```

[6]: # Plot voltage data for 10 minutes before and after the event time
import matplotlib.pyplot as plt
import numpy as np

# Define the time window (10 minutes before and after the event)
window_minutes = 10
window_seconds = window_minutes * 60 # Convert minutes to seconds
event_idx = closest_idx
start_idx = max(0, event_idx - int(window_seconds * data['Timestamp'].diff().
    ↪median() ** -1))

```

```

end_idx = min(len(data) - 1, event_idx + int(window_seconds * data['Timestamp'].
↳diff().median() ** -1))

# Extract the data for the time window
window_data = data.iloc[start_idx:end_idx+1]

# Calculate time relative to the event (in seconds)
relative_time = window_data['Timestamp'] - closest_timestamp

# Create a figure with subplots for each voltage channel
plt.figure(figsize=(15, 10))
voltage_columns = [col for col in data.columns if col != 'Timestamp']

for i, column in enumerate(voltage_columns):
    plt.subplot(3, 3, i+1)

    # Convert voltage to millivolts
    voltage_mv = window_data[column] * 1000 # Convert to mV

    plt.plot(relative_time, voltage_mv)
    plt.axvline(x=0, color='r', linestyle='--', label='Event time')
    plt.title(f'{column} (mV)')
    plt.xlabel('Time relative to event (seconds)')
    plt.ylabel('Voltage (mV)')
    plt.grid(True)

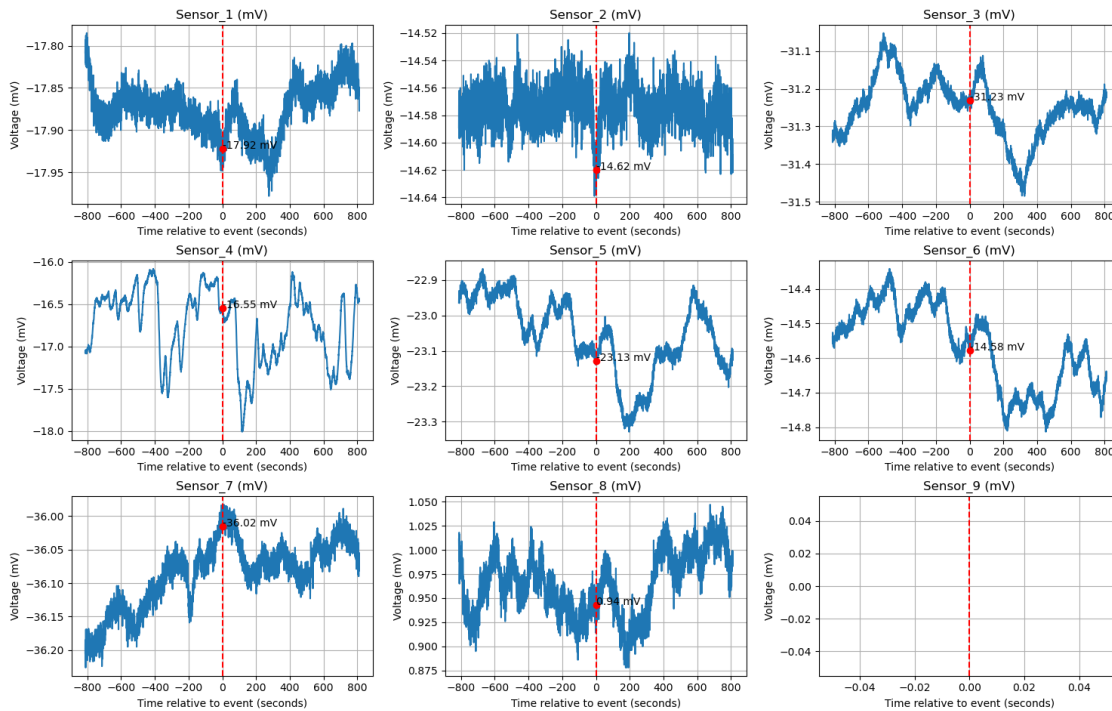
    # Add a red dot at the event time point
    event_value_mv = event_data[column] * 1000 # Convert to mV
    plt.plot(0, event_value_mv, 'ro', markersize=6) # Red dot at event time
    plt.text(1, event_value_mv, f'{event_value_mv:.2f} mV') # Text label
↳without arrow

plt.tight_layout()
plt.suptitle('Voltage Readings ±10 minutes from Event Time', fontsize=16)
plt.subplots_adjust(top=0.92)
plt.show()

```

posx and posy should be finite values  
posx and posy should be finite values

Voltage Readings  $\pm 10$  minutes from Event Time



```
[7]: # Perform Short-Time Fourier Transform (STFT) analysis for each voltage channel
import matplotlib.pyplot as plt
from scipy import signal
import numpy as np

# Create a figure with subplots for STFT of each voltage channel
plt.figure(figsize=(15, 10))
voltage_columns = [col for col in data.columns if col != 'Timestamp']

# Calculate sampling frequency
sampling_freq = 1.0 / data['Timestamp'].diff().median()

for i, column in enumerate(voltage_columns):
    plt.subplot(3, 3, i+1)

    # Get voltage data for this channel
    voltage_data = window_data[column].values

    # Perform STFT
    f, t, Zxx = signal.stft(voltage_data, fs=sampling_freq, nperseg=256)

    # Plot the STFT magnitude (in dB)
    plt.pcolormesh(t, f, np.abs(Zxx), shading='gouraud')
```

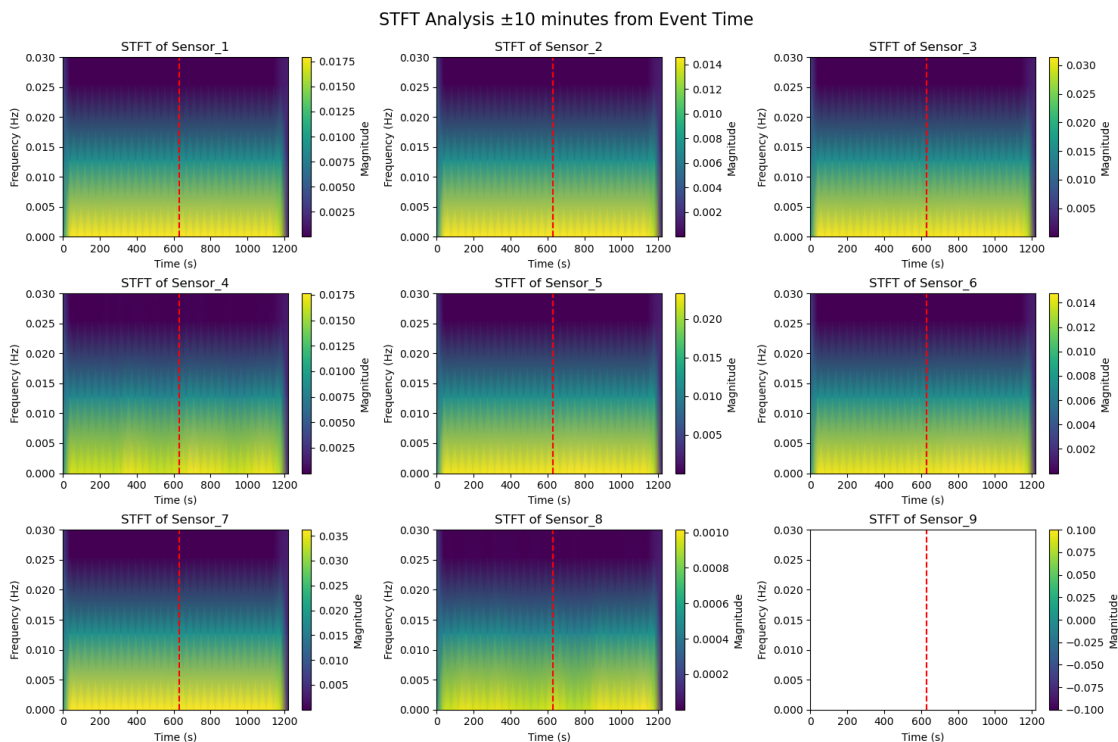
```

# Mark the event time
plt.axvline(x=t[len(t)//2], color='r', linestyle='--', label='Event time')

plt.title(f'STFT of {column}')
plt.ylabel('Frequency (Hz)')
plt.xlabel('Time (s)')
plt.colorbar(label='Magnitude')
plt.ylim(0, 0.03)

plt.tight_layout()
plt.suptitle('STFT Analysis ±10 minutes from Event Time', fontsize=16)
plt.subplots_adjust(top=0.92)
plt.show()

```



```

[8]: # Analyze the 0.02Hz frequency band before and after event for each sensor
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import datetime

# Get dataset name from the notebook filename

```

```

notebook_name = os.path.basename(__file__) if '__file__' in globals() else_
↳ 'Mushroom_25-05-08_0326'
if notebook_name.endswith('.ipynb'):
    notebook_name = notebook_name[:-6] # Remove .ipynb extension
if notebook_name.startswith('da_'):
    notebook_name = notebook_name[3:] # Remove da_ prefix

# Create a directory to save CSV files with dataset name
csv_dir = f"significant_changes_csv_{notebook_name}"
if not os.path.exists(csv_dir):
    os.makedirs(csv_dir)
    print(f"Created directory: {csv_dir}")

# Calculate sampling frequency
sampling_freq = 1.0 / data['Timestamp'].diff().median()

# Find the event time (assuming it's at the center of the filtered data)
event_time = window_data['Timestamp'].mean()

# Loop through each voltage channel
for channel_to_analyze in voltage_columns:
    print(f"\n=== Analysis for {channel_to_analyze} ===")
    voltage_data = window_data[channel_to_analyze].values

    # Perform STFT for the selected channel
    f, t, Zxx = signal.stft(voltage_data, fs=sampling_freq, nperseg=256)

    # Find the closest frequency to 0.02Hz in the STFT results
    target_freq = 0.02
    freq_idx = np.argmin(np.abs(f - target_freq))
    actual_freq = f[freq_idx]
    print(f"Analyzing frequency: {actual_freq:.4f} Hz (closest to 0.02 Hz)")

    # Extract the magnitude data for this frequency
    freq_magnitude = np.abs(Zxx[freq_idx, :])

    # Create a time axis in minutes for better visualization
    time_min = t / 60

    # Plot the magnitude of the 0.02Hz component over time
    plt.figure(figsize=(15, 6))

    # Plot the magnitude
    plt.plot(time_min, freq_magnitude, 'b-', linewidth=2, label=f'{actual_freq:.
↳ 4f} Hz Component')

    # Convert event time to minutes

```

```

event_time_min = t.mean() / 60
plt.axvline(x=event_time_min, color='r', linestyle='--', label='Event Time_
↳(estimated)')

# Calculate average magnitude before and after event
before_mask = t < t.mean()
after_mask = t >= t.mean()

avg_before = np.mean(freq_magnitude[before_mask])
avg_after = np.mean(freq_magnitude[after_mask])

print(f"Average magnitude before event: {avg_before:.4f}")
print(f"Average magnitude after event: {avg_after:.4f}")
print(f"Change: {(avg_after - avg_before):.4f} ({(avg_after - avg_before)/
↳avg_before*100:.2f}%)")

# Add horizontal lines showing the average values
plt.axhline(y=avg_before, color='g', linestyle=':', label=f'Avg Before:
↳{avg_before:.4f}')
plt.axhline(y=avg_after, color='m', linestyle=':', label=f'Avg After:
↳{avg_after:.4f}')

# Add annotations
plt.annotate(f"Avg: {avg_before:.4f}", xy=(time_min[len(time_min)//4],
↳avg_before),
            xytext=(time_min[len(time_min)//4], avg_before*1.1), color='g')
plt.annotate(f"Avg: {avg_after:.4f}", xy=(time_min[3*len(time_min)//4],
↳avg_after),
            xytext=(time_min[3*len(time_min)//4], avg_after*1.1),
↳color='m')

# Set axis labels and title
plt.xlabel('Time (min)')
plt.ylabel('Magnitude')
plt.title(f'Magnitude of {actual_freq:.4f} Hz Component Before and After_
↳Event - {channel_to_analyze}')
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()

# Calculate energy (integral of magnitude squared) before and after event
energy_before = np.sum(freq_magnitude[before_mask]**2)
energy_after = np.sum(freq_magnitude[after_mask]**2)

# Normalize by the number of samples to get average energy

```

```

num_samples_before = np.sum(before_mask)
num_samples_after = np.sum(after_mask)
avg_energy_before = energy_before / num_samples_before
avg_energy_after = energy_after / num_samples_after

print("\nEnergy Analysis:")
print(f"Total energy before event: {energy_before:.4f}")
print(f"Total energy after event: {energy_after:.4f}")
print(f"Average energy before event: {avg_energy_before:.4f}")
print(f"Average energy after event: {avg_energy_after:.4f}")
print(f"Energy change: {(avg_energy_after - avg_energy_before):.4f}␣
↪({(avg_energy_after - avg_energy_before)/avg_energy_before*100:.2f}%)")

# Power Spectral Density (PSD) Analysis
# Calculate power (magnitude squared)
power_matrix = np.abs(Zxx) ** 2

# Convert time to minutes for consistency with previous plots
time_min = t / 60

# Define the event time point (assuming same as before)
event_time_min = time_min[len(time_min) // 2] # Middle point as event time

# Create masks for before and after event
before_mask_time = time_min < event_time_min
after_mask_time = time_min > event_time_min

# Calculate average PSD before and after event
avg_psd_before = np.mean(power_matrix[:, before_mask_time], axis=1)
avg_psd_after = np.mean(power_matrix[:, after_mask_time], axis=1)

# Plot the power spectral density comparison
plt.figure(figsize=(15, 6))
plt.plot(f, avg_psd_before, 'g-', label='Before Event')
plt.plot(f, avg_psd_after, 'm-', label='After Event')

# Calculate and display the difference
psd_diff = avg_psd_after - avg_psd_before
plt.plot(f, psd_diff, 'b--', label='Difference (After - Before)')

# Set axis labels and title
plt.xlabel('Frequency (Hz)')
plt.xlim(0, 0.2) # Limit x-axis to show only frequencies below 0.2 Hz
plt.ylabel('Power Spectral Density')
plt.title(f'Power Spectral Density Comparison Before and After Event -␣
↪{channel_to_analyze}')
plt.grid(True)

```

```

plt.legend()

# Add text box with summary statistics
total_power_before = np.sum(avg_psd_before)
total_power_after = np.sum(avg_psd_after)
power_change = (total_power_after - total_power_before) /
total_power_before * 100

stats_text = f"Total Power Before: {total_power_before:.2f}\n"
stats_text += f"Total Power After: {total_power_after:.2f}\n"
stats_text += f"Change: {power_change:.2f}%"

plt.annotate(stats_text, xy=(0.02, 0.95), xycoords='axes fraction',
             bbox=dict(boxstyle="round,pad=0.5", fc="white", alpha=0.8))

plt.tight_layout()
plt.show()

# Print detailed statistics
print("\nPower Spectral Density Analysis:")
print(f"Total power before event: {total_power_before:.4f}")
print(f"Total power after event: {total_power_after:.4f}")
print(f"Absolute power change: {total_power_after - total_power_before:.4f}")
print(f"Relative power change: {power_change:.2f}%")

# Find frequency bands with the most significant changes
freq_change_percent = (avg_psd_after - avg_psd_before) / (avg_psd_before +
1e-10) * 100 # Avoid division by zero
significant_changes = pd.DataFrame({
    'Frequency': f,
    'Before': avg_psd_before,
    'After': avg_psd_after,
    'Absolute_Change': avg_psd_after - avg_psd_before,
    'Percent_Change': freq_change_percent
})

# Save the significant_changes DataFrame to CSV
csv_filename = os.path.join(csv_dir,
f"{channel_to_analyze}_significant_changes.csv")
significant_changes.to_csv(csv_filename, index=False)
print(f"Saved significant changes data to: {csv_filename}")

# Display top 5 frequencies with largest increase and decrease
print("\nTop 5 frequencies with largest power increase:")
print(significant_changes.sort_values('Percent_Change', ascending=False).
head(5))

```



```

print("\nTop 5 frequencies with largest power decrease:")
print(significant_changes.sort_values('Percent_Change', ascending=True).
↪head(5))

```

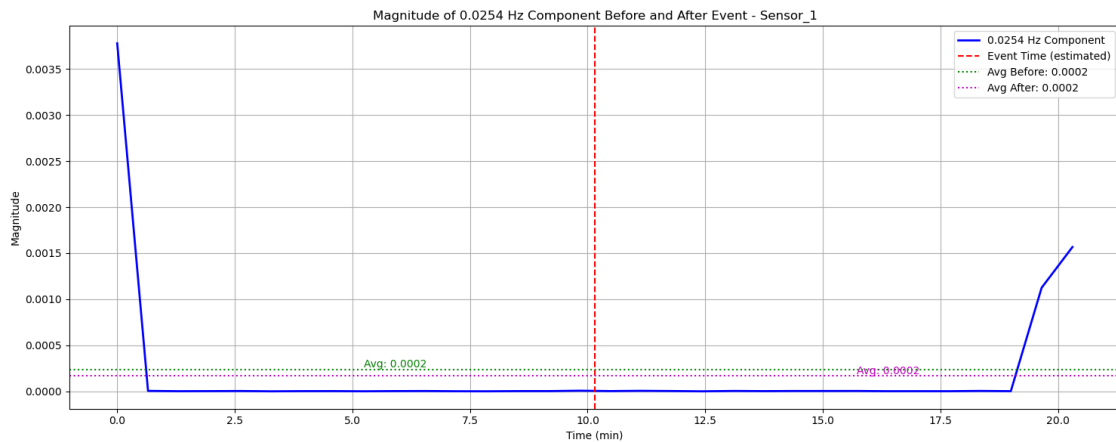
=== Analysis for Sensor\_1 ===

Analyzing frequency: 0.0254 Hz (closest to 0.02 Hz)

Average magnitude before event: 0.0002

Average magnitude after event: 0.0002

Change: -0.0001 (-28.40%)



Energy Analysis:

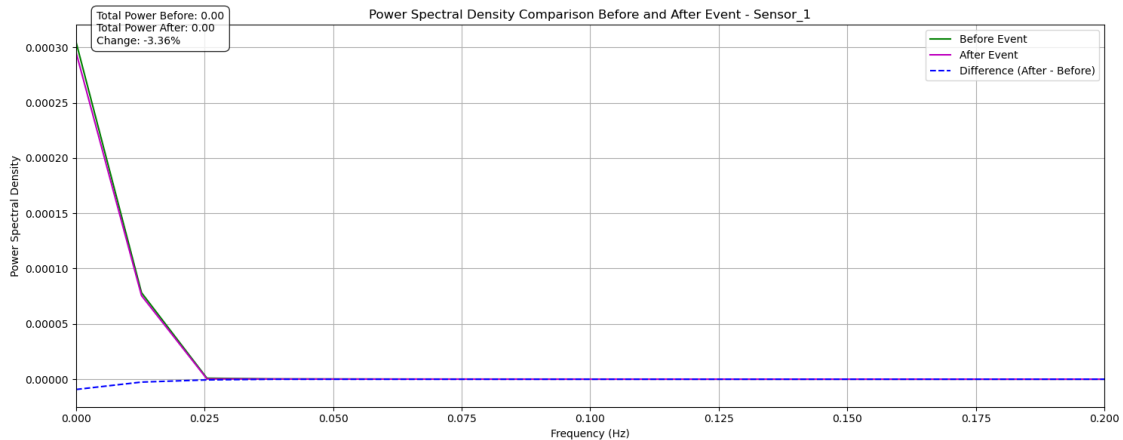
Total energy before event: 0.0000

Total energy after event: 0.0000

Average energy before event: 0.0000

Average energy after event: 0.0000

Energy change: -0.0000 (-73.95%)



# Power Spectral Density Analysis:

Total power before event: 0.0004

Total power after event: 0.0004

Absolute power change: -0.0000

Relative power change: -3.36%

Saved significant changes data to:

significant\_changes\_csv\_Mushroom\_25-05-08\_0326\Sensor\_1\_significant\_changes.csv

## Top 5 frequencies with largest power increase:

	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	3.047069e-04	2.954109e-04	-9.296053e-06	-3.050817
1	0.012722	7.821000e-05	7.558144e-05	-2.628554e-06	-3.360888
3	0.038165	2.229759e-07	1.516832e-07	-7.129262e-08	-31.958914
122	1.552063	3.027450e-10	1.713620e-10	-1.313830e-10	-32.621885
116	1.475732	3.046896e-10	1.722731e-10	-1.324164e-10	-32.720498

## Top 5 frequencies with largest power decrease:

	Frequency	Before	After	Absolute_Change	Percent_Change
2	0.025444	8.921114e-07	2.479339e-07	-6.441775e-07	-72.200099
6	0.076331	5.911832e-08	2.990685e-08	-2.921147e-08	-49.328432
10	0.127218	2.059596e-08	1.099720e-08	-9.598759e-09	-46.379876
14	0.178106	1.042795e-08	5.625005e-09	-4.802941e-09	-45.620878
12	0.152662	1.421280e-08	7.691818e-09	-6.520977e-09	-45.560473

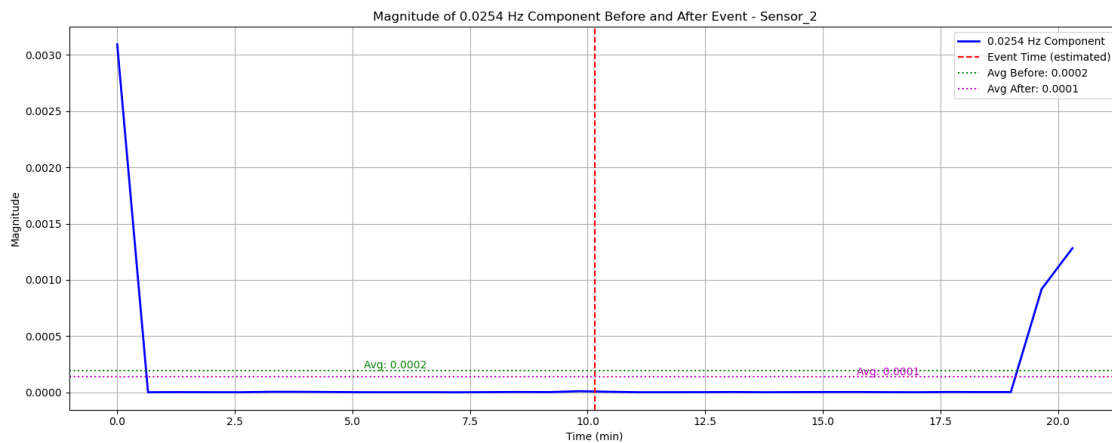
## === Analysis for Sensor\_2 ===

Analyzing frequency: 0.0254 Hz (closest to 0.02 Hz)

Average magnitude before event: 0.0002

Average magnitude after event: 0.0001

Change: -0.0001 (-28.79%)



### Energy Analysis:

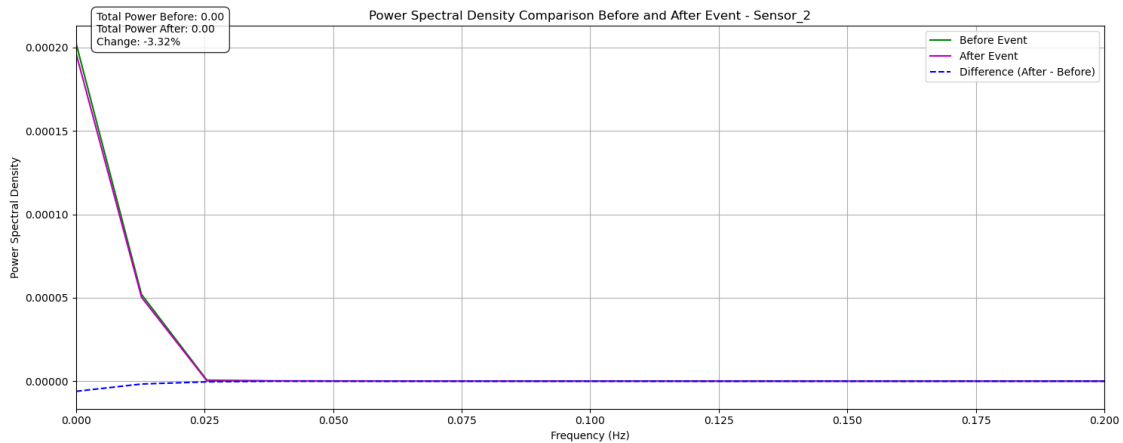
Total energy before event: 0.0000

Total energy after event: 0.0000

Average energy before event: 0.0000

Average energy after event: 0.0000

Energy change: -0.0000 (-74.04%)



### Power Spectral Density Analysis:

Total power before event: 0.0003

Total power after event: 0.0002

Absolute power change: -0.0000

Relative power change: -3.32%

Saved significant changes data to:

significant\_changes\_csv\_Mushroom\_25-05-08\_0326\Sensor\_2\_significant\_changes.csv

### Top 5 frequencies with largest power increase:

	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	2.024967e-04	1.963894e-04	-6.107242e-06	-3.015971
1	0.012722	5.198579e-05	5.025710e-05	-1.728695e-06	-3.325315
128	1.628394	2.025686e-10	1.148475e-10	-8.772110e-11	-28.992133
127	1.615672	2.010078e-10	1.126652e-10	-8.834260e-11	-29.348943
126	1.602950	2.009772e-10	1.125205e-10	-8.845666e-11	-29.389823

### Top 5 frequencies with largest power decrease:

	Frequency	Before	After	Absolute_Change	Percent_Change
2	0.025444	5.982449e-07	1.656795e-07	-4.325654e-07	-72.293654
6	0.076331	3.960058e-08	2.004504e-08	-1.955554e-08	-49.257565
10	0.127218	1.380407e-08	7.323316e-09	-6.480756e-09	-46.610490
12	0.152662	9.553894e-09	5.165403e-09	-4.388491e-09	-45.458248

8      0.101775    2.176005e-08    1.185166e-08      -9.908392e-09      -45.326475

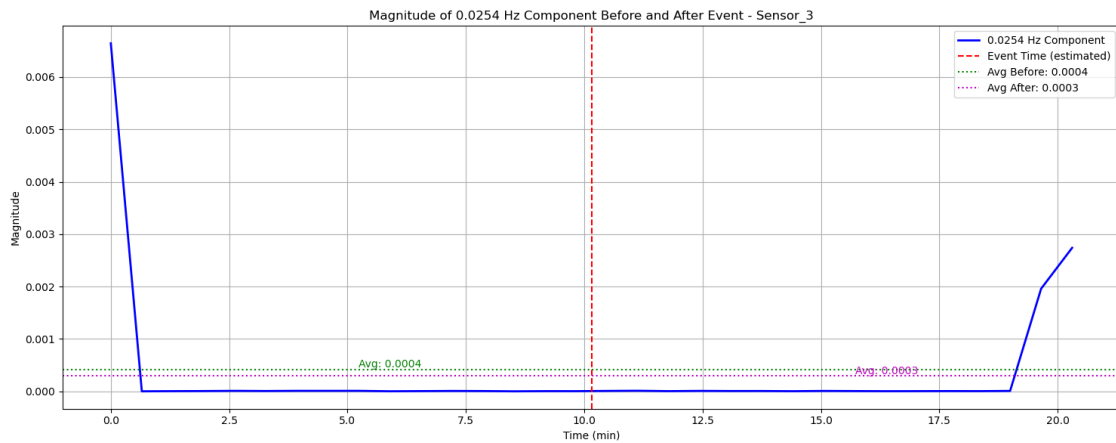
=== Analysis for Sensor\_3 ===

Analyzing frequency: 0.0254 Hz (closest to 0.02 Hz)

Average magnitude before event: 0.0004

Average magnitude after event: 0.0003

Change: -0.0001 (-28.86%)



Energy Analysis:

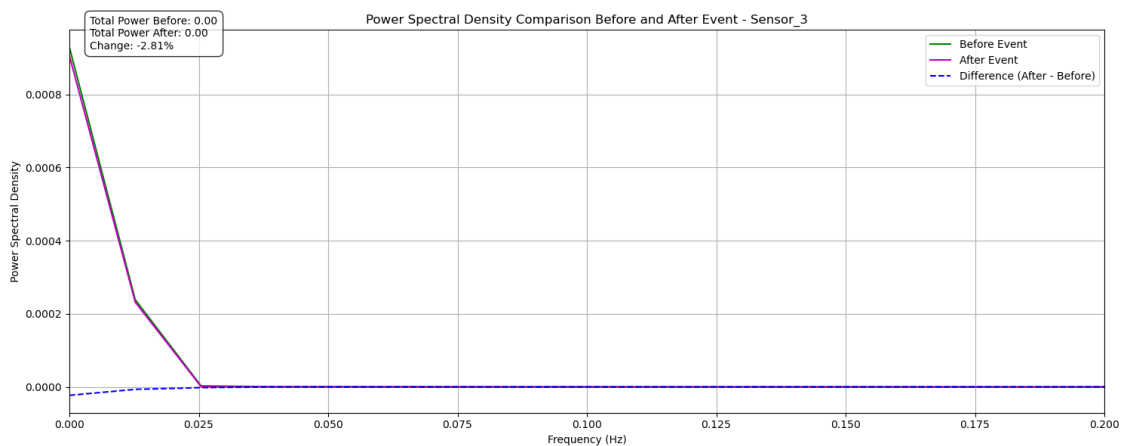
Total energy before event: 0.0000

Total energy after event: 0.0000

Average energy before event: 0.0000

Average energy after event: 0.0000

Energy change: -0.0000 (-74.29%)



# Power Spectral Density Analysis:

Total power before event: 0.0012

Total power after event: 0.0011

Absolute power change: -0.0000

Relative power change: -2.81%

Saved significant changes data to:

significant\_changes\_csv\_Mushroom\_25-05-08\_0326\Sensor\_3\_significant\_changes.csv

## Top 5 frequencies with largest power increase:

	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	9.286842e-04	9.056150e-04	-2.306919e-05	-2.484072
1	0.012722	2.384626e-04	2.317005e-04	-6.762058e-06	-2.835689
3	0.038165	6.903659e-07	4.632254e-07	-2.271404e-07	-32.896691
127	1.615672	9.227440e-10	5.051654e-10	-4.175786e-10	-40.829237
128	1.628394	9.213738e-10	5.023326e-10	-4.190412e-10	-41.027213

## Top 5 frequencies with largest power decrease:

	Frequency	Before	After	Absolute_Change	Percent_Change
2	0.025444	2.759146e-06	7.567207e-07	-2.002426e-06	-72.571470
6	0.076331	1.830036e-07	9.145088e-08	-9.155268e-08	-50.000491
10	0.127218	6.360749e-08	3.348689e-08	-3.012060e-08	-47.279533
12	0.152662	4.406247e-08	2.350881e-08	-2.055366e-08	-46.541015
14	0.178106	3.235110e-08	1.725819e-08	-1.509291e-08	-46.509697

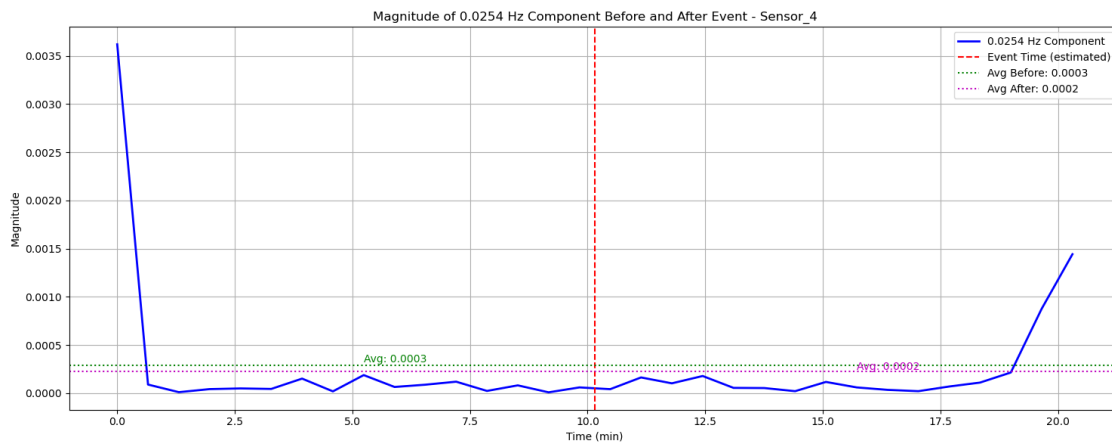
## === Analysis for Sensor\_4 ===

Analyzing frequency: 0.0254 Hz (closest to 0.02 Hz)

Average magnitude before event: 0.0003

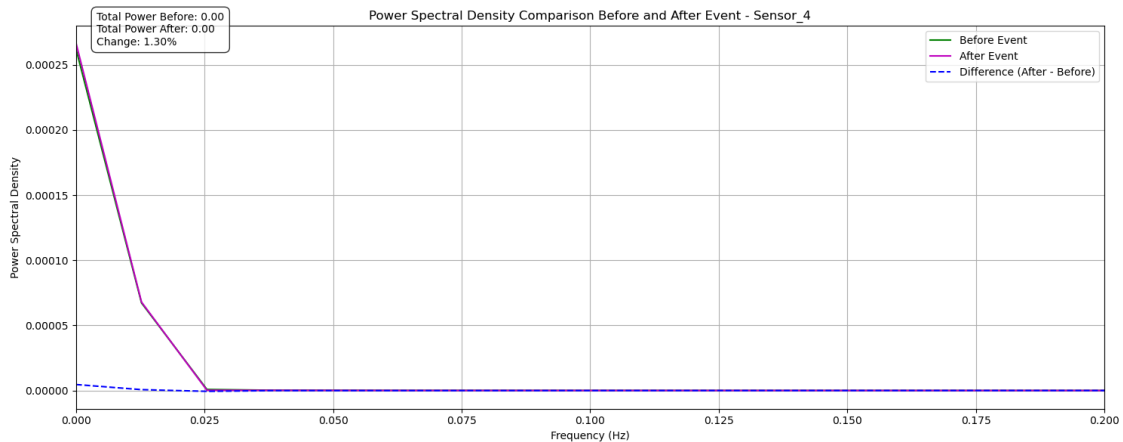
Average magnitude after event: 0.0002

Change: -0.0001 (-23.59%)



## Energy Analysis:

Total energy before event: 0.0000  
Total energy after event: 0.0000  
Average energy before event: 0.0000  
Average energy after event: 0.0000  
Energy change: -0.0000 (-77.24%)



#### Power Spectral Density Analysis:

Total power before event: 0.0003  
Total power after event: 0.0003  
Absolute power change: 0.0000  
Relative power change: 1.30%  
Saved significant changes data to:  
significant\_changes\_csv\_Mushroom\_25-05-08\_0326\Sensor\_4\_significant\_changes.csv

#### Top 5 frequencies with largest power increase:

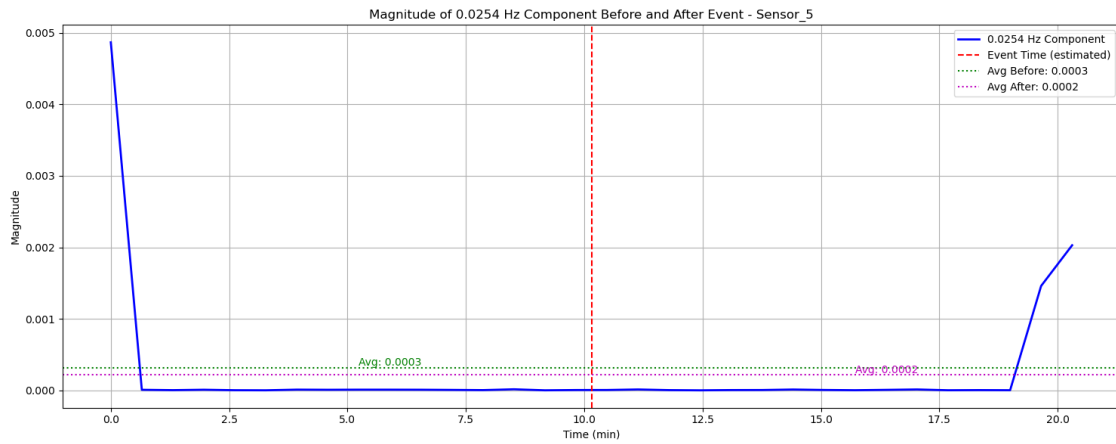
	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	2.618330e-04	2.664266e-04	4.593563e-06	1.754386
1	0.012722	6.722234e-05	6.789363e-05	6.712902e-07	0.998610
115	1.463010	2.808188e-10	1.467664e-10	-1.340524e-10	-35.201093
110	1.399401	2.884133e-10	1.513526e-10	-1.370608e-10	-35.287349
3	0.038165	2.065714e-07	1.335890e-07	-7.298245e-08	-35.313277

#### Top 5 frequencies with largest power decrease:

	Frequency	Before	After	Absolute_Change	Percent_Change
2	0.025444	8.260382e-07	2.004416e-07	-6.255966e-07	-75.725413
6	0.076331	5.410215e-08	2.575431e-08	-2.834785e-08	-52.300222
10	0.127218	1.881620e-08	9.311709e-09	-9.504494e-09	-50.245252
12	0.152662	1.304903e-08	6.484211e-09	-6.564818e-09	-49.926256
8	0.101775	2.966211e-08	1.487846e-08	-1.478365e-08	-49.672708

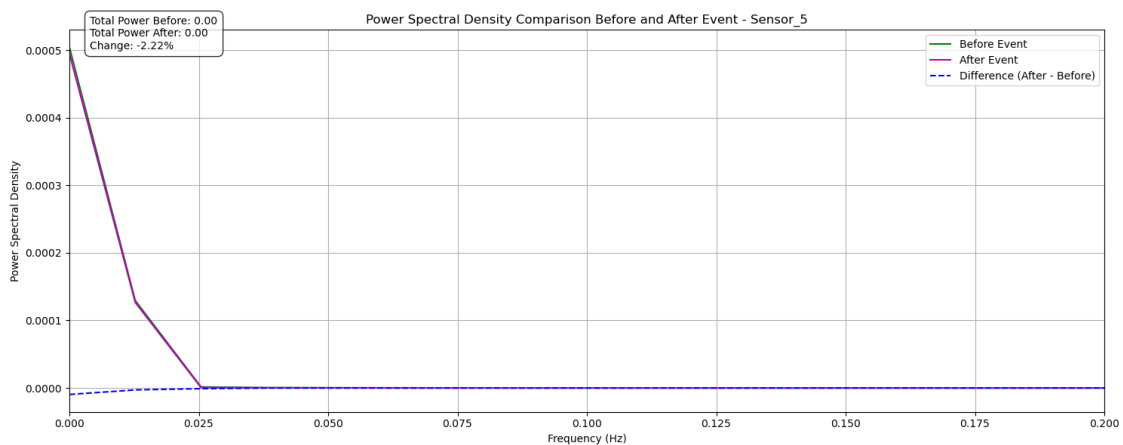
=== Analysis for Sensor\_5 ===

Analyzing frequency: 0.0254 Hz (closest to 0.02 Hz)  
Average magnitude before event: 0.0003  
Average magnitude after event: 0.0002  
Change: -0.0001 (-28.05%)



#### Energy Analysis:

Total energy before event: 0.0000  
Total energy after event: 0.0000  
Average energy before event: 0.0000  
Average energy after event: 0.0000  
Energy change: -0.0000 (-73.58%)



#### Power Spectral Density Analysis:

Total power before event: 0.0006  
Total power after event: 0.0006

Absolute power change: -0.0000

Relative power change: -2.22%

Saved significant changes data to:

significant\_changes\_csv\_Mushroom\_25-05-08\_0326\Sensor\_5\_significant\_changes.csv

Top 5 frequencies with largest power increase:

	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	5.040846e-04	4.944800e-04	-9.604594e-06	-1.905353
1	0.012722	1.293943e-04	1.265233e-04	-2.870981e-06	-2.218783
3	0.038165	3.706181e-07	2.537977e-07	-1.168204e-07	-31.511921
118	1.501176	5.029494e-10	2.856506e-10	-2.172988e-10	-36.039305
125	1.590228	5.003684e-10	2.823404e-10	-2.180280e-10	-36.315704

Top 5 frequencies with largest power decrease:

	Frequency	Before	After	Absolute_Change	Percent_Change
2	0.025444	1.479595e-06	4.169495e-07	-1.062645e-06	-71.815161
6	0.076331	9.816653e-08	5.018312e-08	-4.798341e-08	-48.829860
10	0.127218	3.425337e-08	1.836427e-08	-1.588910e-08	-46.251945
14	0.178106	1.733904e-08	9.458922e-09	-7.880114e-09	-45.186638
12	0.152662	2.366965e-08	1.293174e-08	-1.073791e-08	-45.174874

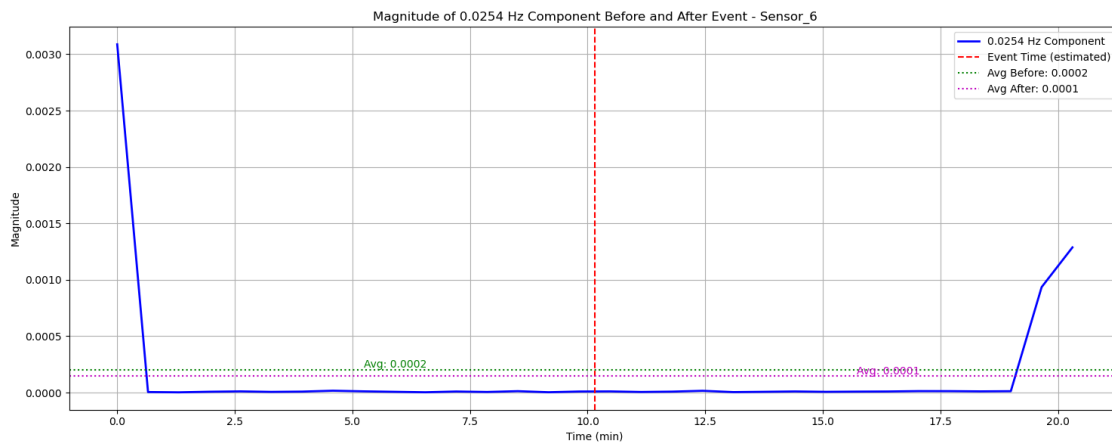
=== Analysis for Sensor\_6 ===

Analyzing frequency: 0.0254 Hz (closest to 0.02 Hz)

Average magnitude before event: 0.0002

Average magnitude after event: 0.0001

Change: -0.0001 (-26.46%)



Energy Analysis:

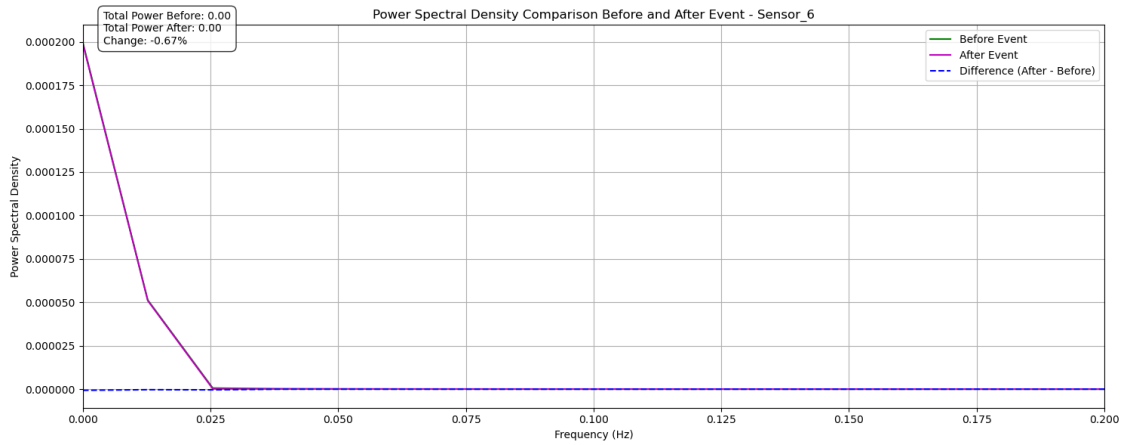
Total energy before event: 0.0000

Total energy after event: 0.0000

Average energy before event: 0.0000



Average energy after event: 0.0000  
Energy change: -0.0000 (-73.44%)



#### Power Spectral Density Analysis:

Total power before event: 0.0003

Total power after event: 0.0003

Absolute power change: -0.0000

Relative power change: -0.67%

Saved significant changes data to:

significant\_changes\_csv\_Mushroom\_25-05-08\_0326\Sensor\_6\_significant\_changes.csv

#### Top 5 frequencies with largest power increase:

	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	1.997078e-04	1.990024e-04	-7.054530e-07	-0.353242
1	0.012722	5.128023e-05	5.093843e-05	-3.417957e-07	-0.666524
126	1.602950	1.966888e-10	1.125427e-10	-8.414607e-11	-28.361733
127	1.615672	1.985400e-10	1.134824e-10	-8.505766e-11	-28.491208
110	1.399401	2.099837e-10	1.207498e-10	-8.923387e-11	-28.786632

#### Top 5 frequencies with largest power decrease:

	Frequency	Before	After	Absolute_Change	Percent_Change
2	0.025444	5.961087e-07	1.688599e-07	-4.272488e-07	-71.660944
6	0.076331	3.941976e-08	2.014135e-08	-1.927841e-08	-48.781693
10	0.127218	1.369872e-08	7.390908e-09	-6.307813e-09	-45.713029
12	0.152662	9.466488e-09	5.173252e-09	-4.293236e-09	-44.877869
8	0.101775	2.165274e-08	1.189777e-08	-9.754964e-09	-44.844764

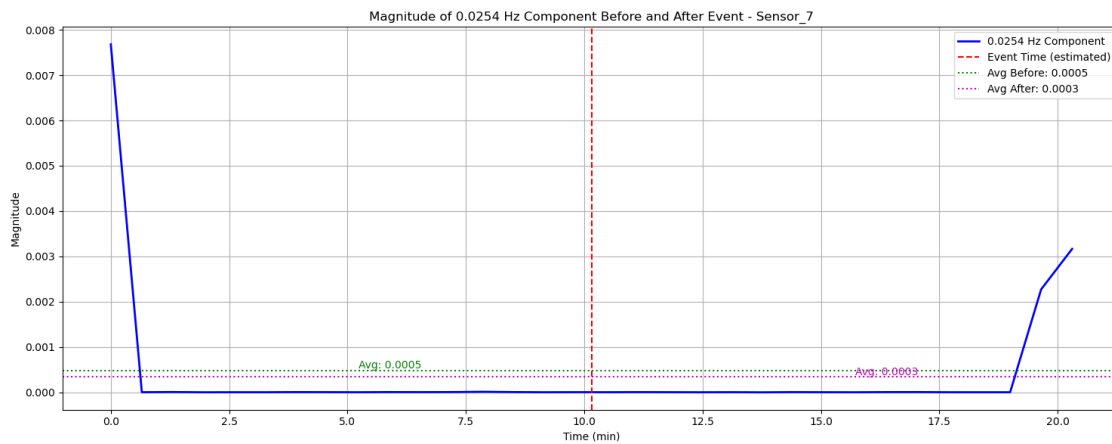
=== Analysis for Sensor\_7 ===

Analyzing frequency: 0.0254 Hz (closest to 0.02 Hz)

Average magnitude before event: 0.0005

Average magnitude after event: 0.0003

Change: -0.0001 (-29.27%)



#### Energy Analysis:

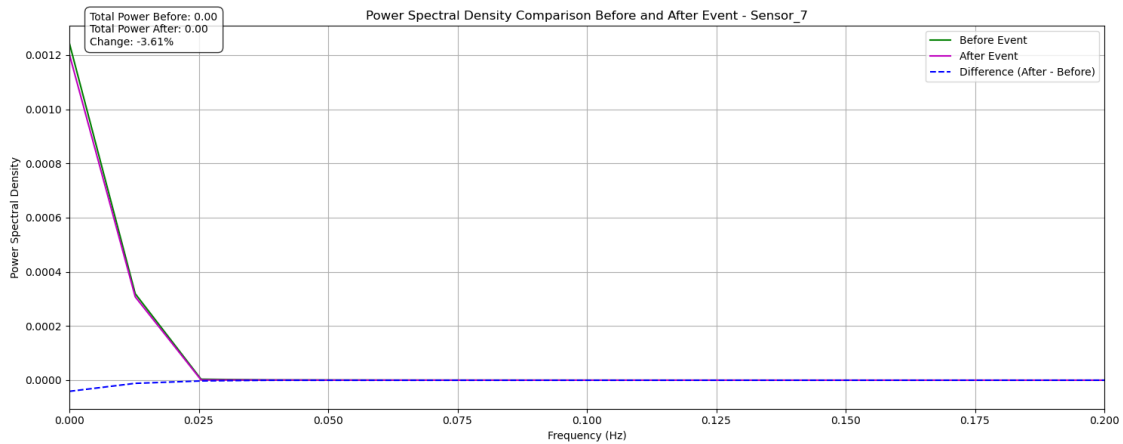
Total energy before event: 0.0001

Total energy after event: 0.0000

Average energy before event: 0.0000

Average energy after event: 0.0000

Energy change: -0.0000 (-74.26%)



#### Power Spectral Density Analysis:

Total power before event: 0.0016

Total power after event: 0.0015

Absolute power change: -0.0001

Relative power change: -3.61%

Saved significant changes data to:

significant\_changes\_csv\_Mushroom\_25-05-08\_0326\Sensor\_7\_significant\_changes.csv

Top 5 frequencies with largest power increase:

	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	1.243343e-03	1.202299e-03	-4.104409e-05	-3.301106
1	0.012722	3.191989e-04	3.076523e-04	-1.154657e-05	-3.617358
3	0.038165	9.216992e-07	6.193278e-07	-3.023714e-07	-32.802310
117	1.488454	1.261281e-09	6.901630e-10	-5.711183e-10	-41.954467
107	1.361236	1.329892e-09	7.244815e-10	-6.054101e-10	-42.339582

Top 5 frequencies with largest power decrease:

	Frequency	Before	After	Absolute_Change	Percent_Change
2	0.025444	3.685544e-06	1.011846e-06	-2.673698e-06	-72.543566
6	0.076331	2.441684e-07	1.220331e-07	-1.221353e-07	-50.000447
10	0.127218	8.521939e-08	4.478417e-08	-4.043521e-08	-47.392763
14	0.178106	4.312921e-08	2.297444e-08	-2.015477e-08	-46.623029
12	0.152662	5.887613e-08	3.140312e-08	-2.747301e-08	-46.583272

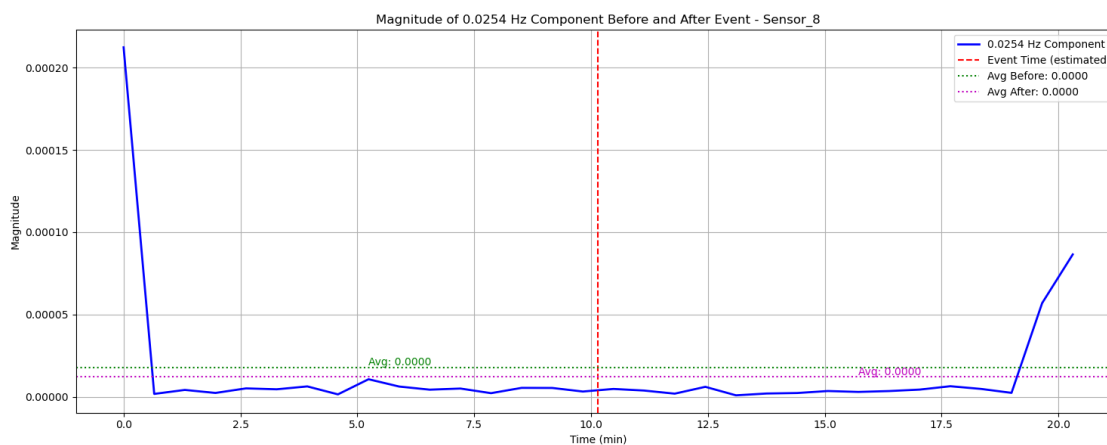
=== Analysis for Sensor\_8 ===

Analyzing frequency: 0.0254 Hz (closest to 0.02 Hz)

Average magnitude before event: 0.0000

Average magnitude after event: 0.0000

Change: -0.0000 (-31.18%)



Energy Analysis:

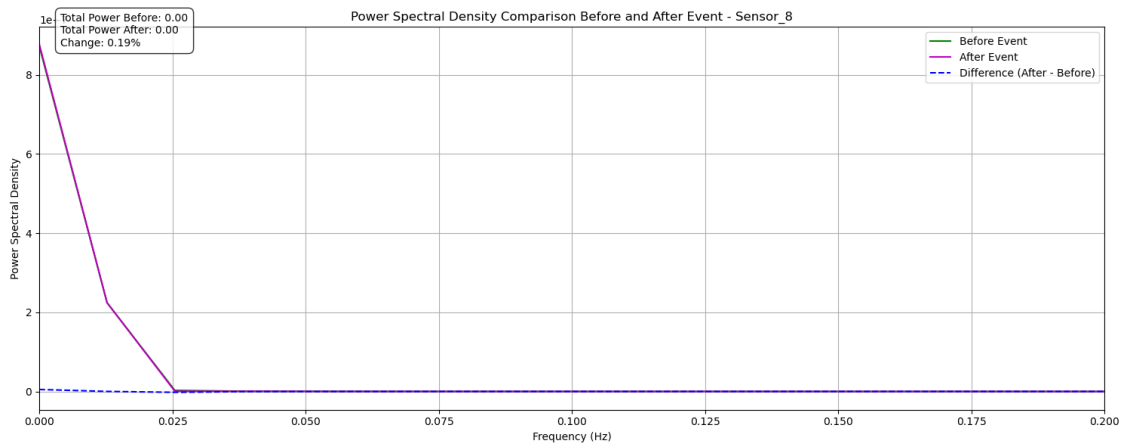
Total energy before event: 0.0000

Total energy after event: 0.0000

Average energy before event: 0.0000

Average energy after event: 0.0000

Energy change: -0.0000 (-75.96%)



#### Power Spectral Density Analysis:

Total power before event: 0.0000

Total power after event: 0.0000

Absolute power change: 0.0000

Relative power change: 0.19%

Saved significant changes data to:

significant\_changes\_csv\_Mushroom\_25-05-08\_0326\Sensor\_8\_significant\_changes.csv

#### Top 5 frequencies with largest power increase:

	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	8.717563e-07	8.768461e-07	5.089855e-09	0.583795
1	0.012722	2.240795e-07	2.243180e-07	2.385210e-10	0.106397
116	1.475732	1.311825e-12	1.396380e-12	8.455424e-14	0.083459
92	1.170408	1.547388e-12	1.497515e-12	-4.987296e-14	-0.049113
71	0.903250	1.880897e-12	1.827187e-12	-5.371029e-14	-0.052719

#### Top 5 frequencies with largest power decrease:

	Frequency	Before	After	Absolute_Change	Percent_Change
2	0.025444	2.841301e-09	7.270722e-10	-2.114229e-09	-71.880737
3	0.038165	7.278655e-10	4.536506e-10	-2.742150e-10	-33.123127
6	0.076331	1.828052e-10	9.041484e-11	-9.239040e-11	-32.669267
4	0.050887	4.277626e-10	2.619809e-10	-1.657818e-10	-31.412186
5	0.063609	2.386392e-10	1.451597e-10	-9.347942e-11	-27.604432

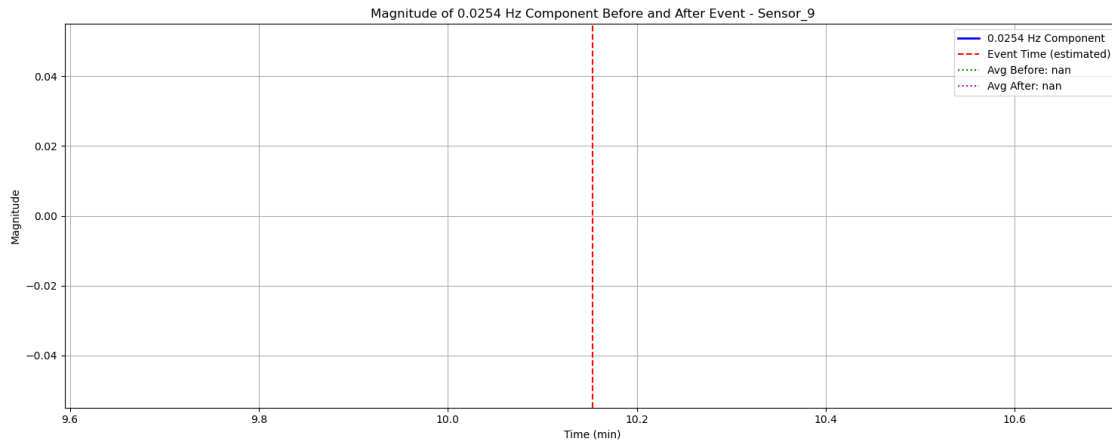
#### === Analysis for Sensor\_9 ===

Analyzing frequency: 0.0254 Hz (closest to 0.02 Hz)

Average magnitude before event: nan

Average magnitude after event: nan

Change: nan (nan%)



#### Energy Analysis:

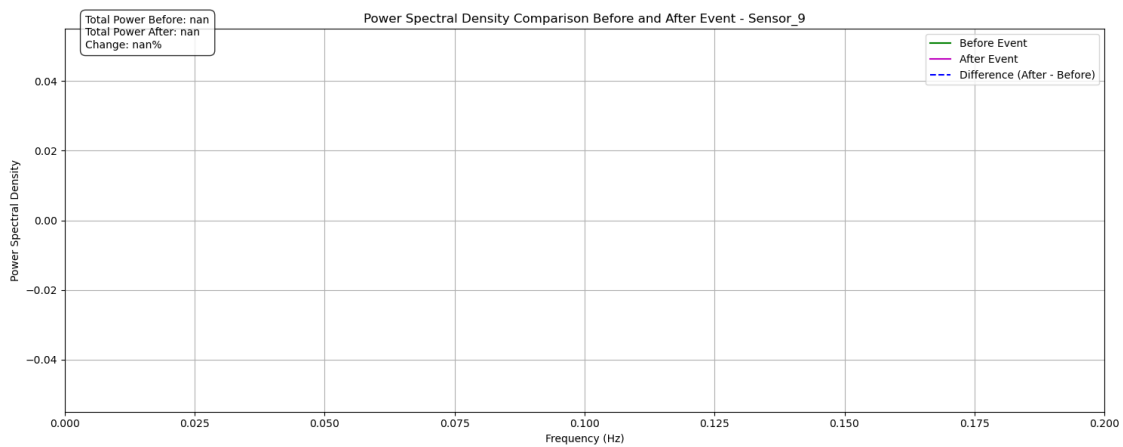
Total energy before event: nan

Total energy after event: nan

Average energy before event: nan

Average energy after event: nan

Energy change: nan (nan%)



#### Power Spectral Density Analysis:

Total power before event: nan

Total power after event: nan

Absolute power change: nan

Relative power change: nan%

Saved significant changes data to:

significant\_changes\_csv\_Mushroom\_25-05-08\_0326\Sensor\_9\_significant\_changes.csv

Top 5 frequencies with largest power increase:

	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	NaN	NaN	NaN	NaN
1	0.012722	NaN	NaN	NaN	NaN
2	0.025444	NaN	NaN	NaN	NaN
3	0.038165	NaN	NaN	NaN	NaN
4	0.050887	NaN	NaN	NaN	NaN

Top 5 frequencies with largest power decrease:

	Frequency	Before	After	Absolute_Change	Percent_Change
0	0.000000	NaN	NaN	NaN	NaN
1	0.012722	NaN	NaN	NaN	NaN
2	0.025444	NaN	NaN	NaN	NaN
3	0.038165	NaN	NaN	NaN	NaN
4	0.050887	NaN	NaN	NaN	NaN

```
[9]: import seaborn as sns

# Analyze significant changes across all sensors
print("\nAnalyzing significant changes across all sensors...")

# Define the directory containing the CSV files
csv_dir_path = "significant_changes_csv_Mushroom_25-05-08_0326"

# Get all CSV files in the directory
csv_files = [f for f in os.listdir(csv_dir_path) if f.
              ↪endswith('_significant_changes.csv')]

# Initialize lists to store summary data
sensor_names = []
top_increase_freqs = []
top_decrease_freqs = []
all_sensor_data = {}

# Create a figure for comparing all sensors
plt.figure(figsize=(15, 6))

# Process each sensor's data
for csv_file in csv_files:
    # Extract sensor name from filename
    sensor_name = csv_file.split('_significant_changes.csv')[0]
    sensor_names.append(sensor_name)

    # Load the CSV data
    csv_path = os.path.join(csv_dir_path, csv_file)
    sensor_data = pd.read_csv(csv_path)
    all_sensor_data[sensor_name] = sensor_data
```

```

# Sort by absolute percent change
sensor_data['Abs_Percent_Change'] = np.abs(sensor_data['Percent_Change'])

# Get top increases and decreases
top_increases = sensor_data.sort_values('Percent_Change', ascending=False).
↳head(20)
top_increase_freqs.append(top_increases['Frequency'].tolist())

top_decreases = sensor_data.sort_values('Percent_Change', ascending=True).
↳head(20)
top_decrease_freqs.append(top_decreases['Frequency'].tolist())

# Plot frequency vs percent change for this sensor
plt.scatter(sensor_data['Frequency'], sensor_data['Percent_Change'],
            alpha=0.3, label=sensor_name)

# Add plot details
plt.axhline(y=0, color='k', linestyle='--', alpha=0.3)
plt.xlabel('Frequency (Hz)')
plt.ylabel('Percent Change (%)')
plt.title('Frequency Distribution of Power Changes - All Sensors')
plt.legend()
plt.grid(True, alpha=0.3)
plt.tight_layout()
plt.show()

# Analyze patterns in top increases and decreases
print("\nAnalyzing patterns in top increases and decreases...")

# Create figures for top increases and decreases
plt.figure(figsize=(15, 6))
plt.subplot(1, 2, 1)
for i, sensor_name in enumerate(sensor_names):
    sensor_data = all_sensor_data[sensor_name]
    top_increases = sensor_data.sort_values('Percent_Change', ascending=False).
↳head(10)
    plt.scatter(top_increases['Frequency'], top_increases['Percent_Change'],
                label=sensor_name, s=100, alpha=0.7)

# Removed annotation of frequencies to avoid overlapping text

plt.title('Top 10 Frequencies with Largest Increases')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Percent Change (%)')
plt.legend()
plt.grid(True, alpha=0.3)

```

```

plt.subplot(1, 2, 2)
for i, sensor_name in enumerate(sensor_names):
    sensor_data = all_sensor_data[sensor_name]
    top_decreases = sensor_data.sort_values('Percent_Change', ascending=True).
↳head(10)
    plt.scatter(top_decreases['Frequency'], top_decreases['Percent_Change'],
                label=sensor_name, s=100, alpha=0.7)

    # Removed annotation of frequencies to avoid overlapping text

plt.title('Top 10 Frequencies with Largest Decreases')
plt.xlabel('Frequency (Hz)')
plt.ylabel('Percent Change (%)')
plt.legend()
plt.grid(True, alpha=0.3)

plt.tight_layout()
plt.show()

# Analyze frequency overlap between sensors for top increases and decreases
print("\nAnalyzing frequency overlap between sensors...")

# For increases
increase_overlap = set(top_increase_freqs[0])
for freqs in top_increase_freqs[1:]:
    increase_overlap = increase_overlap.intersection(set(freqs))

# For decreases
decrease_overlap = set(top_decrease_freqs[0])
for freqs in top_decrease_freqs[1:]:
    decrease_overlap = decrease_overlap.intersection(set(freqs))

print(f"Common frequencies showing increases across all sensors:↳
↳{sorted(list(increase_overlap))}")
print(f"Common frequencies showing decreases across all sensors:↳
↳{sorted(list(decrease_overlap))}")

# Analyze the distribution of top changes by frequency range
for sensor_name in sensor_names:
    sensor_data = all_sensor_data[sensor_name]

    # Define frequency bands
    sensor_data['Frequency_Band'] = pd.cut(sensor_data['Frequency'],
                                           bins=[0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.
↳6, 0.7],

```



```

                                labels=['0-0.1', '0.1-0.2', '0.2-0.
↪3', '0.3-0.4', '0.4-0.5', '0.5-0.6', '0.6-0.7'])

    # Count top increases and decreases by frequency band
    top_increases = sensor_data.sort_values('Percent_Change', ascending=False).
↪head(20)
    top_decreases = sensor_data.sort_values('Percent_Change', ascending=True).
↪head(20)

    increase_band_counts = top_increases['Frequency_Band'].value_counts().
↪sort_index()
    decrease_band_counts = top_decreases['Frequency_Band'].value_counts().
↪sort_index()

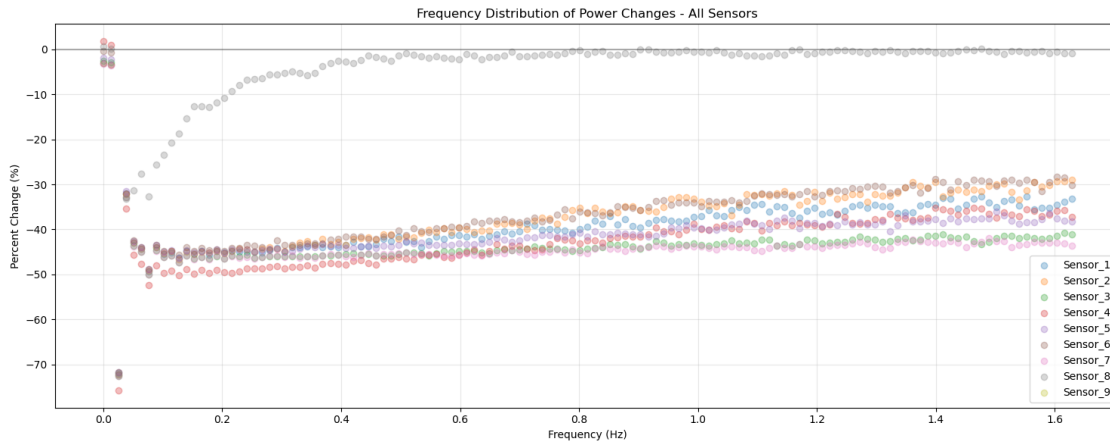
    # Plot distribution of top changes by frequency band
    plt.figure(figsize=(15, 6))
    plt.subplot(1, 2, 1)
    increase_band_counts.plot(kind='bar', color='green', alpha=0.7)
    plt.title(f'{sensor_name}: Distribution of Top 20 Increases by Frequency_
↪Band')
    plt.xlabel('Frequency Band (Hz)')
    plt.ylabel('Count')
    plt.grid(True, alpha=0.3)

    plt.subplot(1, 2, 2)
    decrease_band_counts.plot(kind='bar', color='red', alpha=0.7)
    plt.title(f'{sensor_name}: Distribution of Top 20 Decreases by Frequency_
↪Band')
    plt.xlabel('Frequency Band (Hz)')
    plt.ylabel('Count')
    plt.grid(True, alpha=0.3)

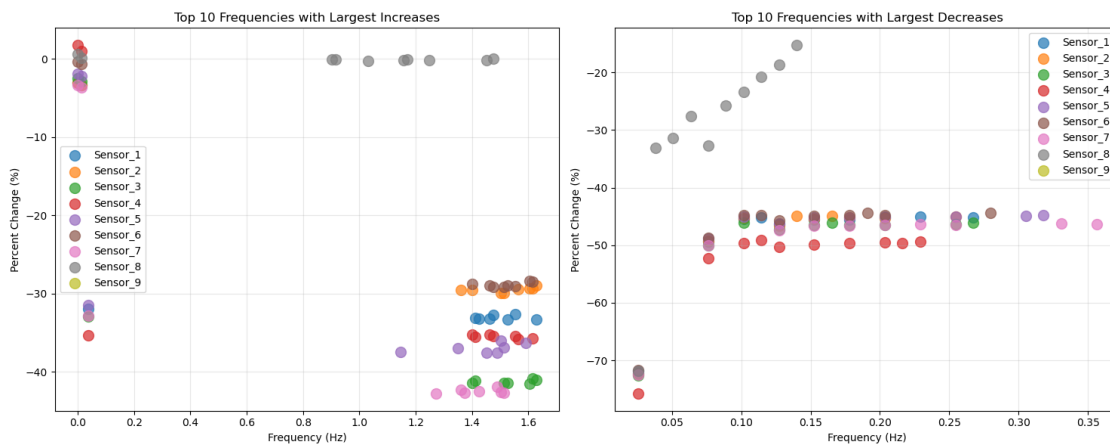
    plt.tight_layout()
    plt.show()

```

Analyzing significant changes across all sensors...



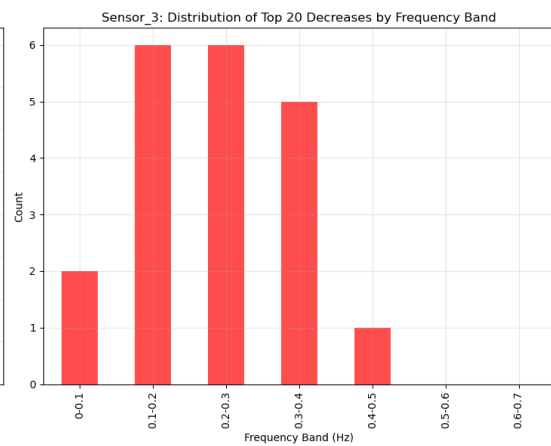
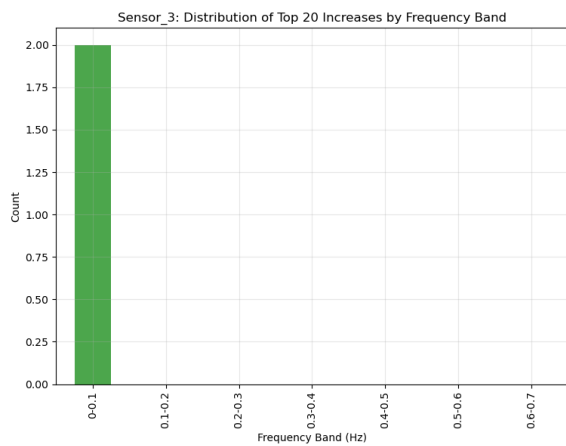
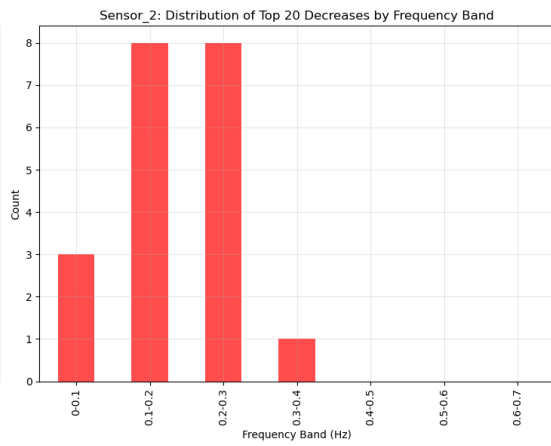
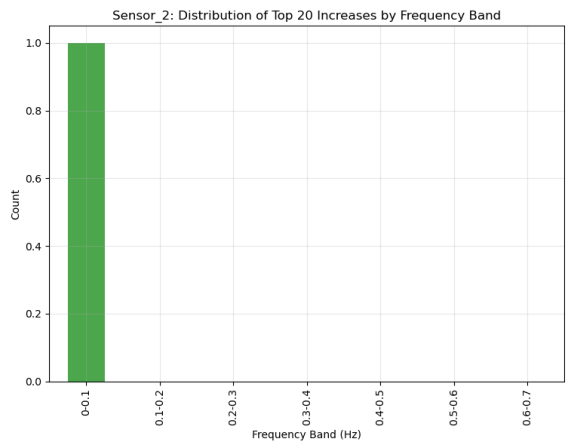
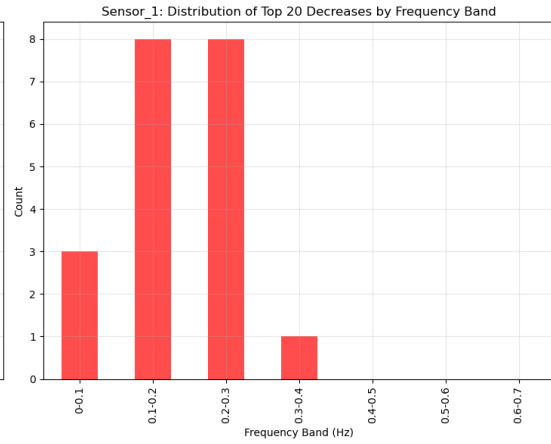
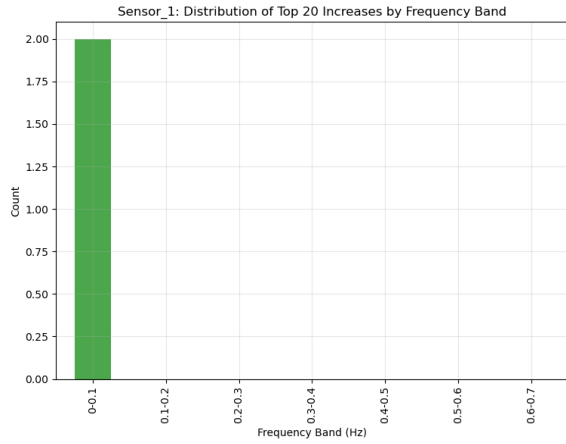
Analyzing patterns in top increases and decreases...

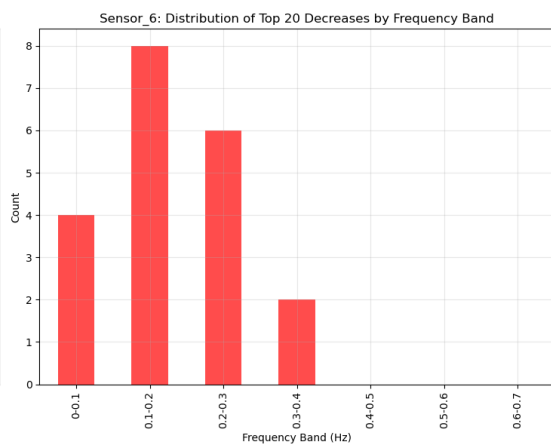
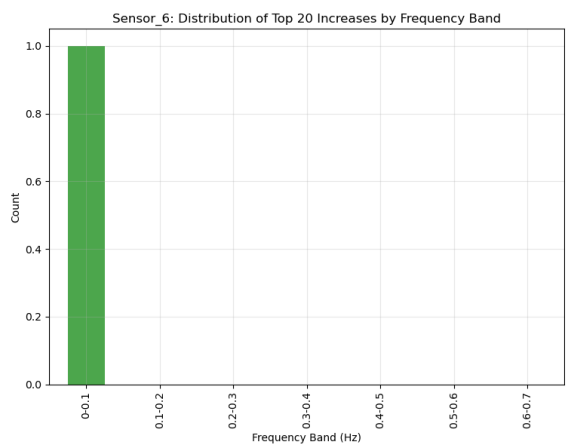
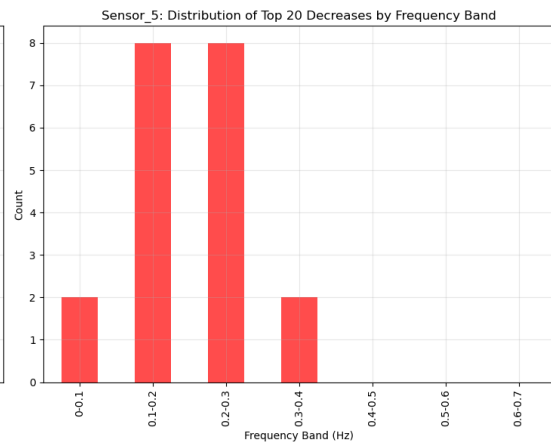
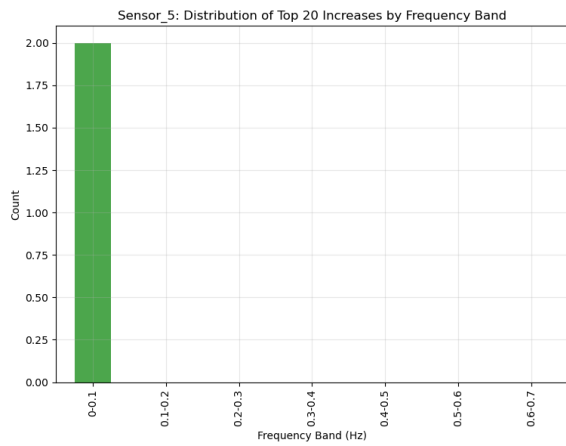
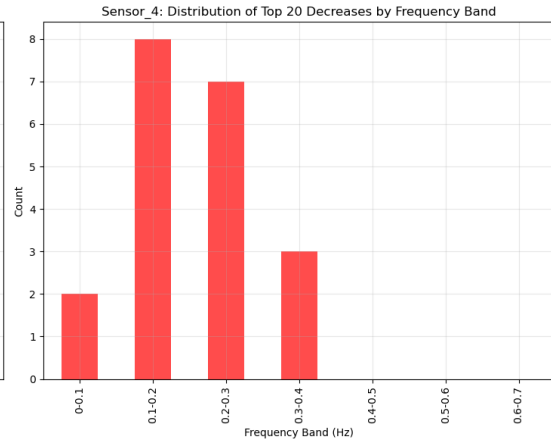
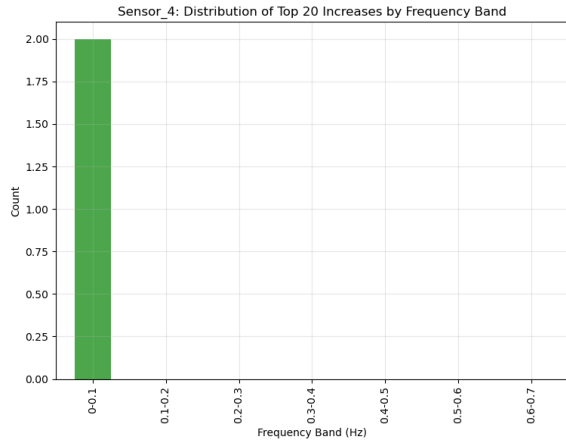


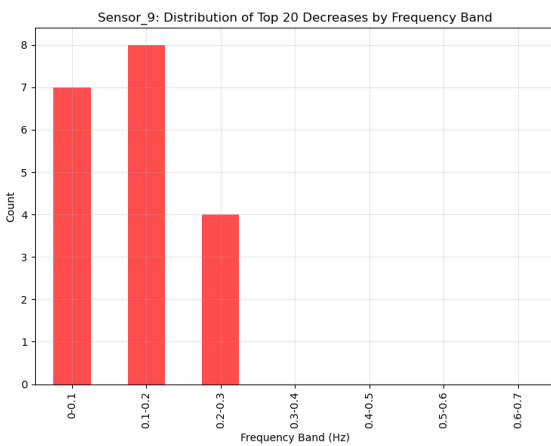
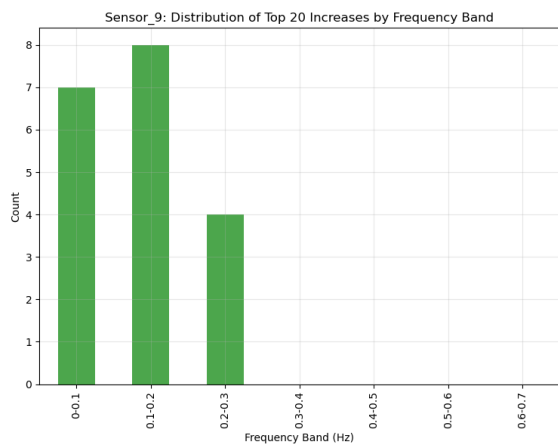
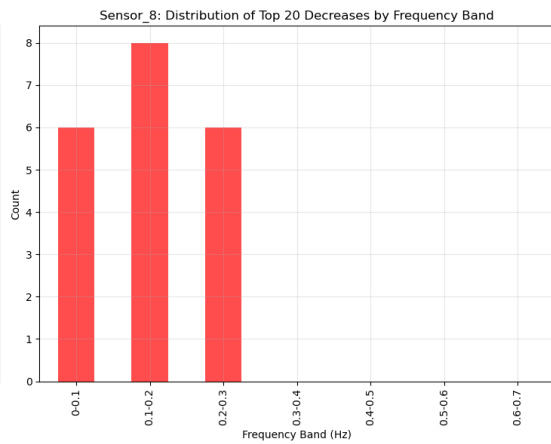
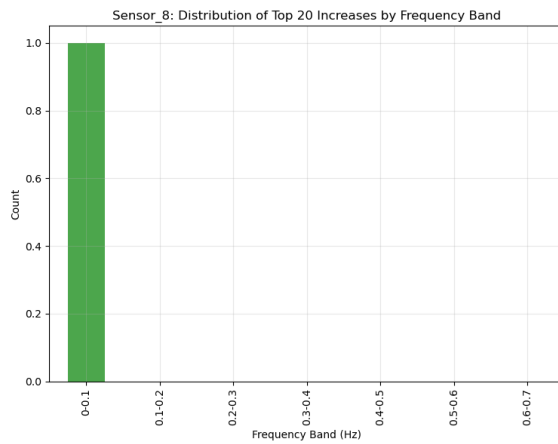
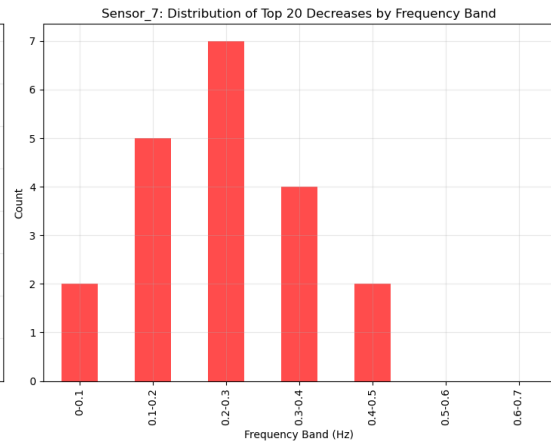
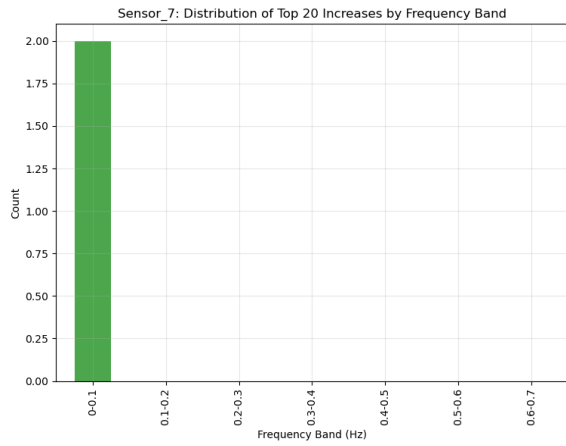
Analyzing frequency overlap between sensors...

Common frequencies showing increases across all sensors: [0.0, 0.0127218279707093]

Common frequencies showing decreases across all sensors: [0.0254436559414187, 0.0763309678242562, 0.101774623765675, 0.1272182797070937, 0.1526619356485125, 0.1653837636192219, 0.1781055915899312, 0.20354924753135, 0.2289929034727688]







[ ]: