Getting rid of electromagnetic interference (hissing, humming and other nondesirable noise)

For acoustic analysis, we usually aim to get audio recordings that faithfully reflext original vocal production. However, in many cases, recordings are contain various unwanted noise sources. These can include: electromagnetic interference (hissing, humming), background noise, and other artifacts. While denoising algorithms can help reduce these unwanted noise sources, they can also introduce distortions to the (speech) signal, altering acoustic measurements such as amplitude, fundamental frequency (f0), and formant frequencies (F1, F2, F3).

Noise in recorded audio can most commonly come from:

- Electromagnetic interference from electronic devices (hissing, humming); usually caused by poor shielding of cables or recording equipment.
- Broadband noise from environment (e.g., air conditioners, computer fans).

One might also want to remove noise in form of breath noises, but because they are physiogical (and physiologically relevant), we do not consider them as noise here.

There is several reasons why is it difficult to remove noise:

1. Noise overlaps with speech/vocalization spectrum

Human speech have energy ranging from ca. 80 Hz to over 4-8 kHz depending on the sound. Noise sources often occupy similar frequency region, making it difficult to separate noise from the actual signal. Simply removing entire frequency bands would therefore also remove important (speech) information.

2. Spectrum changes affect formants and f0

Even small changes in the spectrum can affect formant frequencies (F1, F2, F3), as they are commonly estimated through LPC (linear predictive coding) or formant-peak analysis and depend on the exact shape of harmonic peaks. Denoising usually creates shifts in formant frequencies, increase frame-to-frame jitter (variability) and can cause mis-tracking.

Even common techniques such as notch filtering, FFT-domain attenuation nor algorithms such as noisereduce do not leave the signal completely unaltered. Removing tone can remove harmonics or formant energy, while noise reduction tends to smooth spectral peaks, reduce high-frequency detail or alter amplitude modulation patterns.

Importantly, denoising often leaves residual artifacts, such as metallic or underwater-like sound, which can also affect acoustic measurements and the perceptual quality of the recording.

Every denoising is therefore a trade-off between preserving the original acoustic structure and removing unwanted noise. This also highly depends on what is the purpose of the recording and what analysis is to be performed.

Goal of this notebook

This notebook aims to:

1. Demonstrate how to remove (mainly) tonal electrical noise from recorded vocalizations using combination of FFT notch filtering and spectral gating (noise reduction).
2. Compare several denoisig profiles ranging from very conservative (minimal denoising) to aggressive (maximal denoising).
3. Quantify distortions introduced by each profile using bias in acoustic features, preservation of acoustic shape and stability of dynamic features.

The notebook introduces a reproducible workflow that can be further adapted to accomodate different types of noises and analysis needs.

Prerequisites

install requirements

Preparing environment

```
import os
import glob


# These are the folders and files we will be working with
curfolder = os.getcwd()
noisefolder = os.path.join(curfolder, 'noisyfiles')
noisyfiles = glob.glob(os.path.join(noisefolder, '*.wav'))
```

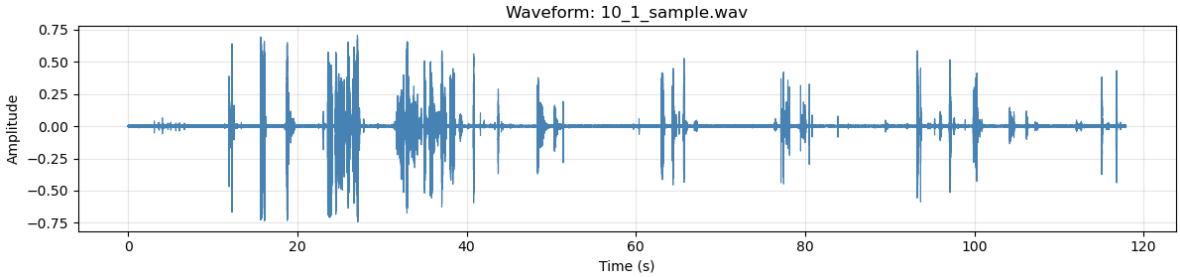
```
# Get rid of those that have strong, balanced, conservative, justNR in their names (these are
noisyfiles = [f for f in noisyfiles if not any(x in f for x in ['strong', 'balanced', 'conser

print(f"Found {len(noisyfiles)} noisy audio files in folder: {noisefolder}")
```

```
Found 7 noisy audio files in folder: c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles
```

These are the example files that contain tonal electrical noise (hissing, humming). Notice the thick baseline in the waveform that corresponds to the underlying noise.

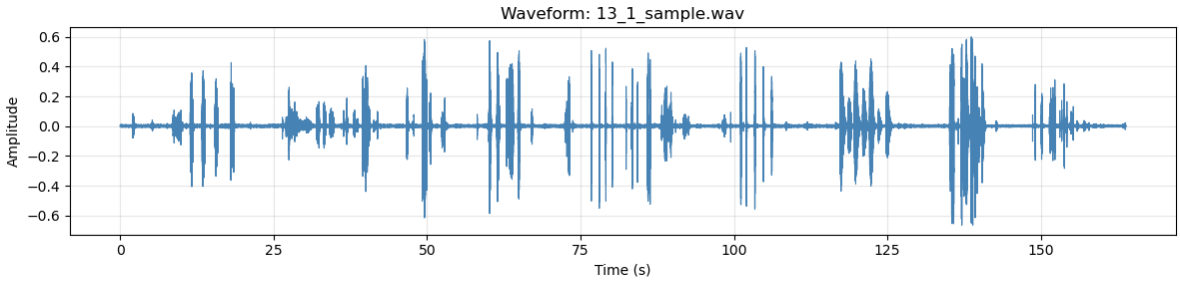
```
import IPython.display as ipd
import matplotlib.pyplot as plt
import soundfile as sf
import numpy as np

for f in noisyfiles:
    print(f"Playing file: {os.path.basename(f)}")

    # Load audio
    y, sr = sf.read(f)
    if y.ndim > 1:
        y = y.mean(axis=1)  # convert to mono for plotting

    # Plot waveform
    plt.figure(figsize=(12, 3))
    t = np.arange(len(y)) / sr
    plt.plot(t, y, color='steelblue', linewidth=0.8)
    plt.title(f"Waveform: {os.path.basename(f)}")
    plt.xlabel("Time (s)")
    plt.ylabel("Amplitude")
    plt.grid(True, alpha=0.3)
    plt.tight_layout()
    plt.show()

    # Play audio
    display(ipd.Audio(y, rate=sr))
```

```
Playing file: 10_1_sample.wav
```



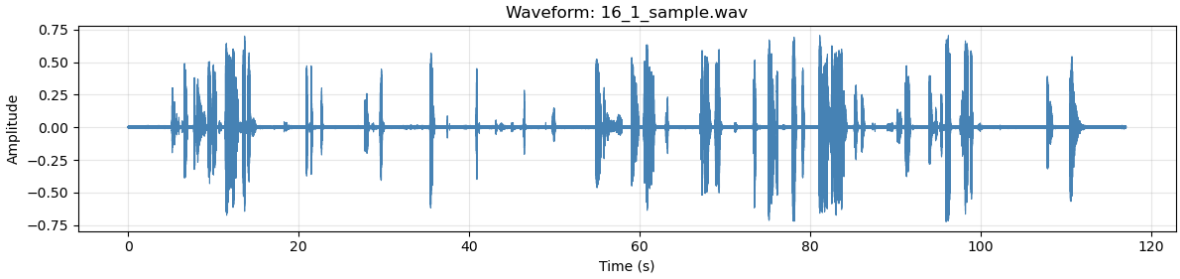
<IPython.lib.display.Audio object>

Playing file: 13_1_sample.wav



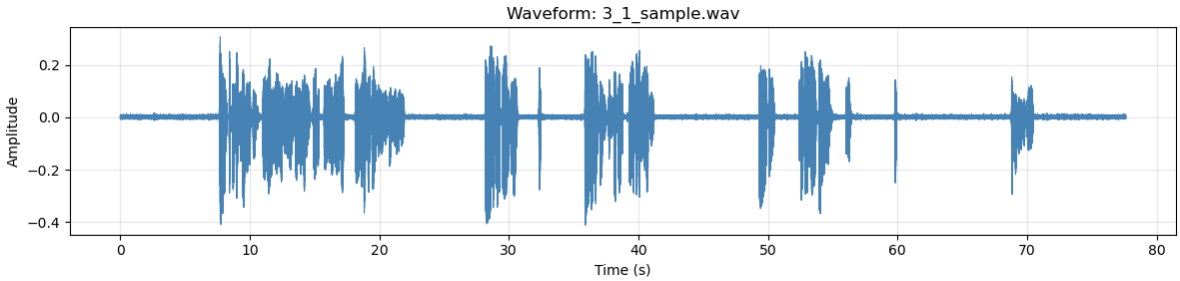
<IPython.lib.display.Audio object>

Playing file: 16_1_sample.wav



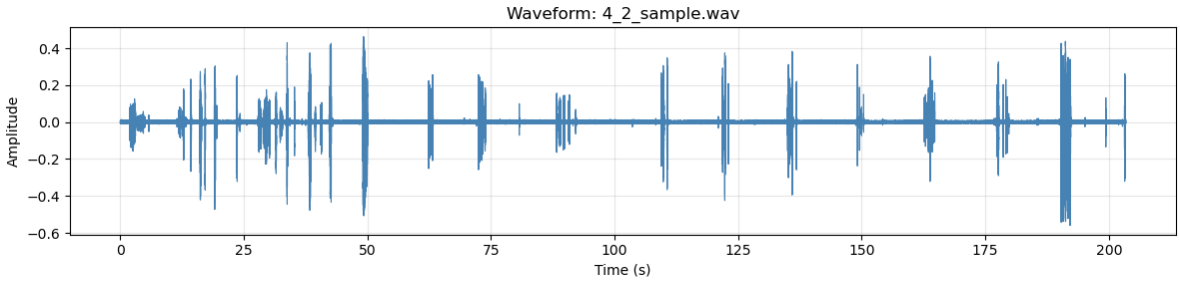
<IPython.lib.display.Audio object>

Playing file: 3_1_sample.wav



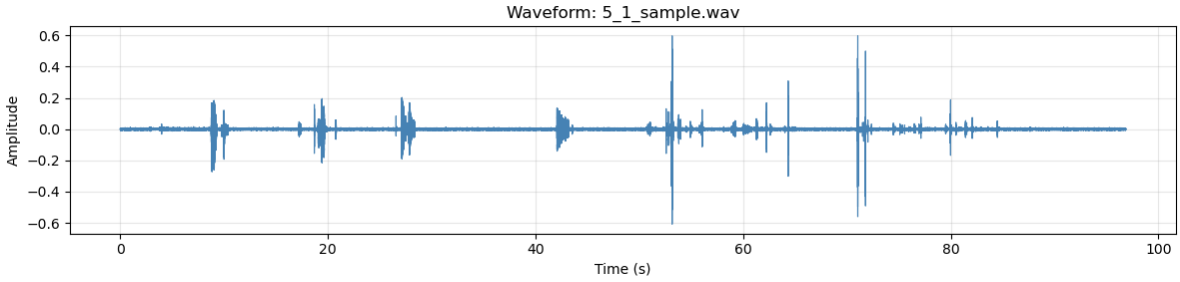
<IPython.lib.display.Audio object>

Playing file: 4_2_sample.wav



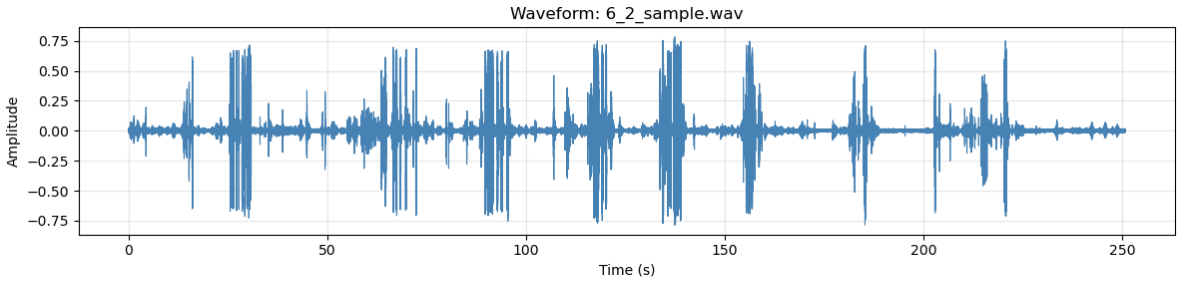
<IPython.lib.display.Audio object>

Playing file: 5_1_sample.wav



<IPython.lib.display.Audio object>

Playing file: 6_2_sample.wav




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

Using FFT to attenuate hiss in audio files

Very often, gating algorithms such as noisereduce are not sufficient to remove tonal electrical noise (hissing, humming). This is because such noise is often very strong and concentrated at specific frequencies.

For this reason, we will apply Fast Fourier Transform (FFT) to identify the main frequencies of the noise, and then apply notch filters to attenuate those frequencies. To understand FFT, you can watch Steve Brunton's excellent video on YouTube. The following code was also inspired by this StackOverflow post.

To understand the workflow, we will first demonstrate the steps on a single audio file, and then we will apply the same procedure to all files in the folder, using different denoising profiles.

Step-by-step FFT denoising demonstration

First, we will compute median spectrum across all noisy files to identify the main frequencies of the noise. We leverage the fact that the noise is consistent across all files, while the speech/vocalization content varies. Therefore, by averaging across all files, we suppress file-specific content (here vocalizations) and highlight stable, persistent lines. Afterwards, we will identify the main (most prominent, well-separated) peaks in the averaged spectrum.

```
import soundfile as sf
import noisereduce as nr
from scipy.io import wavfile
from scipy.signal import find_peaks, peak_widths

data, rate = sf.read(noisyfiles[1]) # load in one of the files with noise
```

```
def compute_median_spectrum(file_list, n_fft=65536, max_files=40):
    """
    Compute median magnitude spectrum across a set of files.
    Use this to find persistent (environment) spectral lines.
    """
    mags = []
    sr_ref = None
    for i, f in enumerate(file_list):
        if i >= max_files:
            break
        try:
            data, sr = sf.read(f)
        except Exception:
            continue
        if data.ndim == 2:
            data = data.mean(axis=1)
        if sr_ref is None:
            sr_ref = sr
        # trim or pad to reasonable length if file is huge (keeps memory bounded)
        if len(data) > 10 * sr_ref:  # use first 10s
            data = data[:10 * sr_ref]
        FFT = np.fft.rfft(data, n=n_fft)
        mags.append(np.abs(FFT))
    if len(mags) == 0:
        raise RuntimeError("No spectra computed")
    mags = np.vstack(mags)
    median_mag = np.median(mags, axis=0)
    freq = np.fft.rfftfreq(n_fft, d=1.0 / sr_ref)
    return freq, median_mag, sr_ref

def detect_persistent_peaks(freq, median_mag, min_freq=1000, prominence_frac=0.05, min_sep_hz
    bins_per_hz = len(freq) / (freq.max() if freq.max()>0 else 1.0)
    distance_bins = max(1, int(min_sep_hz * bins_per_hz))
    prominence = max(1e-12, np.max(median_mag) * prominence_frac)
    peaks, _ = find_peaks(median_mag, prominence=prominence, distance=distance_bins)
    peaks = peaks[freq[peaks] >= min_freq]
    return freq[peaks].tolist()

# Compute median spectrum from example noise files
freq_ref, med_mag, sr_ref = compute_median_spectrum(noisyfiles, n_fft=65536, max_files=30)

# Print the results
```

```
print(f"Computed median spectrum from {len(noisyfiles)} files at sample rate {sr_ref} Hz")
print(f"Median spectrum has {len(med_mag)} frequency bins up to {freq_ref[-1]:.1f} Hz")

# Detect persistent peaks in the median spectrum
persistent = detect_persistent_peaks(freq_ref, med_mag, min_freq=900, prominence_frac=0.03)
print(f"Detected {len(persistent)} persistent peaks at frequencies: {persistent}")
```

```
Computed median spectrum from 7 files at sample rate 48000 Hz
Median spectrum has 32769 frequency bins up to 24000.0 Hz
Detected 54 persistent peaks at frequencies: [900.146484375, 950.68359375, 1000.48828125, 104
```

To attenuate the identified noise frequencies, we will apply notch filters at those frequencies. A notch filter is a type of band-stop filter that attenuates a narrow frequency band while leaving other frequencies relatively unaffected. By applying notch filters at the identified noise frequencies, we can effectively reduce the tonal noise while preserving the overall structure of the audio signal.

The `erase_freq()` function takes in data and compute Fast Fourier Transform (FFT) to convert the signal to frequency domain. It then find peaks in the recording and builds a list of frequencies to attenuate. This list can be either given (`targets`) or if `targets==None`, it picks those peaks that are near persistent ones (within `match_tol_hz`), or narrow enough (`max_width_hz`). It then applies notch filters at those frequencies by building Hanning windows and scaling the FFT bins accordingly. As a result of Hanning window, the middle of the band is attenuated the most, edges are tapered smoothly. This is to prevent ringing artifacts that would occur with hard brick notch (setting band to zero).

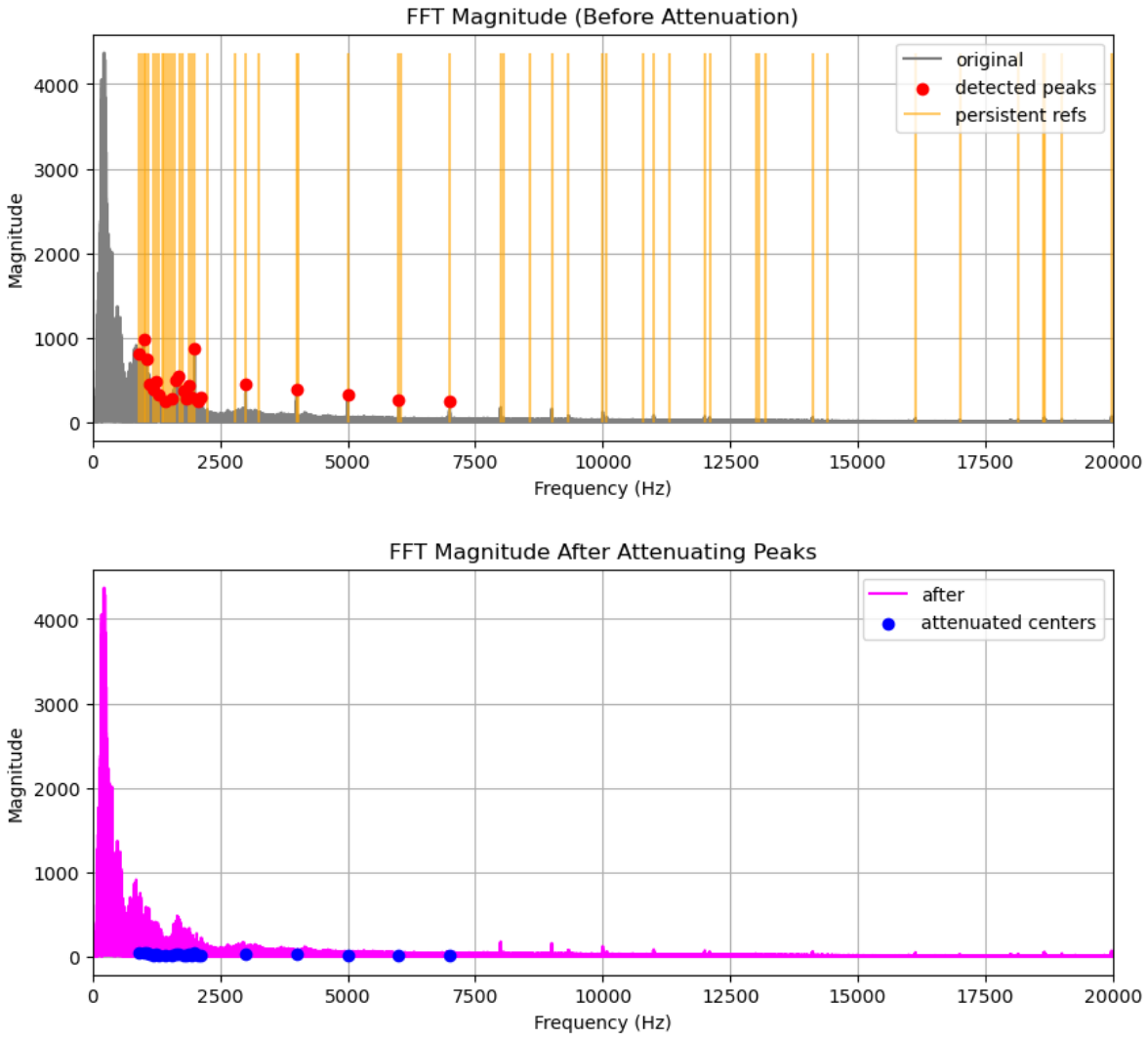
Finally, it converts the signal back to time domain using inverse FFT.

```
def attenuate_freq_band(FFT, freq, center_hz, width_hz, attenuation=0.9):
    idx_band = np.where((freq >= center_hz - width_hz / 2) & (freq <= center_hz + width_hz /
    if idx_band.size == 0:
        return FFT
    win = np.hanning(len(idx_band))
    multiplier = 1.0 - attenuation * win
    FFT[idx_band] = FFT[idx_band] * multiplier
    return FFT

def erase_freq(data, rate, targets=None, width_hz=25, plot_before=True, plot_after=True,
               attenuation=0.9, min_freq=900, min_sep_hz=50, prominence_frac=0.05,
               persistent_refs=None, match_tol_hz=2.0, max_width_hz=10.0):
    """
    If targets provided: attenuate those.
```

```
    If targets is None: auto-detect peaks in this recording but ONLY attenuate those that:
      - match a persistent_refs list (if provided), OR
      - are very narrow peaks (width < max_width_hz) indicating tonal electrical lines.
    persistent_refs: list of persistent center Hz (from compute_median_spectrum -> detect_pe
    match_tol_hz: tolerance for matching a recording peak to a persistent ref (Hz)
    max_width_hz: consider peaks narrower than this as likely interference
    """
    FFT_data = np.fft.rfft(data)
    freq = np.fft.rfftfreq(len(data), d=1.0 / rate)
    magnitude = np.abs(FFT_data)

    # find peaks in this recording
    bins_per_hz = len(freq) / (rate / 2.0)
    distance_bins = max(1, int(min_sep_hz * bins_per_hz))
    prominence = max(1e-12, np.max(magnitude) * prominence_frac)
    peaks, _ = find_peaks(magnitude, prominence=prominence, distance=distance_bins)
    peaks = peaks[freq[peaks] >= min_freq]

    # compute widths in bins and convert to Hz
    if len(peaks) > 0:
        results_width = peak_widths(magnitude, peaks, rel_height=0.5)
        widths_bins = results_width[0]
        widths_hz = widths_bins / bins_per_hz
    else:
        widths_hz = np.array([])

    # prepare list of centers to actually attenuate
    centers_to_attenuate = []

    if targets:
        # explicit targets override
        centers_to_attenuate = [t for t in targets if t >= min_freq]
    else:
        for i, p in enumerate(peaks):
            hz = freq[p]
            w = widths_hz[i] if i < len(widths_hz) else np.inf
            is_narrow = (w <= max_width_hz)
            matches_persistent = False
            if persistent_refs is not None and len(persistent_refs) > 0:
                # match within tolerance
                if np.any(np.isclose(hz, persistent_refs, atol=match_tol_hz)):
                    matches_persistent = True
```

```
            # Only attenuate if it's persistent OR narrow tonal
            if matches_persistent or is_narrow:
                centers_to_attenuate.append(hz)

    if plot_before:
        plt.figure(figsize=(10, 4))
        plt.plot(freq, magnitude, color='gray', label='original')
        if len(peaks):
            plt.scatter(freq[peaks], magnitude[peaks], color='red', zorder=5, label='detecte
        if persistent_refs:
            # mark persistent refs
            plt.vlines(persistent_refs, ymin=0, ymax=np.max(magnitude), color='orange', alpha
        plt.title("FFT Magnitude (Before Attenuation)")
        plt.xlabel("Frequency (Hz)")
        plt.ylabel("Magnitude")
        plt.xlim(0, 20000)
        plt.legend()
        plt.grid(True)
        plt.show()

    # Apply attenuation around each chosen center
    for hz in centers_to_attenuate:
        FFT_data = attenuate_freq_band(FFT_data, freq, hz, width_hz, attenuation)

    if plot_after:
        plt.figure(figsize=(10, 4))
        plt.plot(freq, np.abs(FFT_data), color='magenta', label='after')
        if centers_to_attenuate:
            plt.scatter(centers_to_attenuate, np.abs(FFT_data)[[int(np.argmin(np.abs(freq - h
                        color='blue', zorder=5, label='attenuated centers')
        plt.title("FFT Magnitude After Attenuating Peaks")
        plt.xlabel("Frequency (Hz)")
        plt.ylabel("Magnitude")
        plt.xlim(0, 20000)
        plt.legend()
        plt.grid(True)
        plt.show()

    new_data = np.fft.irfft(FFT_data)
    return new_data

# Now we attenuate globally with value 0.95 (which is quite strong)
```

```
new_data = erase_freq(data, rate, targets=None, width_hz=30, plot_before=True, plot_after=Tru
                      attenuation=0.95, persistent_refs=persistent, match_tol_hz=2.0, max_widt
```



Finally, we will now also use `noisereduce` to further reduce broadband noise.

```
noise_clip = new_data[:int(rate * 0.5)]  # ensure this is noise-only, optional

reduced_noise = nr.reduce_noise(
    y=new_data,
    sr=rate,
    stationary=True,
```

```
    y_noise=noise_clip,
    n_std_thresh_stationary=1.7,
    prop_decrease=0.95,
)
```

Now we can compare the original, fft-only and fft+nr denoised audio.

```
for original, fft_only, fft_nr in zip([data], [new_data], [reduced_noise]):
    print("Original Audio:")
    display(ipd.Audio(original, rate=rate))
    print("FFT-only Denoised Audio:")
    display(ipd.Audio(fft_only, rate=rate))
    print("FFT + Noisereduce Denoised Audio:")
    display(ipd.Audio(fft_nr, rate=rate))
```

```
Original Audio:
```

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

```
FFT-only Denoised Audio:
```

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

```
FFT + Noisereduce Denoised Audio:
```

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

The double denoising indeed seems to remove most of the hiss as well as other broadband noise, while preserving the speech content. However, we have now applied strong attenuation everywhere from 900 Hz upwards. Up until 3500 Hz, there is lot of relevant speech energy that we necessarily alter when applying too strong denoising.

It might be the case that your hiss occupies only higher frequencies, in which case you can set the limit higher and you may get away with less distortion, while taking care of most of the hiss. In the case of these files, hiss seems to occupy also lower frequencies, so we have to apply some denoising across the entire spectrum (or here above 900 Hz seems good enough). However, we can change the strength depending on which frequencies we are dealing with.

In the following, we will do exactly that. We define several regions in the frequency spectrum, and assign them different attenuation values. Our bands are:

- Low: 300-1200 Hz
- Mid: 1200-3500 Hz
- High: 3500-8000 Hz
- Ultra: >8000 Hz

(You can, however, change these bands as you see fit.)

We will also create several denoising profiles, ranging from very conservative (minimal denoising) to very strong (maximal denoising). Each profile will have different attenuation values for each frequency band. Afterwards, we will analyze how each profile affects acoustic measurements to make a principled choice about which profile to use.

The profiles are defined as follows:

```
        "very_conservative": {
            "low":   (4.0, 0.25),   # width_hz, attenuation
            "mid":   (6.0, 0.30),
            "high":  (10.0, 0.45),
            "ultra": (15.0, 0.55),
        },
        "conservative": {
            "low":   (5.0, 0.30),
            "mid":   (8.0, 0.40),
            "high":  (15.0, 0.60),
            "ultra": (20.0, 0.70),
        },
        "balanced": {
            "low":   (6.0, 0.35),
            "mid":   (10.0, 0.45),
            "high":  (20.0, 0.75),
            "ultra": (30.0, 0.90),
        },
        "strong": {
            "low":   (8.0, 0.45),
            "mid":   (12.0, 0.60),
            "high":  (25.0, 0.85),
            "ultra": (35.0, 0.95),
        },
        "very_strong": {
            "low":   (10.0, 0.60),
            "mid":   (15.0, 0.75),
            "high":  (30.0, 0.95),
            "ultra": (40.0, 0.98),
```

```
        },
    }
```

Applying different denoising profiles to all files

```
def erase_freq(
    data,
    rate,
    targets=None,
    plot_before=False,
    plot_after=True,
    min_freq=900,
    min_sep_hz=50,
    prominence_frac=0.05,
    persistent_refs=None,
    match_tol_hz=2.0,
    max_width_hz=10.0,
    profile="balanced"   # choose attenuation profile
):
    """
    Global-FFT tonal attenuation with band-dependent notch parameters.

    If targets provided: attenuate those.
    If targets is None: auto-detect peaks in this recording but ONLY attenuate those that:
      - match a persistent_refs list (if provided), OR
      - are very narrow peaks (width < max_width_hz) indicating tonal electrical lines.

    `profile` controls how strong the attenuation is in each band:
      "very_conservative", "conservative", "balanced", "strong", "very_strong"
    """

    FFT_data = np.fft.rfft(data)
    freq = np.fft.rfftfreq(len(data), d=1.0 / rate)
    magnitude = np.abs(FFT_data)

    # ---------------- PROFILES: band-dependent params ---------------------
    # bands: (low_edge, high_edge)
    # we ignore < 300 Hz here (you can change that if you want)
    BAND_DEFS = [
        ("low",   300.0, 1200.0),
```

```
        ("mid",  1200.0, 3500.0),
        ("high", 3500.0, 8000.0),
        ("ultra",8000.0, np.inf),
    ]

    PROFILES = {
        "very_conservative": {
            "low":   (4.0, 0.25),   # width_hz, attenuation
            "mid":   (6.0, 0.30),
            "high":  (10.0, 0.45),
            "ultra": (15.0, 0.55),
        },
        "conservative": {
            "low":   (5.0, 0.30),
            "mid":   (8.0, 0.40),
            "high":  (15.0, 0.60),
            "ultra": (20.0, 0.70),
        },
        "balanced": {
            "low":   (6.0, 0.35),
            "mid":   (10.0, 0.45),
            "high":  (20.0, 0.75),
            "ultra": (30.0, 0.90),
        },
        "strong": {
            "low":   (8.0, 0.45),
            "mid":   (12.0, 0.60),
            "high":  (25.0, 0.85),
            "ultra": (35.0, 0.95),
        },
        "very_strong": {
            "low":   (10.0, 0.60),
            "mid":   (15.0, 0.75),
            "high":  (30.0, 0.95),
            "ultra": (40.0, 0.98),
        },
    }

    if profile not in PROFILES:
        raise ValueError(f"Unknown profile '{profile}'. "
                         f"Choose one of: {list(PROFILES.keys())}")
    profile_params = PROFILES[profile]
```

```
    # helper to map hz → band name
    def band_for_hz(hz):
        if hz < 300.0:
            return None  # ignore sub-300 Hz in this pipeline
        for band_name, lo, hi in BAND_DEFS:
            if lo <= hz < hi:
                return band_name
        return "ultra"  # just in case

    # helper: choose notch params based on freq & profile
    def choose_notch_params(hz):
        band = band_for_hz(hz)
        if band is None:
            return None, None
        base_width, base_att = profile_params[band]
        # still respect global max_width_hz if given
        if max_width_hz is not None:
            local_width = min(base_width, max_width_hz) if band in ("low", "mid") else base_w
        else:
            local_width = base_width
        local_att = base_att
        return local_width, local_att

    # Peak detection
    bins_per_hz = len(freq) / (rate / 2.0)
    distance_bins = max(1, int(min_sep_hz * bins_per_hz))
    prominence = max(1e-12, np.max(magnitude) * prominence_frac)
    peaks, _ = find_peaks(magnitude, prominence=prominence, distance=distance_bins)
    peaks = peaks[freq[peaks] >= min_freq]

    if len(peaks) > 0:
        results_width = peak_widths(magnitude, peaks, rel_height=0.5)
        widths_bins = results_width[0]
        widths_hz = widths_bins / bins_per_hz
    else:
        widths_hz = np.array([])

    centers_to_attenuate = []

    if targets:
        centers_to_attenuate = [t for t in targets if t >= min_freq]
    else:
```

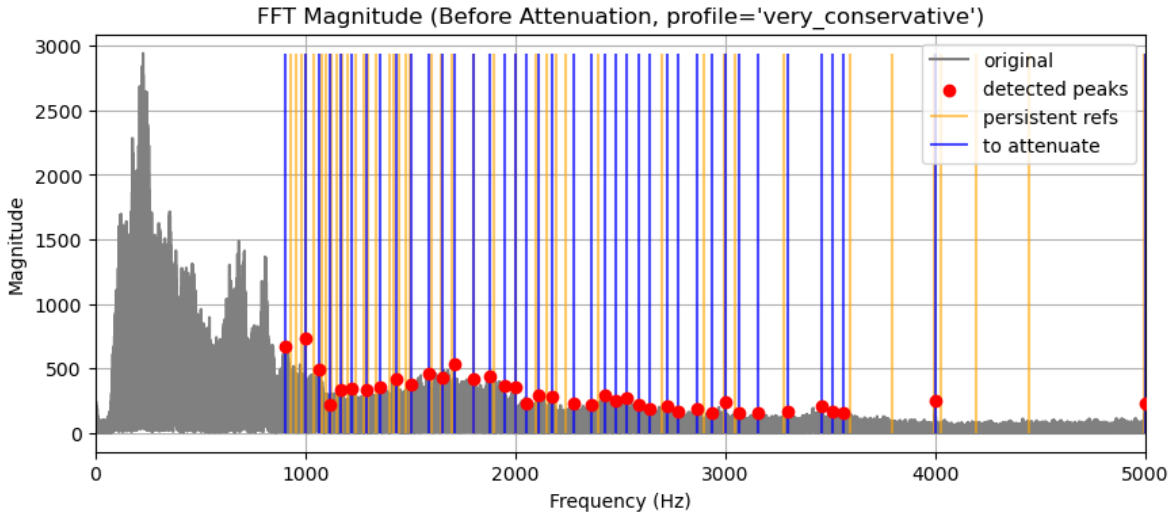
```
        for i, p in enumerate(peaks):
            hz = freq[p]
            w = widths_hz[i] if i < len(widths_hz) else np.inf
            is_narrow = (w <= max_width_hz)
            matches_persistent = False
            if persistent_refs is not None and len(persistent_refs) > 0:
                if np.any(np.isclose(hz, persistent_refs, atol=match_tol_hz)):
                    matches_persistent = True
            if matches_persistent or is_narrow:
                centers_to_attenuate.append(hz)

    # Plot before attenuation
    if plot_before:
        plt.figure(figsize=(10, 4))
        plt.plot(freq, magnitude, color='gray', label='original')
        if len(peaks):
            plt.scatter(freq[peaks], magnitude[peaks],
                        color='red', zorder=5, label='detected peaks')
        if persistent_refs:
            plt.vlines(persistent_refs, ymin=0, ymax=np.max(magnitude),
                       color='orange', alpha=0.6, label='persistent refs')
        if centers_to_attenuate:
            plt.vlines(centers_to_attenuate, ymin=0, ymax=np.max(magnitude),
                       color='blue', alpha=0.7, label='to attenuate')
        plt.title(f"FFT Magnitude (Before Attenuation, profile='{profile}')")
        plt.xlabel("Frequency (Hz)")
        plt.ylabel("Magnitude")
        plt.xlim(0, min(5000, freq.max()))
        plt.legend()
        plt.grid(True)
        plt.show()

    # Apply attenuation
    for hz in centers_to_attenuate:
        local_width, local_att = choose_notch_params(hz)
        if local_width is None:
            continue
        FFT_data = attenuate_freq_band(FFT_data, freq, hz,
                                       width_hz=local_width,
                                       attenuation=local_att)

    # Plot after attenuation
```

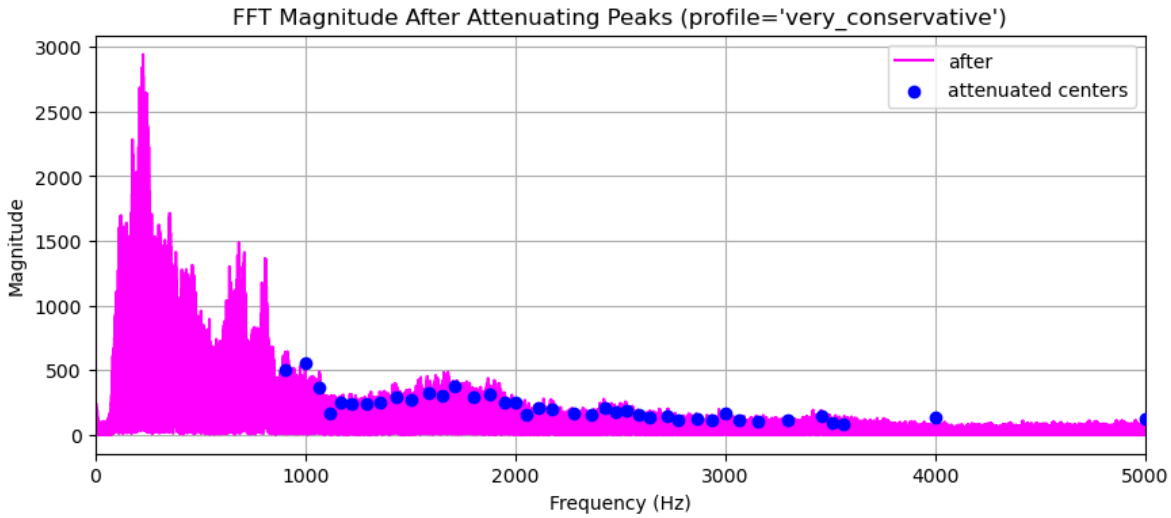
```
    if plot_after:
        mag_after = np.abs(FFT_data)
        plt.figure(figsize=(10, 4))
        plt.plot(freq, mag_after, color='magenta', label='after')
        if centers_to_attenuate:
            idx_centers = [int(np.argmin(np.abs(freq - h))) for h in centers_to_attenuate]
            plt.scatter(centers_to_attenuate, mag_after[idx_centers],
                        color='blue', zorder=5, label='attenuated centers')
        plt.title(f"FFT Magnitude After Attenuating Peaks (profile='{profile}')")
        plt.xlabel("Frequency (Hz)")
        plt.ylabel("Magnitude")
        plt.xlim(0, min(5000, freq.max()))
        plt.legend()
        plt.grid(True)
        plt.show()

    new_data = np.fft.irfft(FFT_data)
    return new_data
```

```
profiles = [
    "very_conservative",
    "conservative",
    "balanced",
    "strong",
    "very_strong",
]

noisyfiles = os.path.join(curfolder, 'noisyfiles')
filestoprocess = glob.glob(os.path.join(noisyfiles, '*.wav'))

for sample in filestoprocess:

    data, rate = sf.read(sample)

    for prof in profiles:
        print("Processing with profile:", prof)
        out = erase_freq(
            data,
            rate,
            min_freq=900,
            min_sep_hz=50,
            prominence_frac=0.05,
```

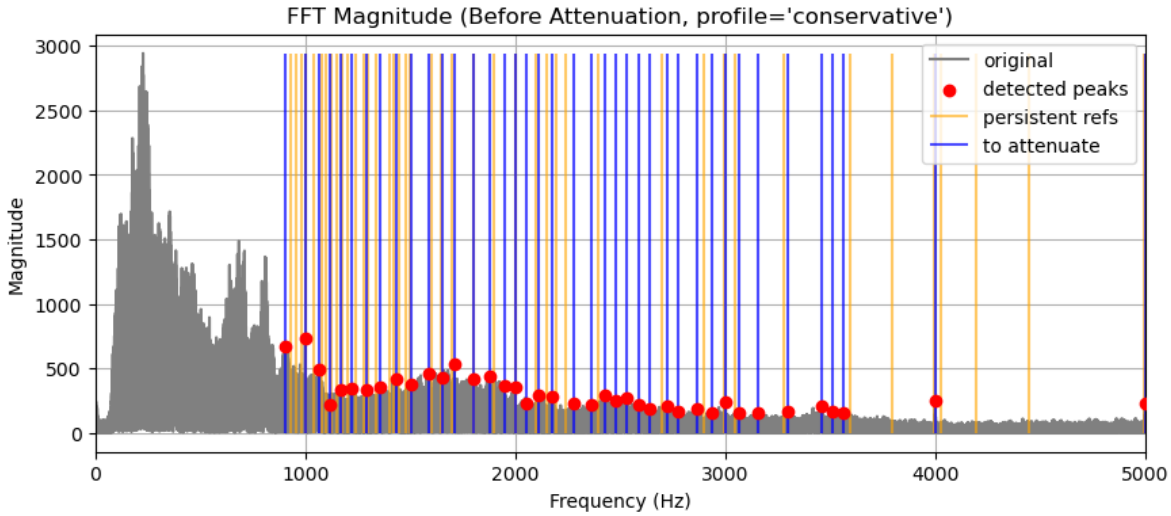
```
            persistent_refs=persistent,
            match_tol_hz=2.0,
            max_width_hz=10.0,    # caps low/mid band widths
            profile=prof,
            plot_before=True,
            plot_after=True
        )

        # then noisereduce
        # noise_clip = out[:int(rate * 0.5)]
        reduced = nr.reduce_noise(
            y=out,
            sr=rate,
            stationary=True,
            n_std_thresh_stationary=1.5,
            prop_decrease=0.8,
        )

        out_path = f"{os.path.splitext(sample)[0]}_{prof}.wav"
        sf.write(out_path, reduced, rate)
        print("Saved", out_path)
```

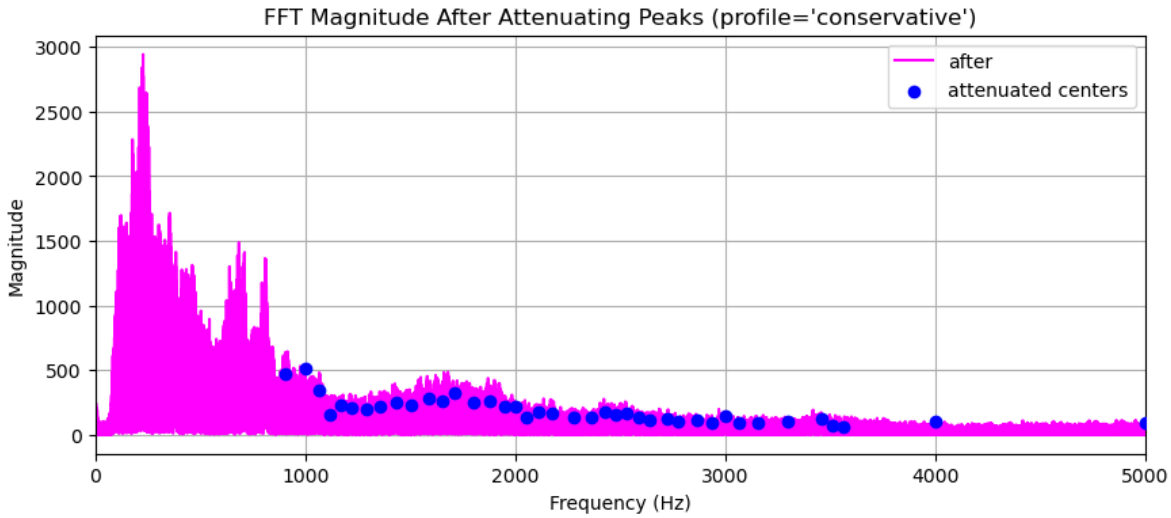
```
Processing with profile: very_conservative
```



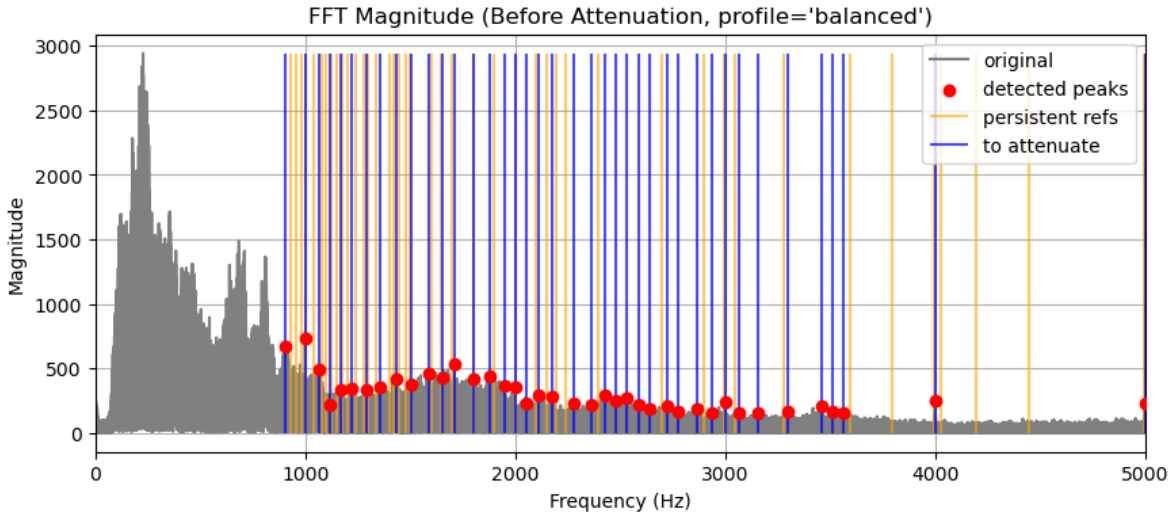


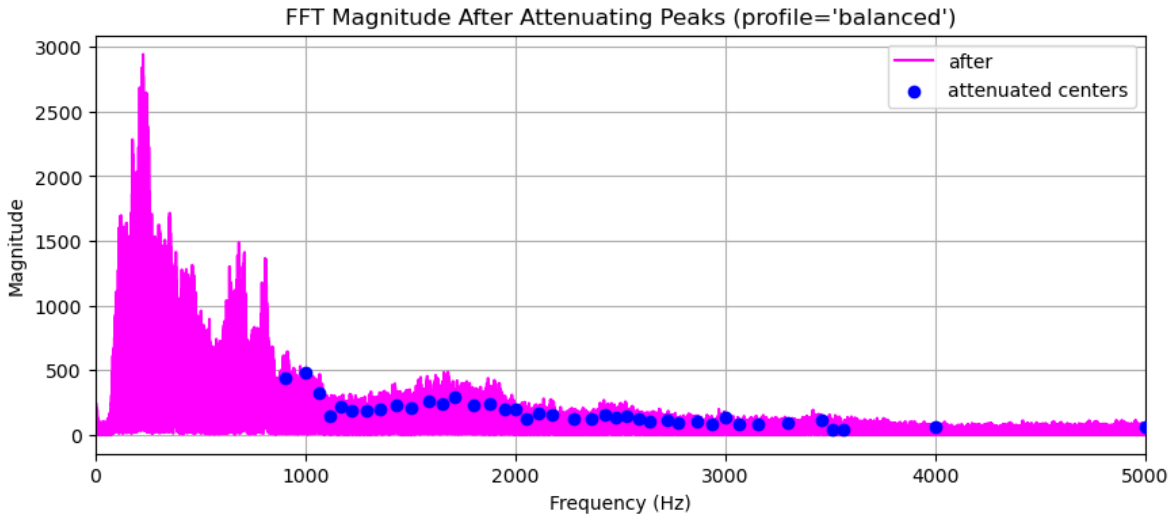

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\10_1_sample_very_conservative
Processing with profile: conservative
```



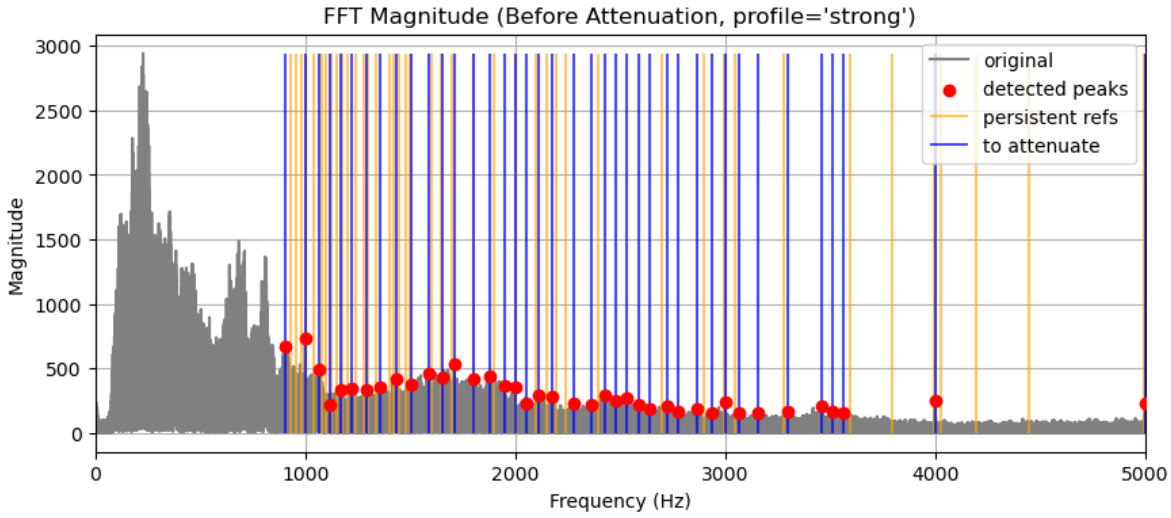


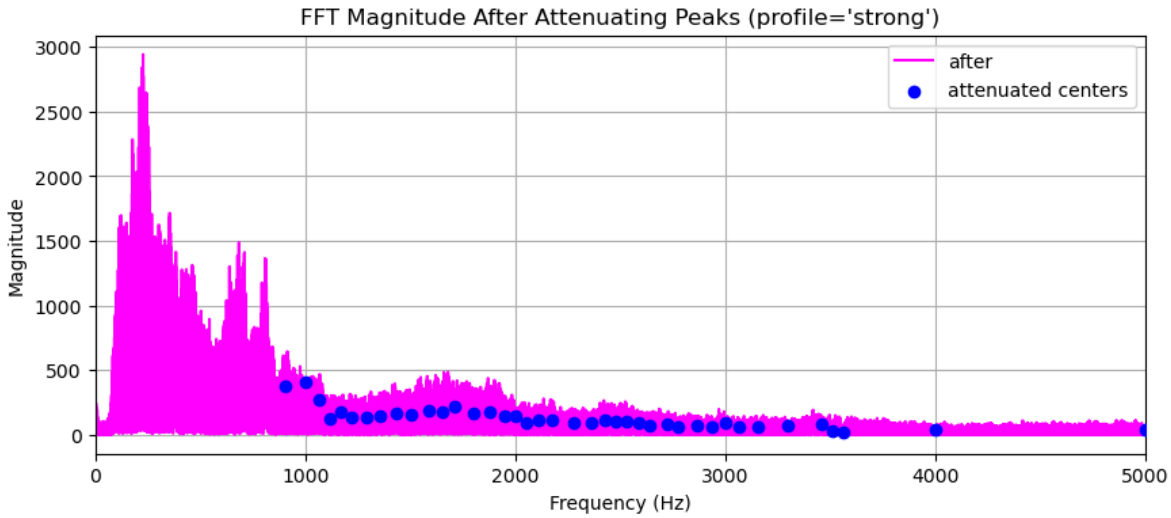

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\10_1_sample_conservative.wav
Processing with profile: balanced
```



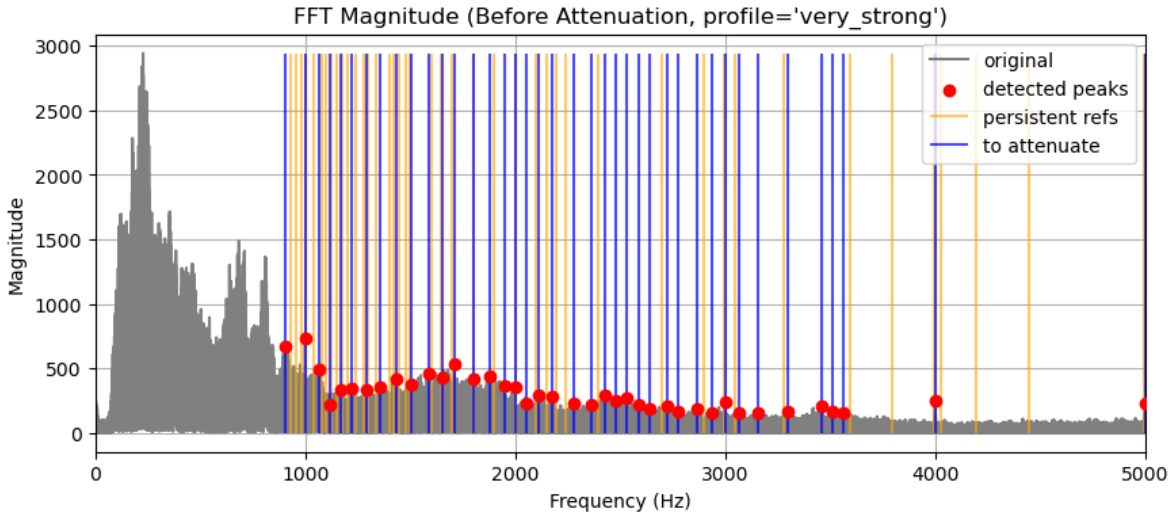


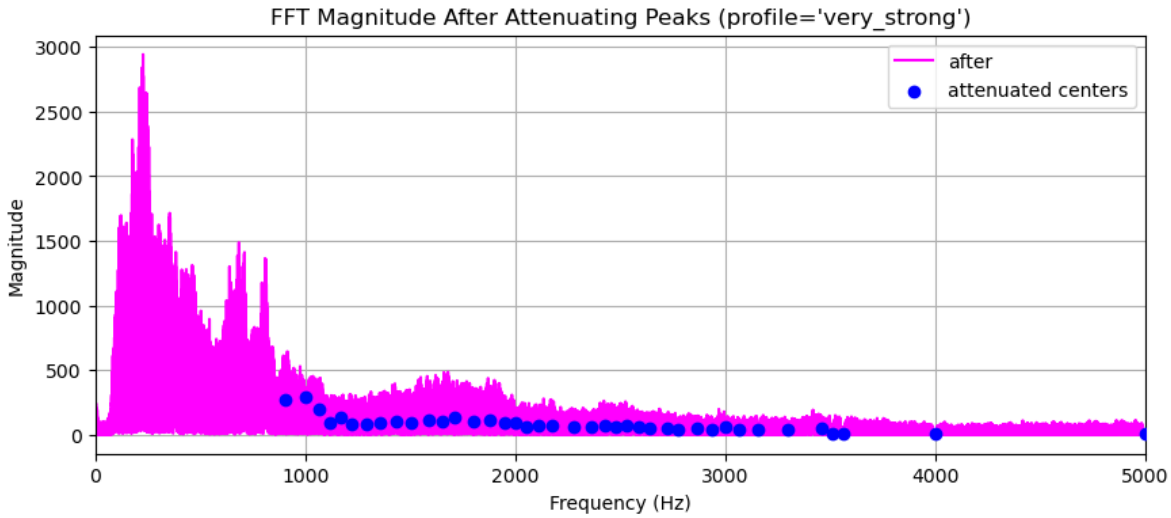

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\10_1_sample_balanced.wav
Processing with profile: strong
```



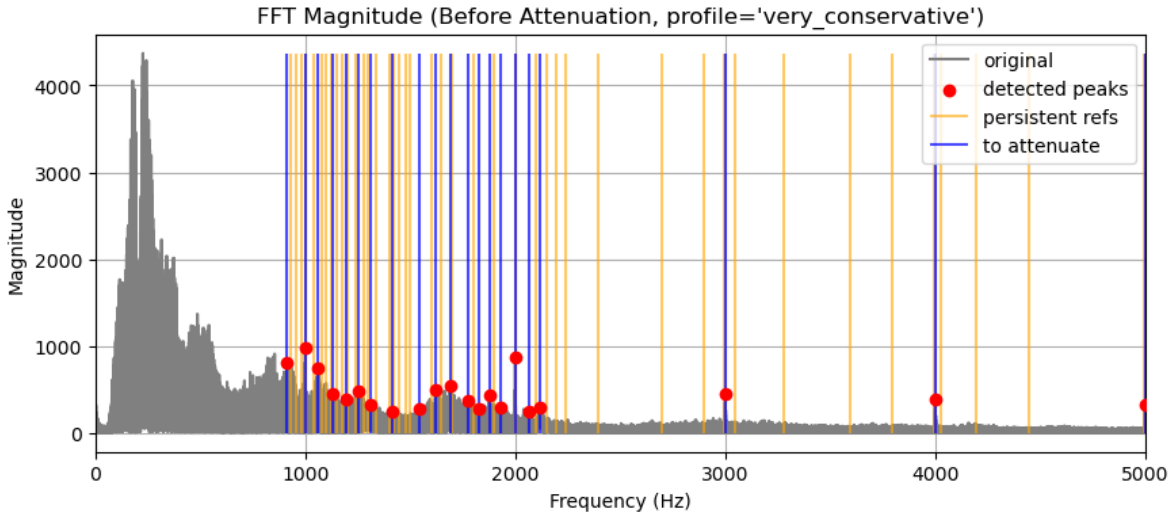


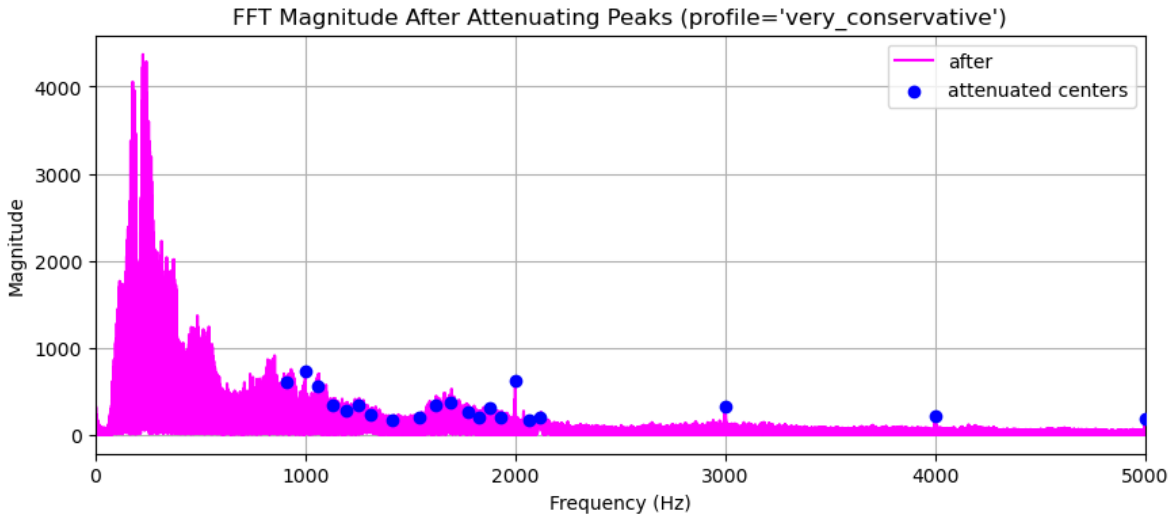

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\10_1_sample_strong.wav
Processing with profile: very_strong
```



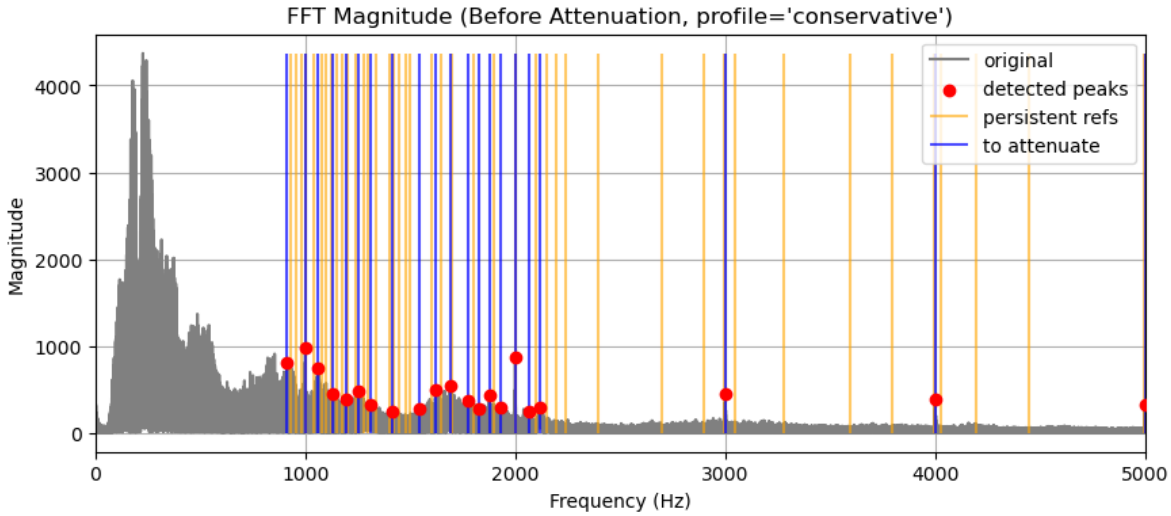


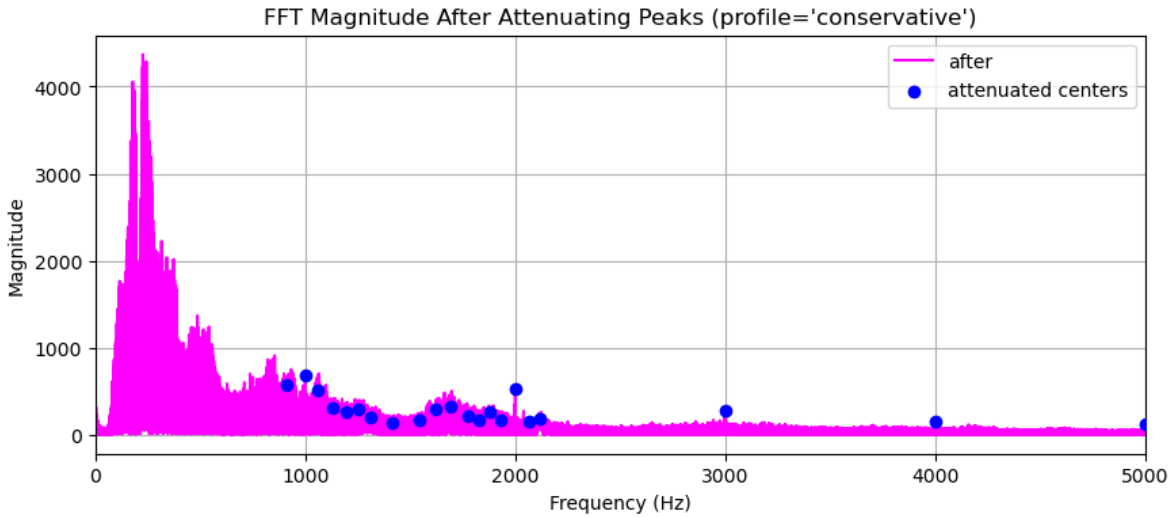

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\10_1_sample_very_strong.wav
Processing with profile: very_conservative
```



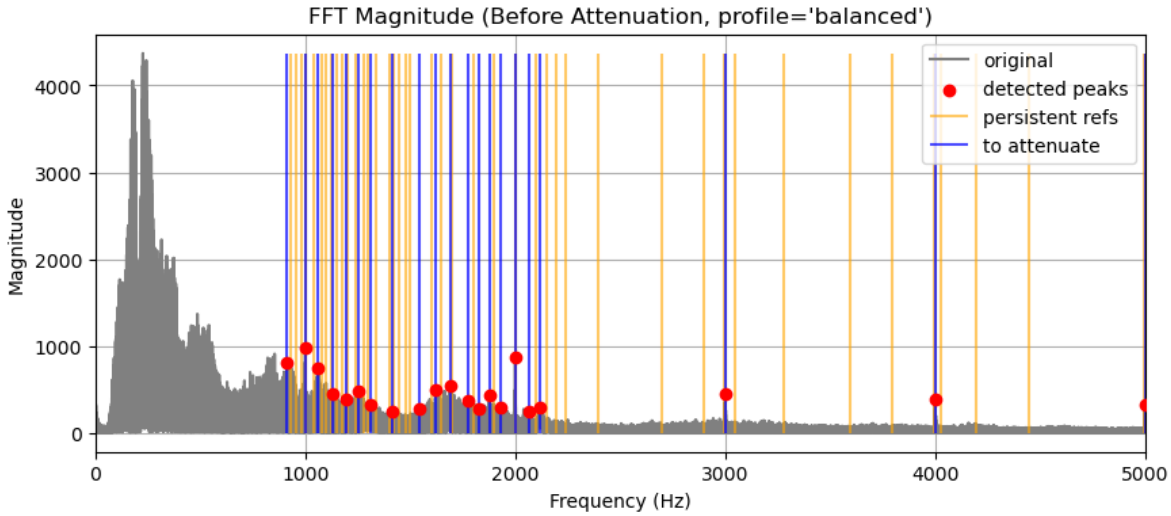


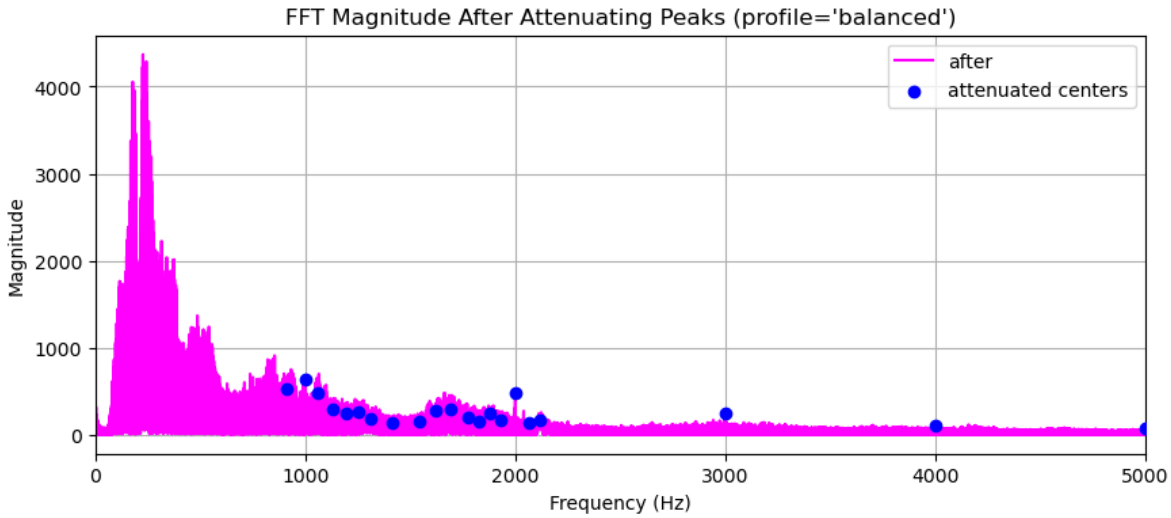

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\13_1_sample_very_conservative
Processing with profile: conservative
```



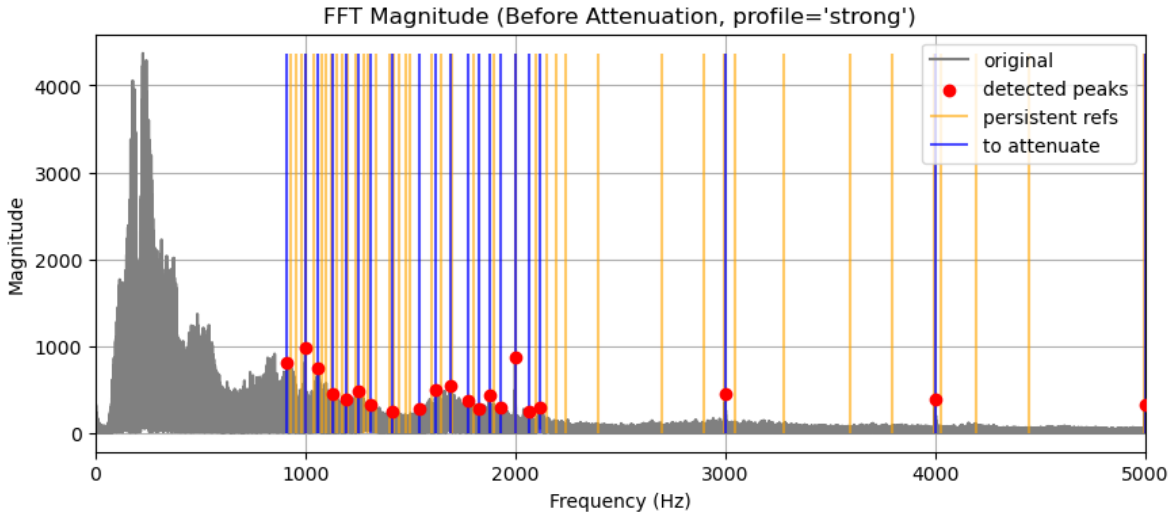


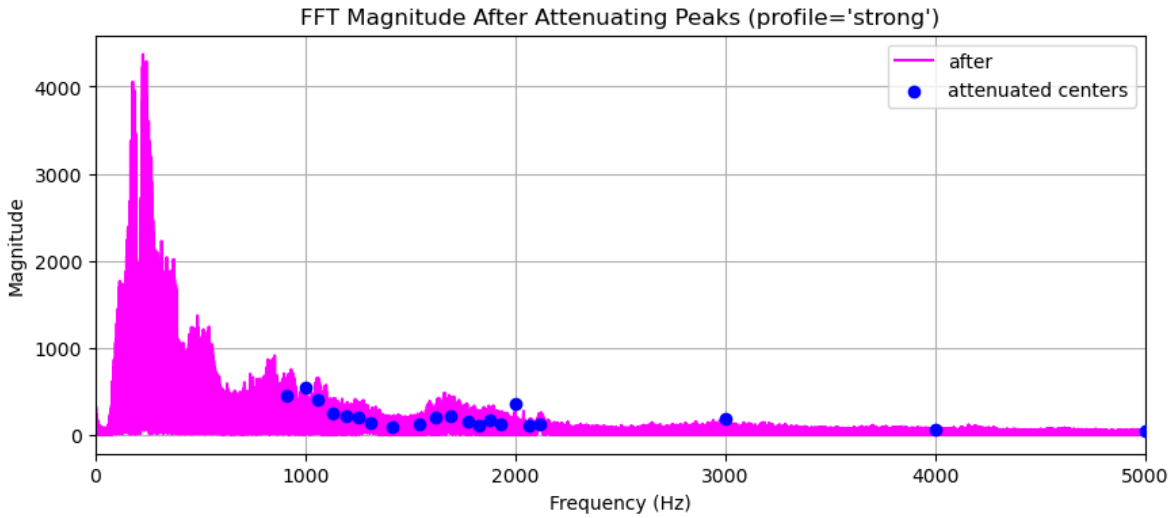

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\13_1_sample_conservative.wav
Processing with profile: balanced
```



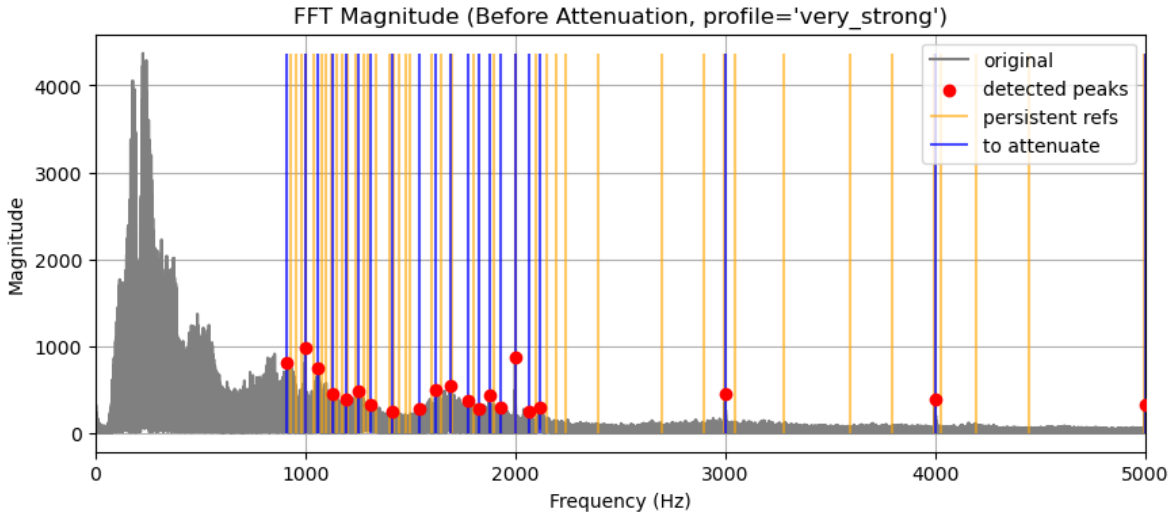


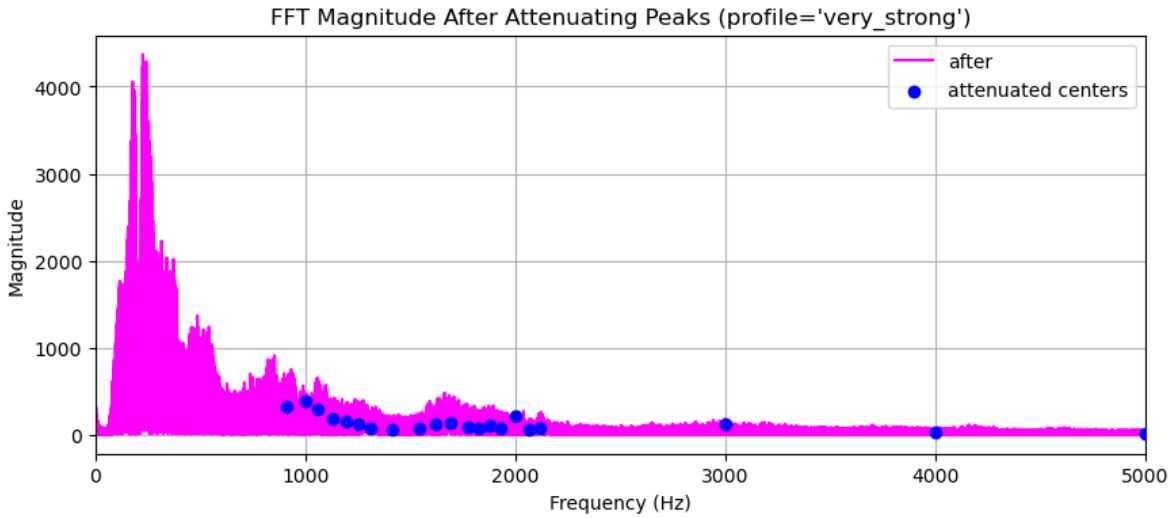

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\13_1_sample_balanced.wav
Processing with profile: strong
```



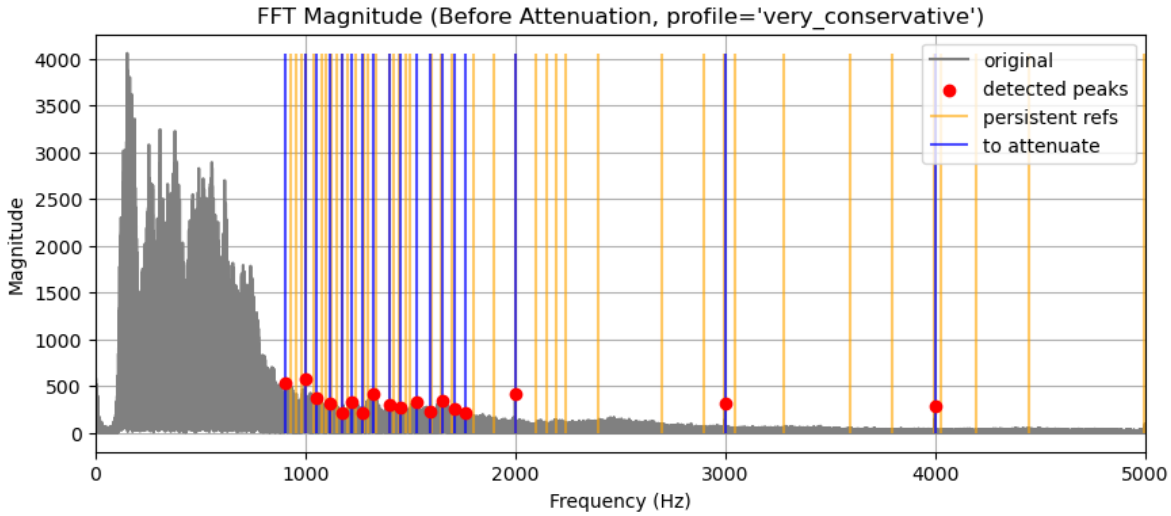


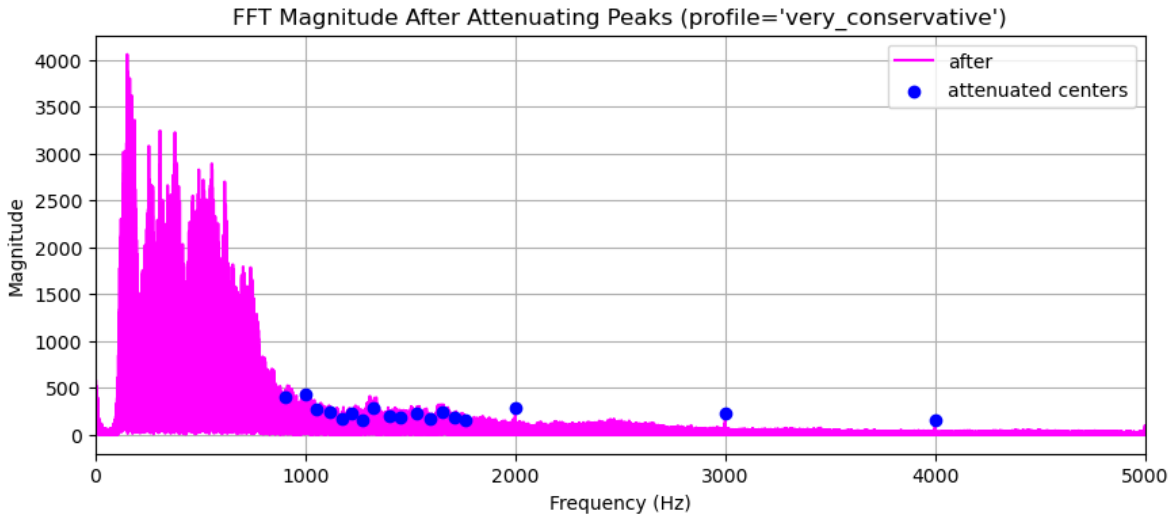

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\13_1_sample_strong.wav
Processing with profile: very_strong
```



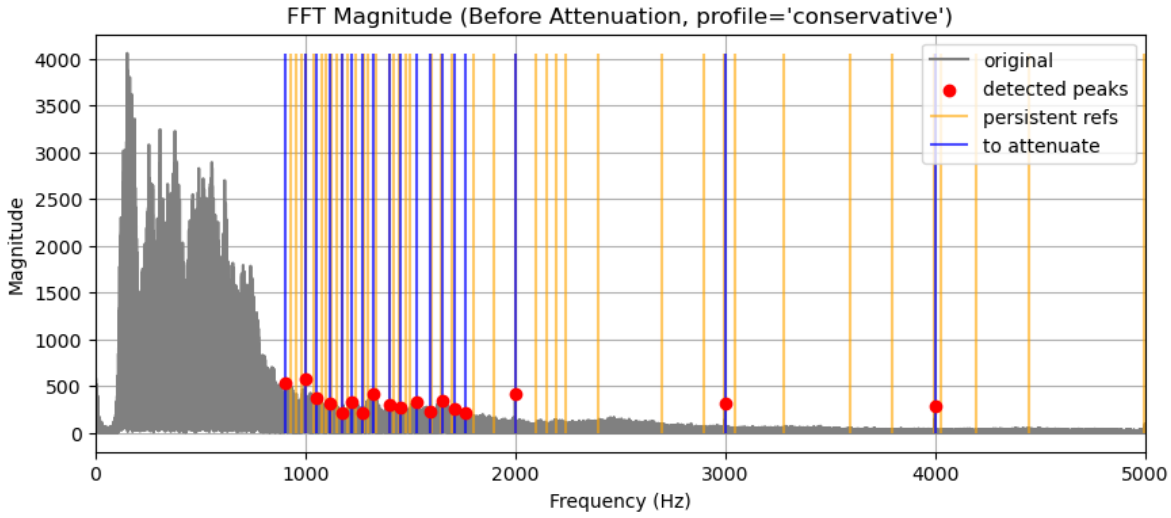


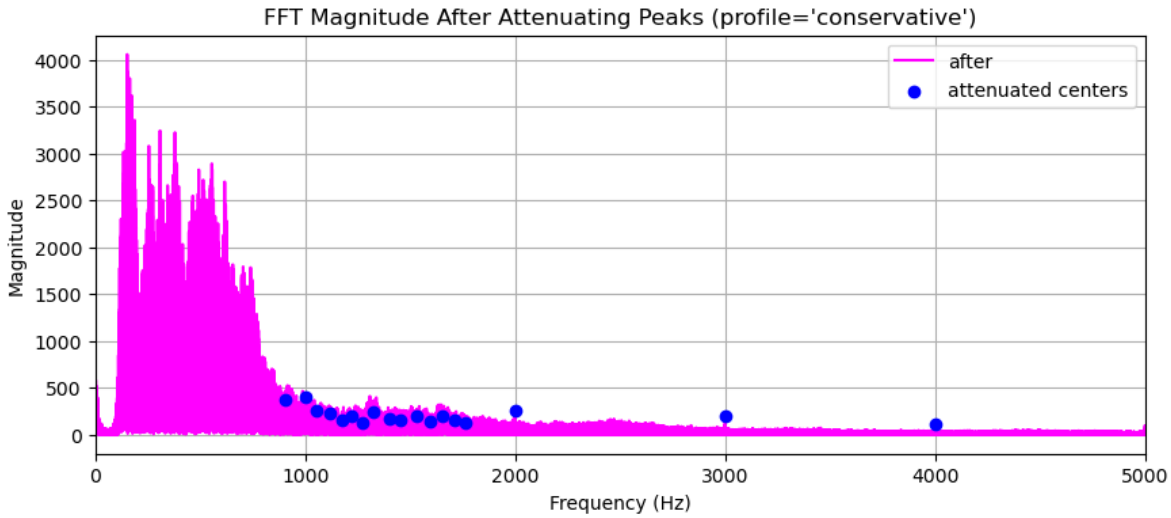

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\13_1_sample_very_strong.wav
Processing with profile: very_conservative
```



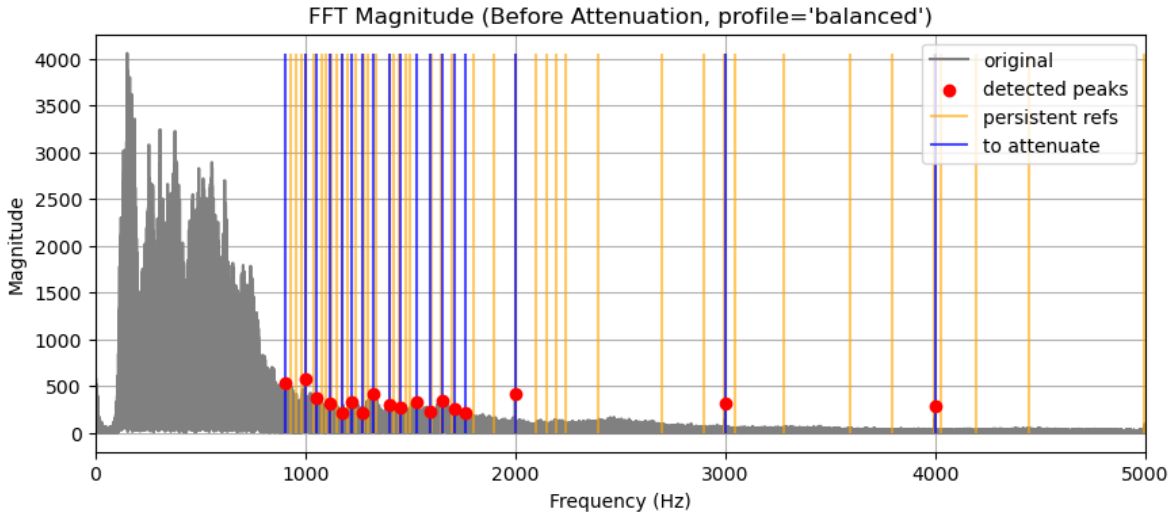


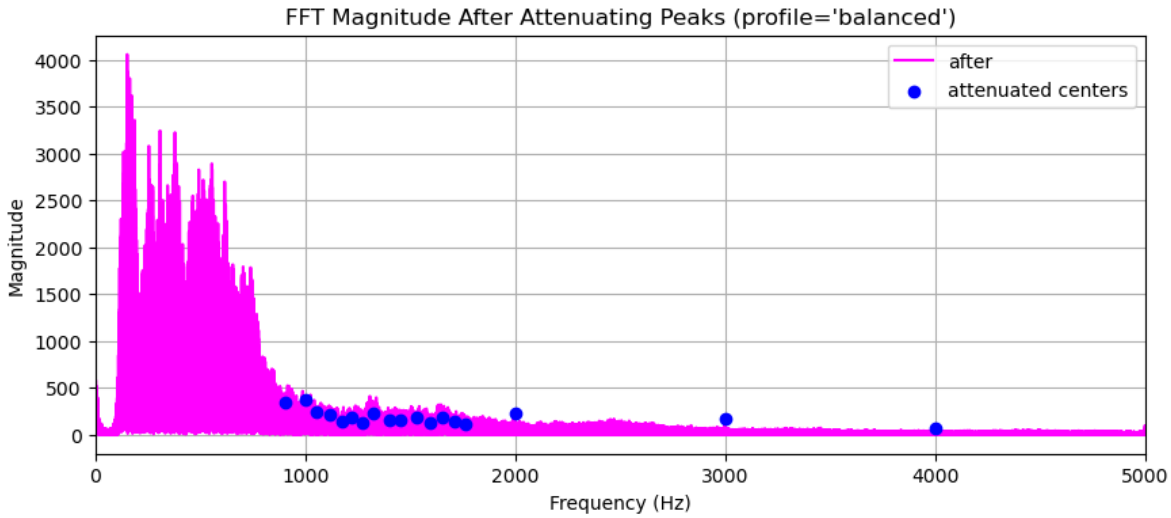

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\16_1_sample_very_conservative
Processing with profile: conservative
```



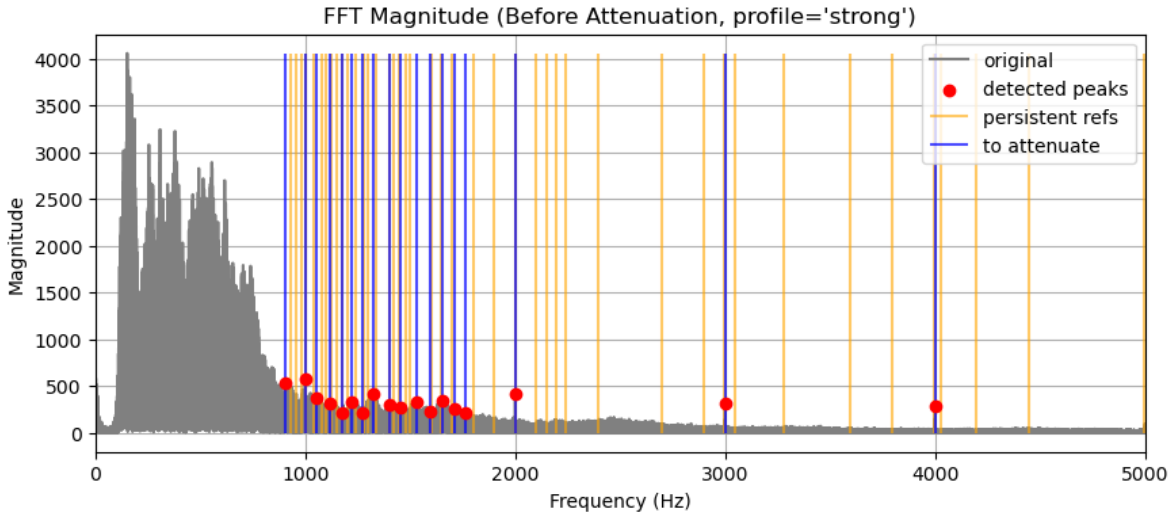


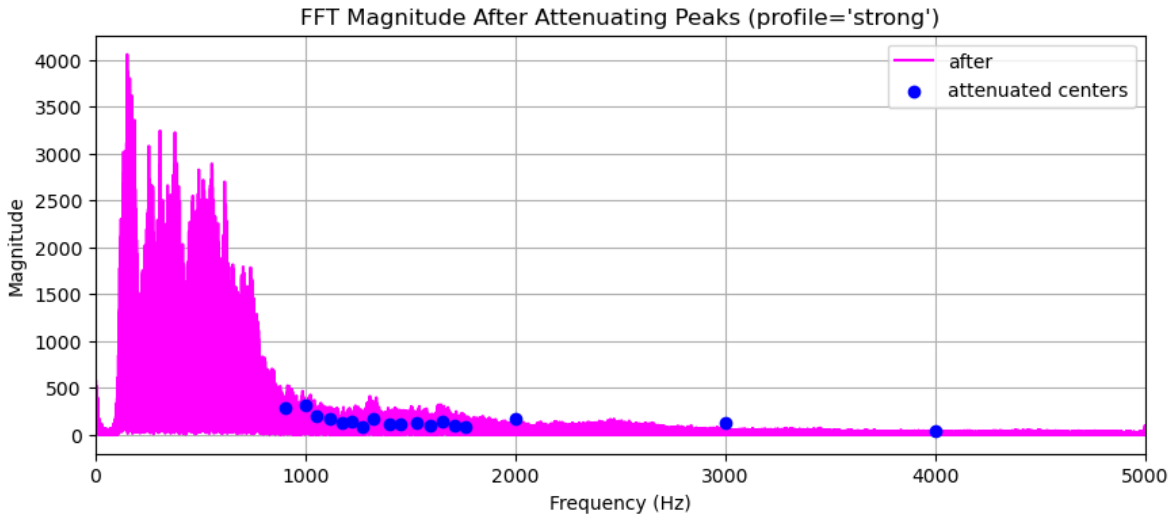

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\16_1_sample_conservative.wav
Processing with profile: balanced
```



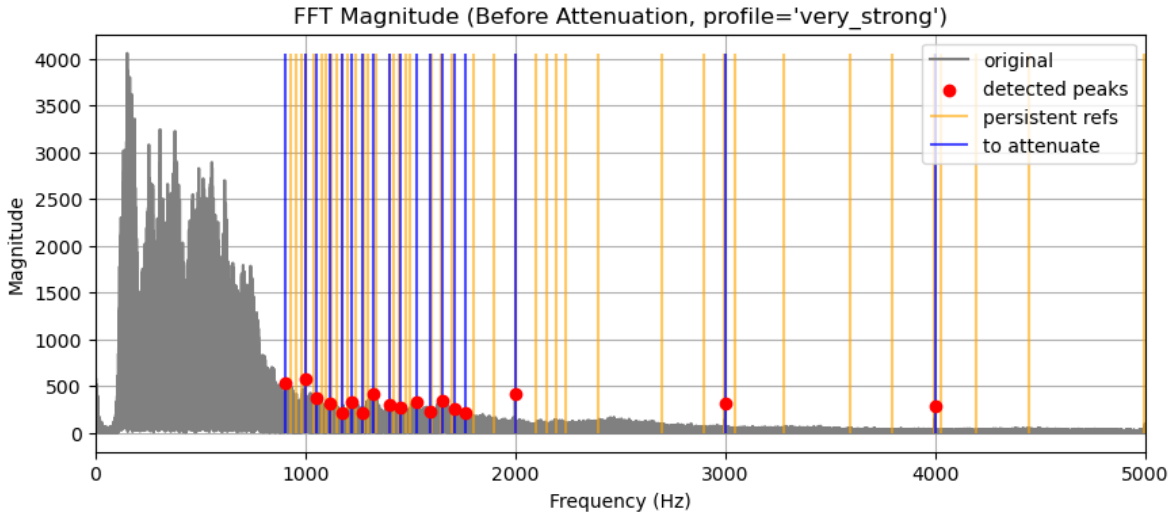


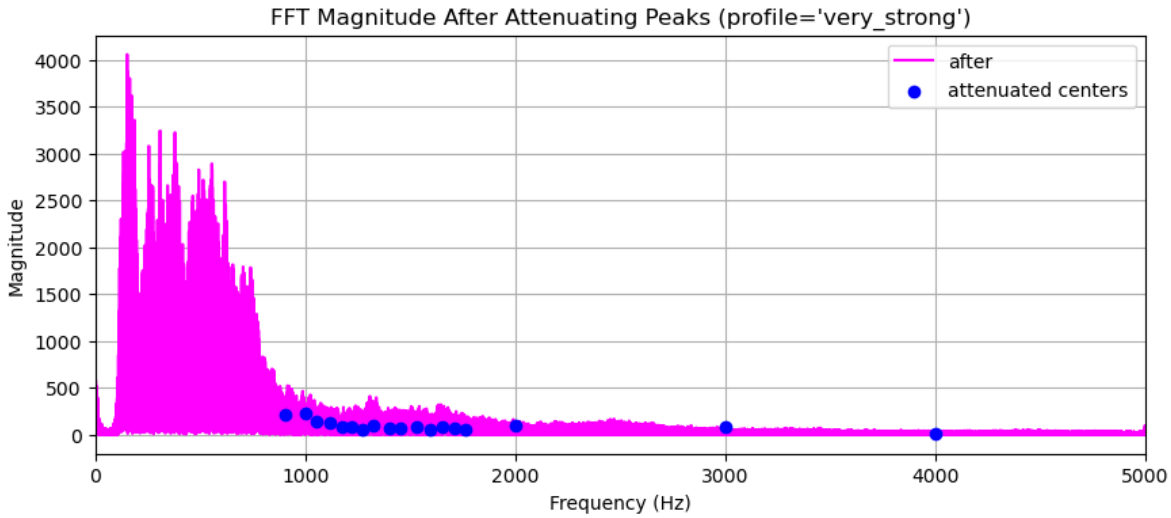

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\16_1_sample_balanced.wav
Processing with profile: strong
```



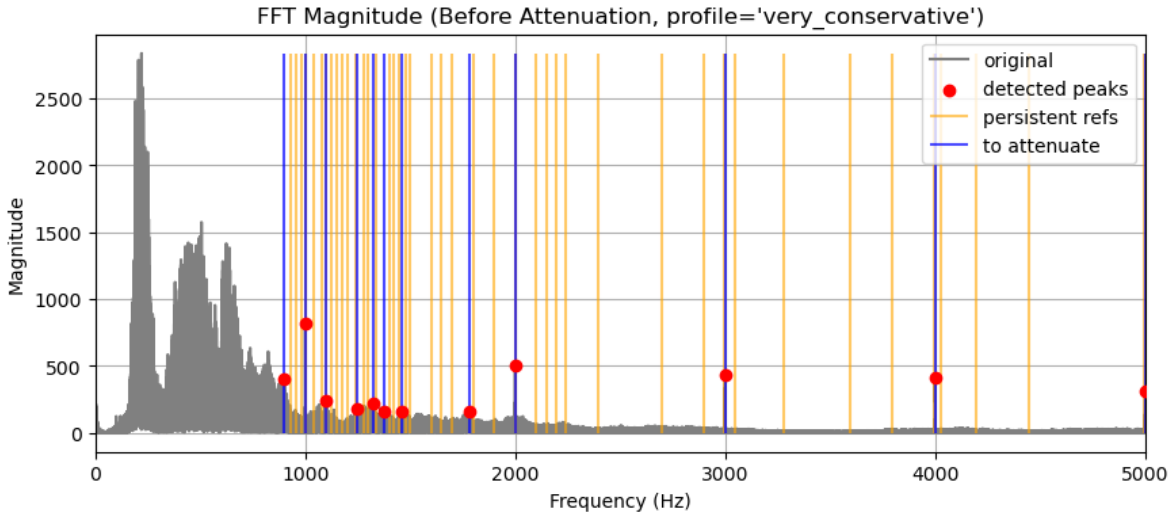


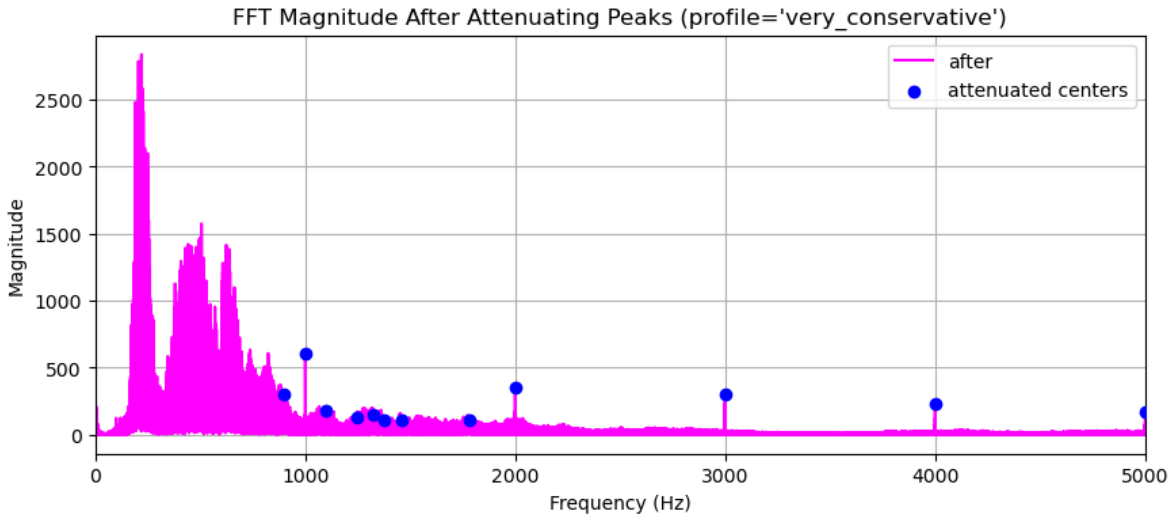

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\16_1_sample_strong.wav
Processing with profile: very_strong
```



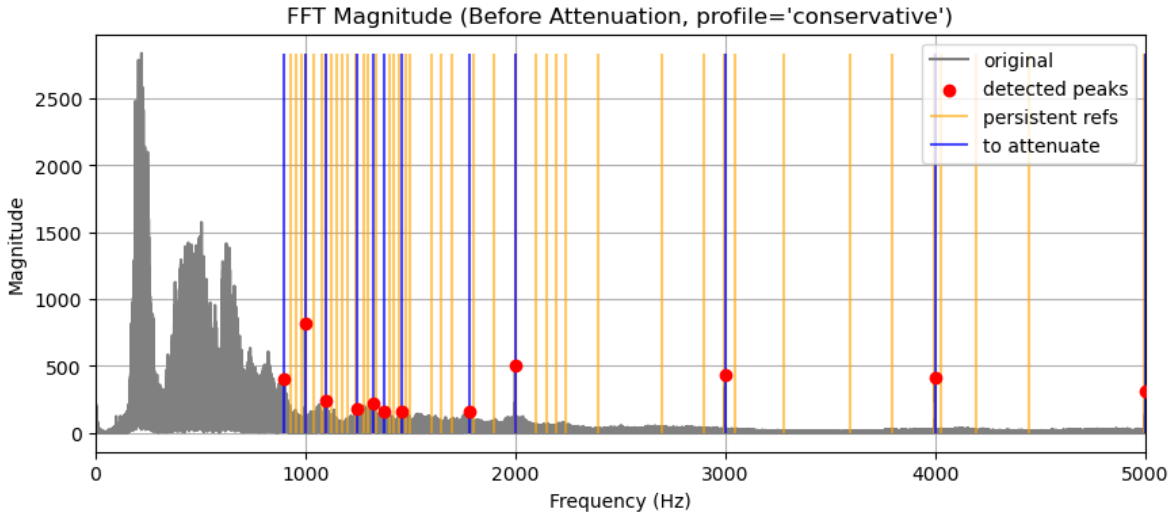


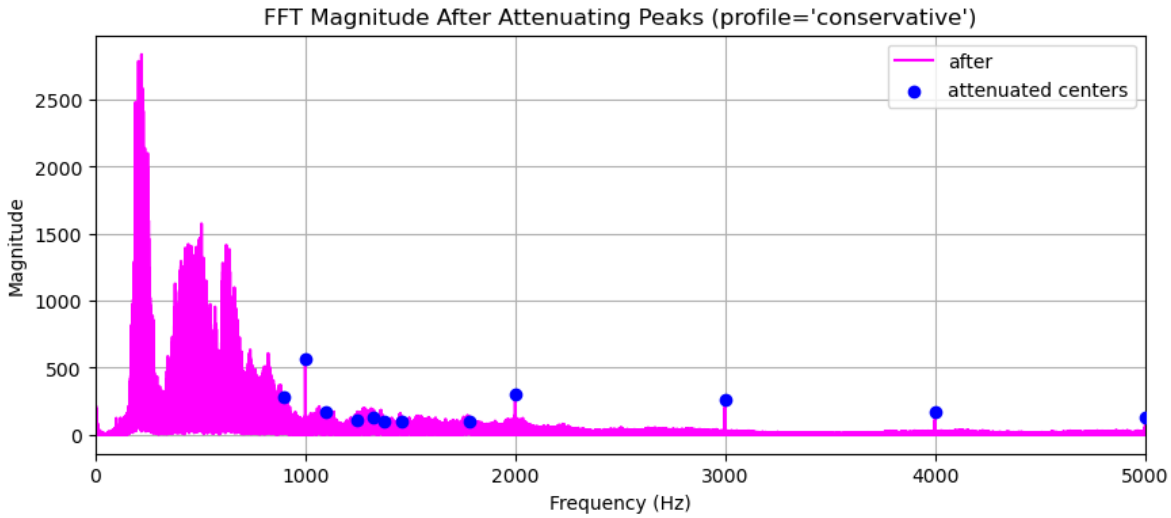

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\16_1_sample_very_strong.wav
Processing with profile: very_conservative
```



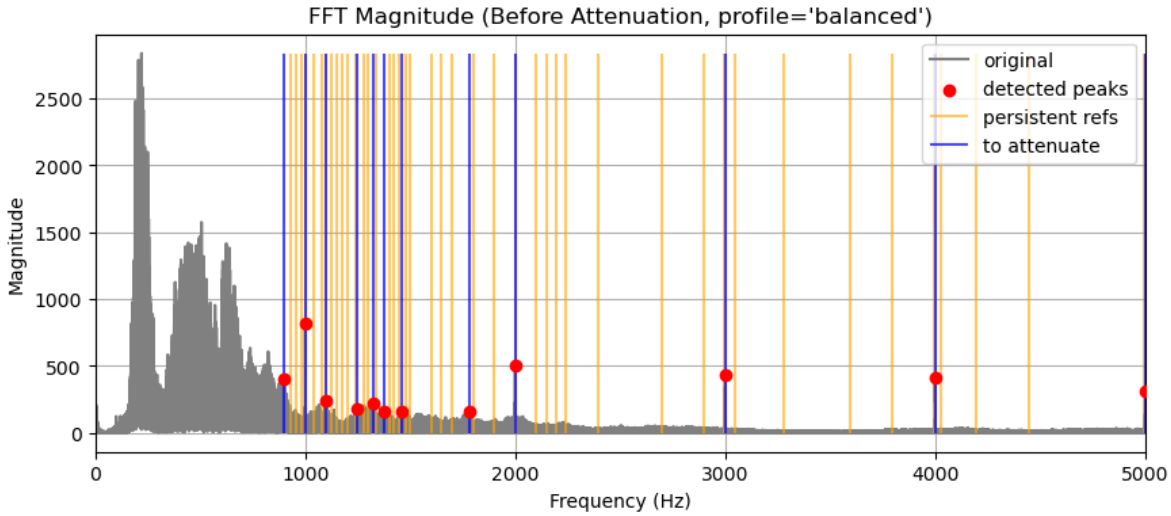


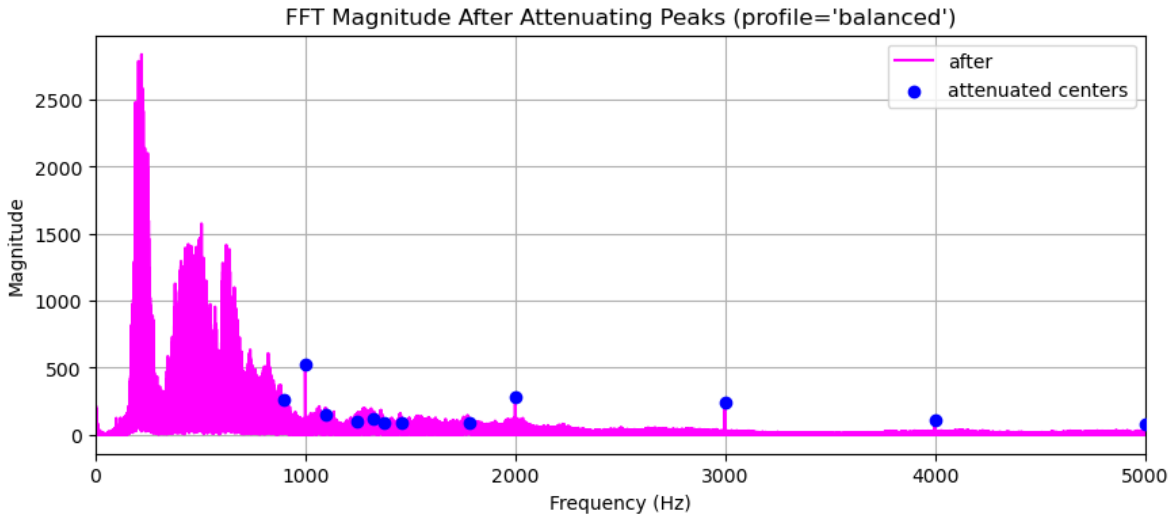

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\3_1_sample_very_conservative.w
Processing with profile: conservative
```



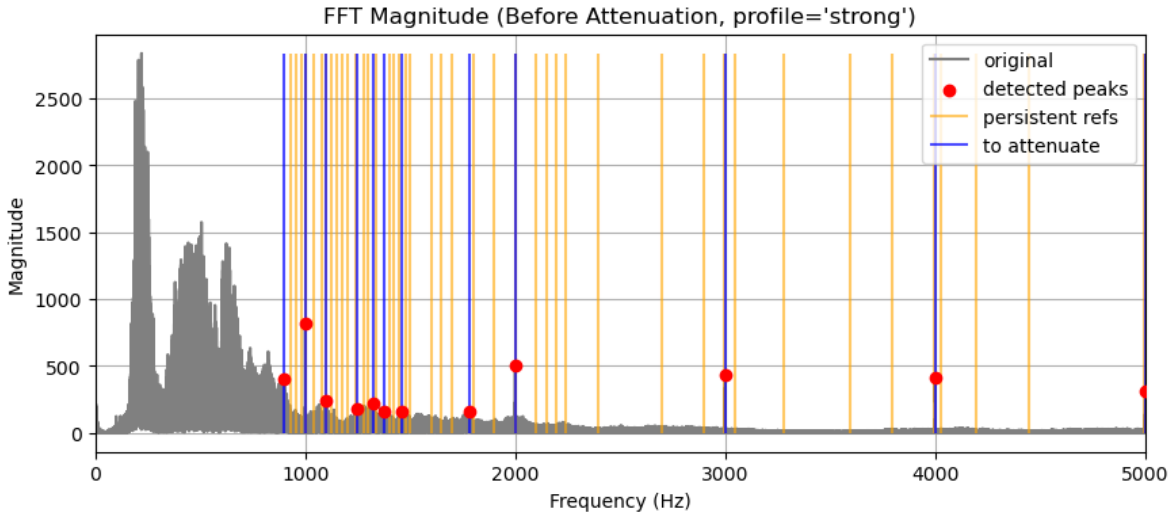


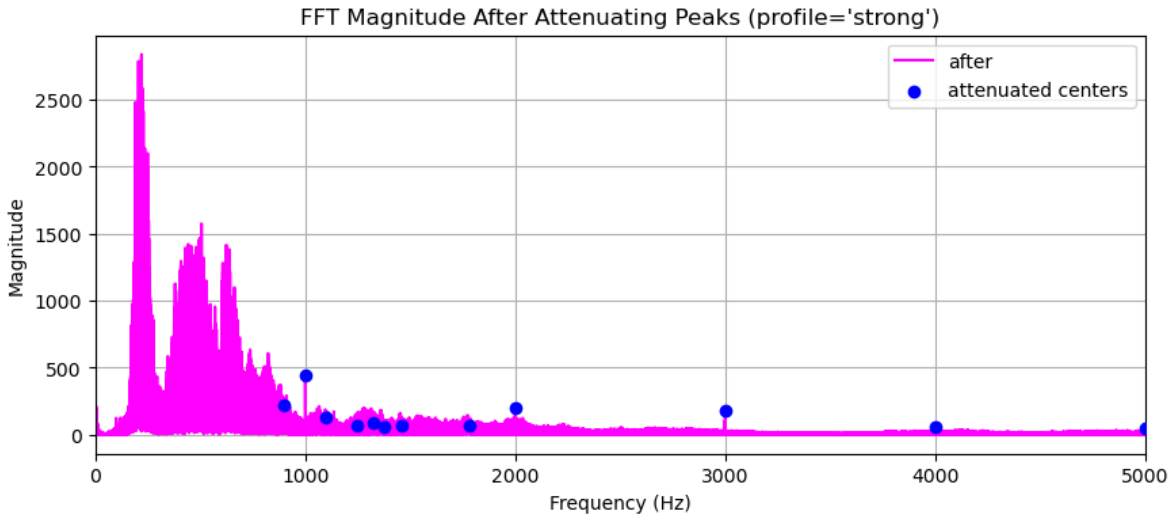

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\3_1_sample_conservative.wav
Processing with profile: balanced
```



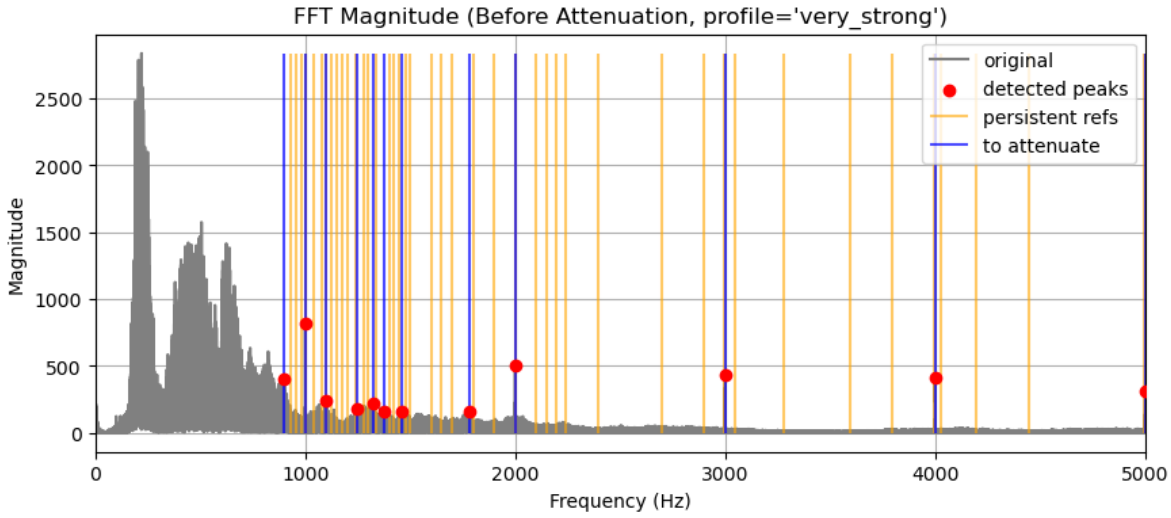


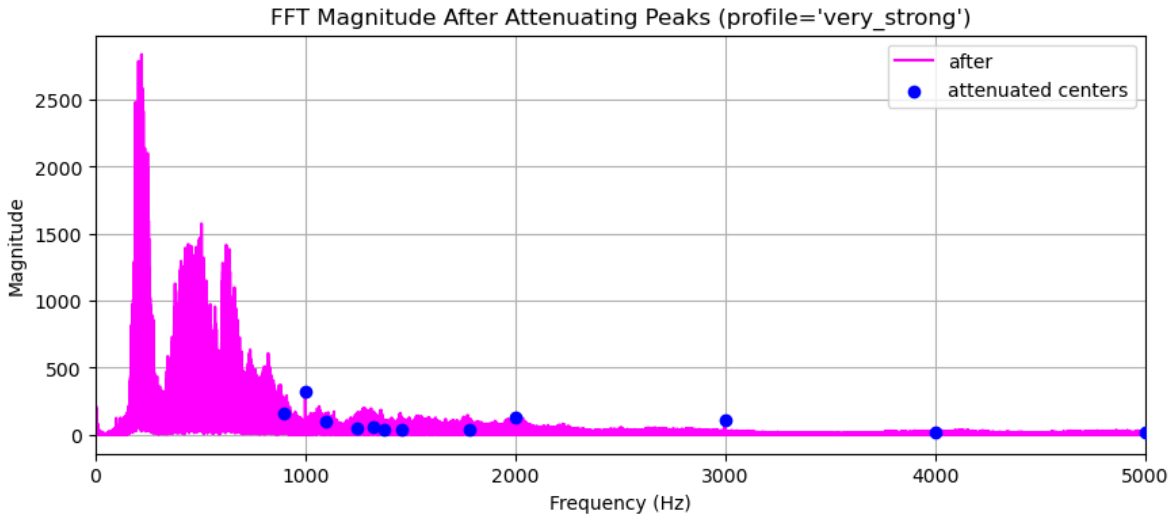

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\3_1_sample_balanced.wav
Processing with profile: strong
```



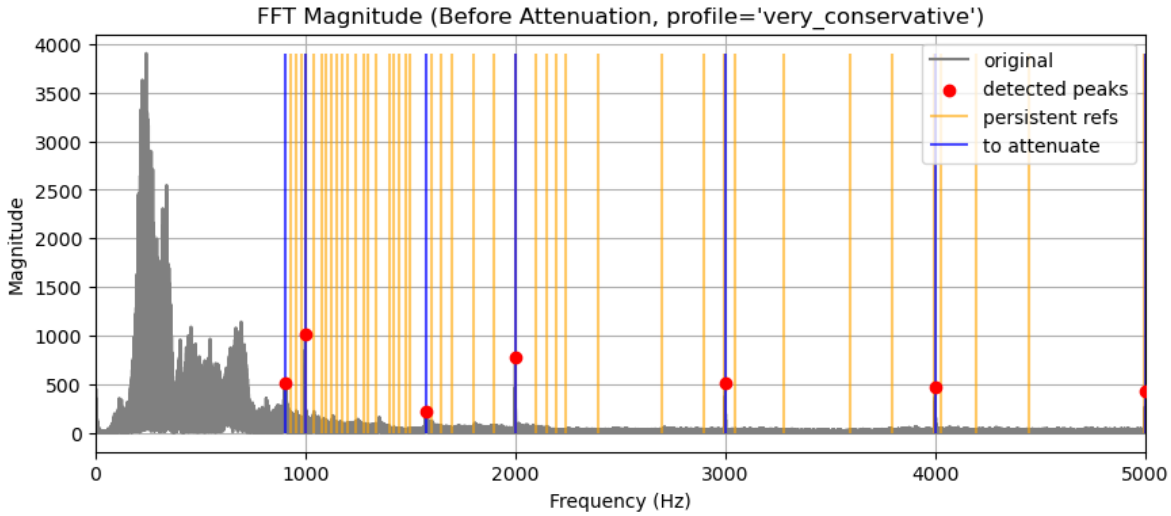


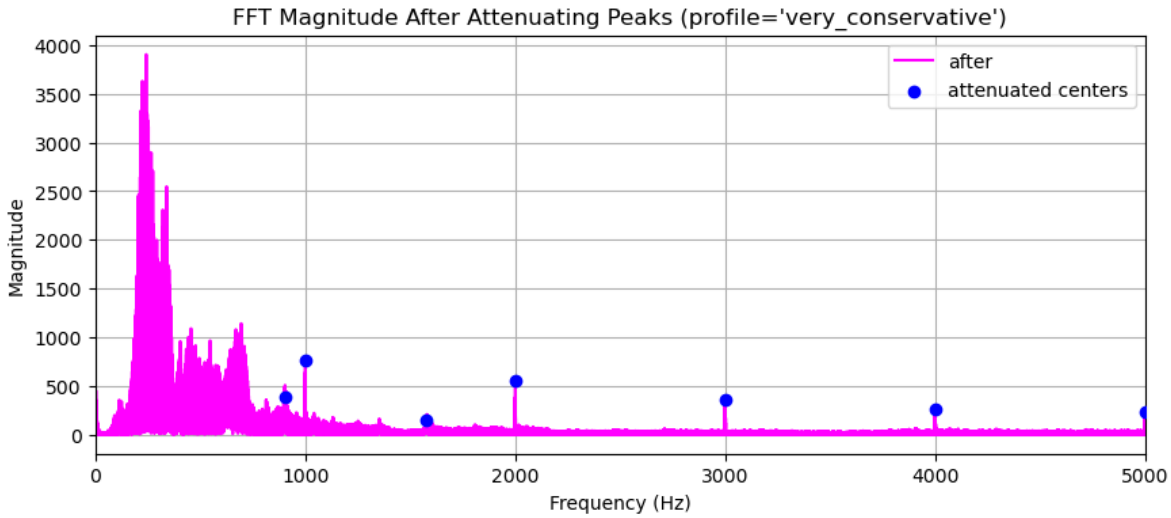

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\3_1_sample_strong.wav
Processing with profile: very_strong
```



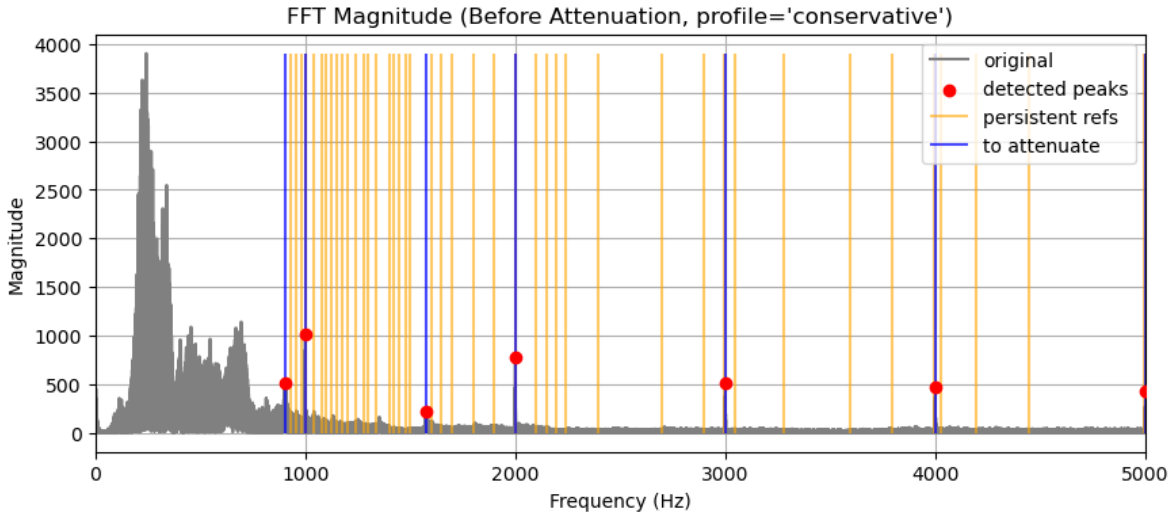


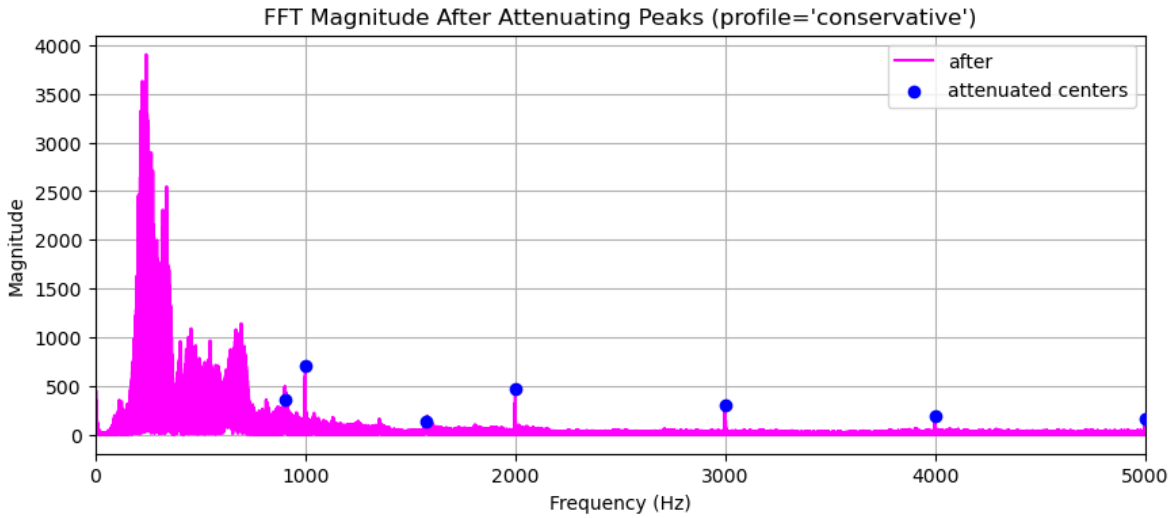

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\3_1_sample_very_strong.wav
Processing with profile: very_conservative
```



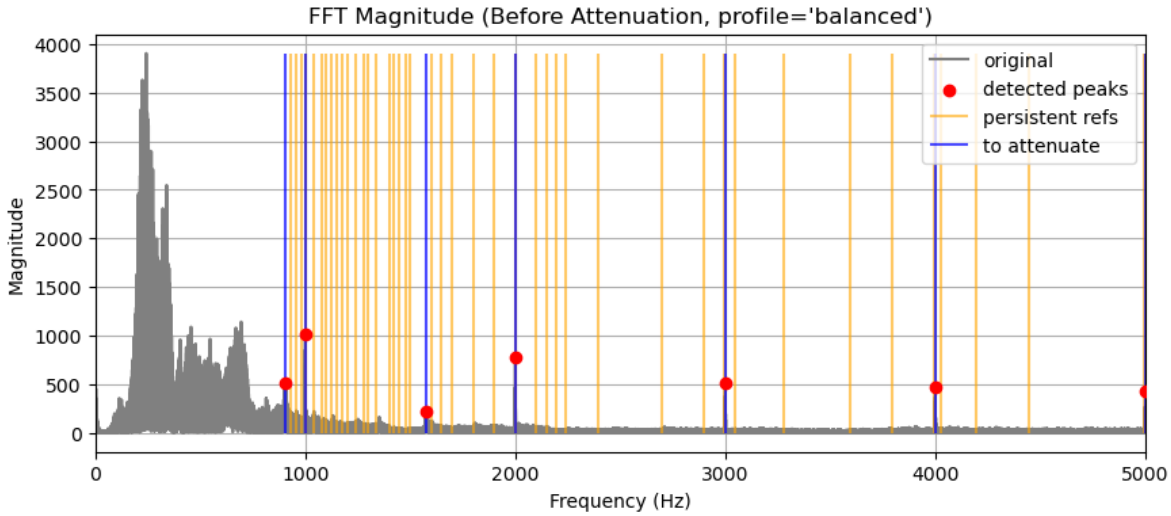


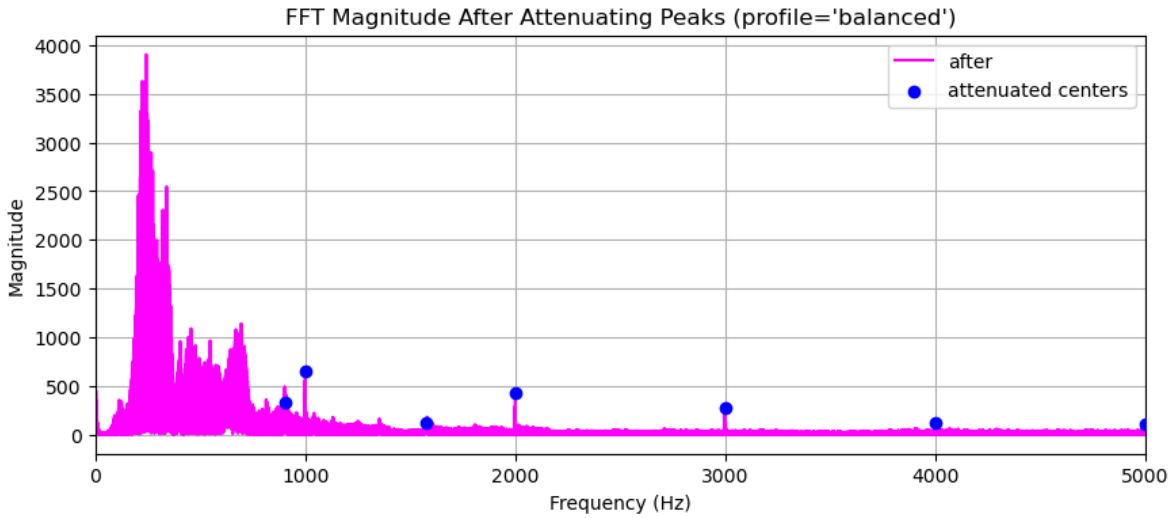

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\4_2_sample_very_conservative.w
Processing with profile: conservative
```



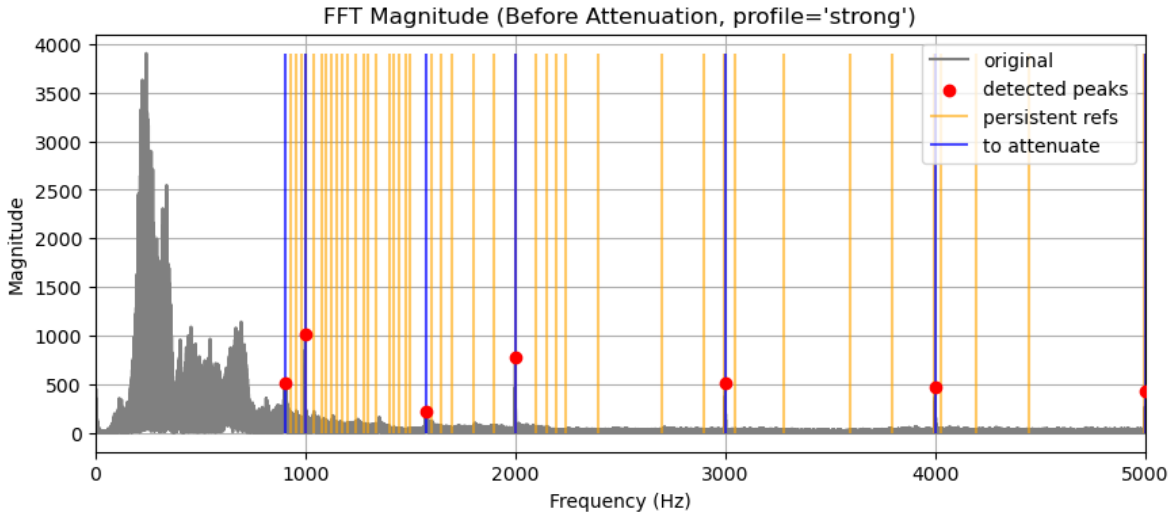


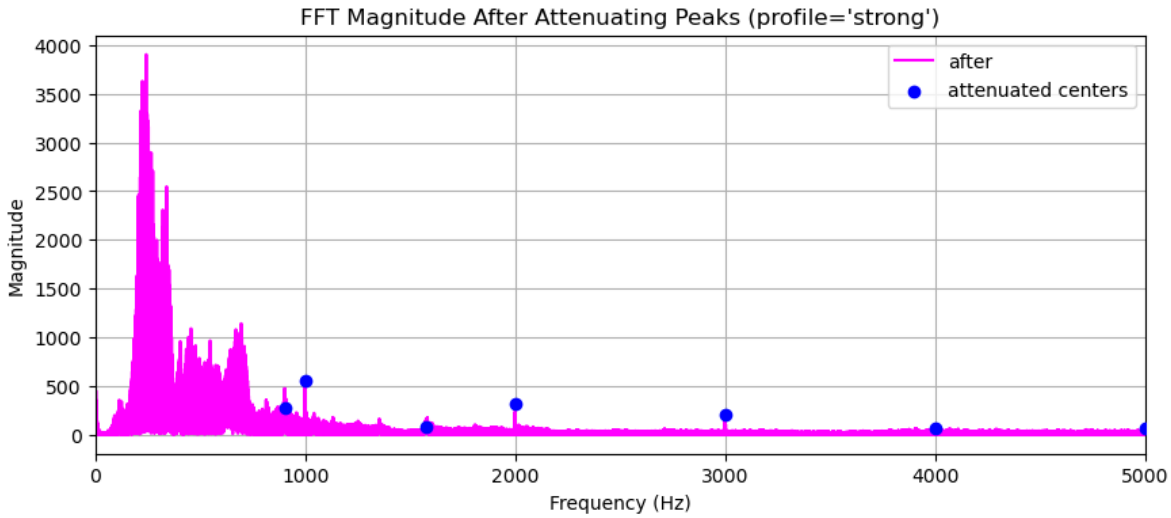

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\4_2_sample_conservative.wav
Processing with profile: balanced
```



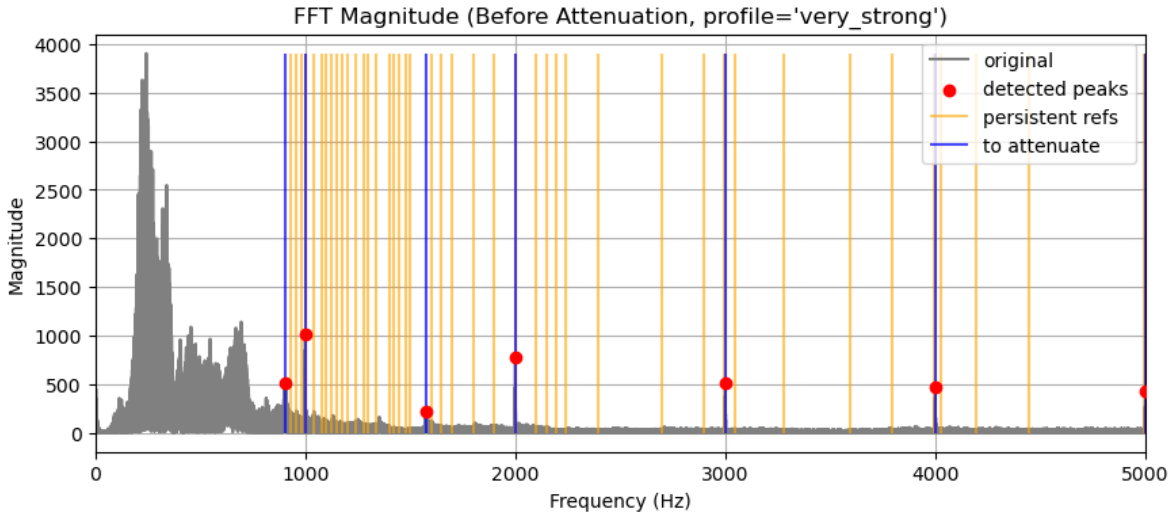


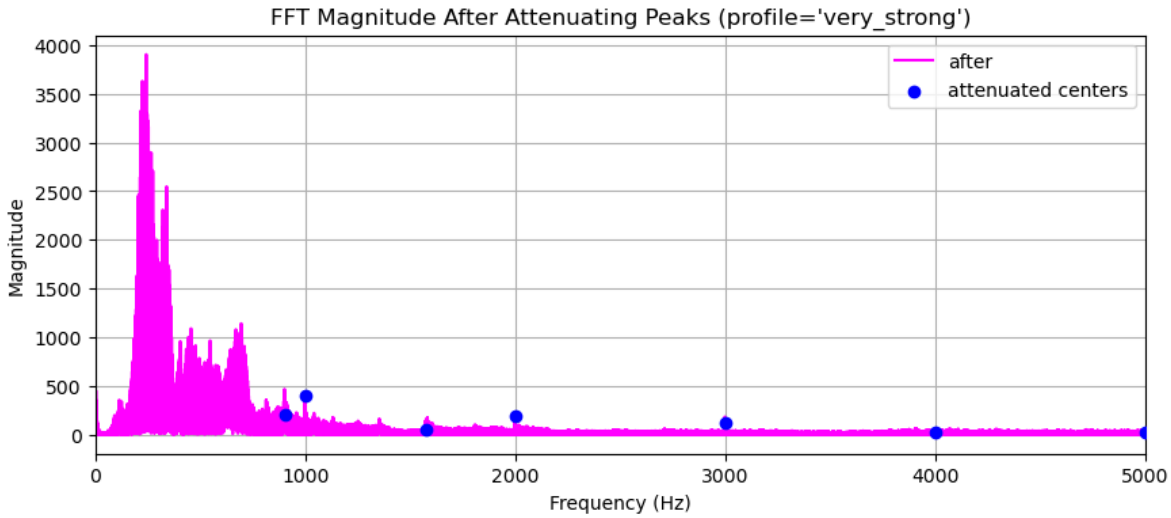

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\4_2_sample_balanced.wav
Processing with profile: strong
```



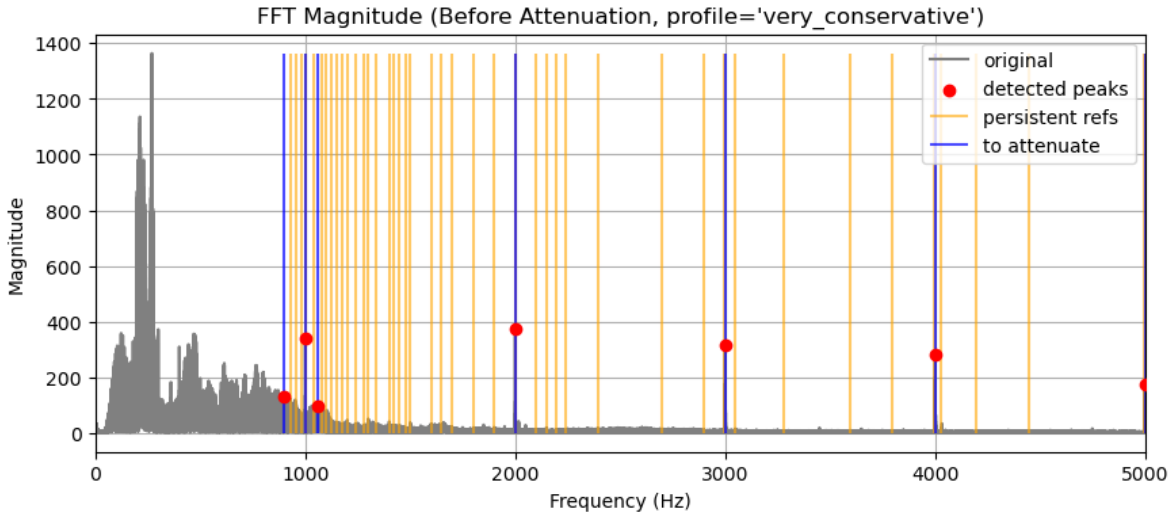


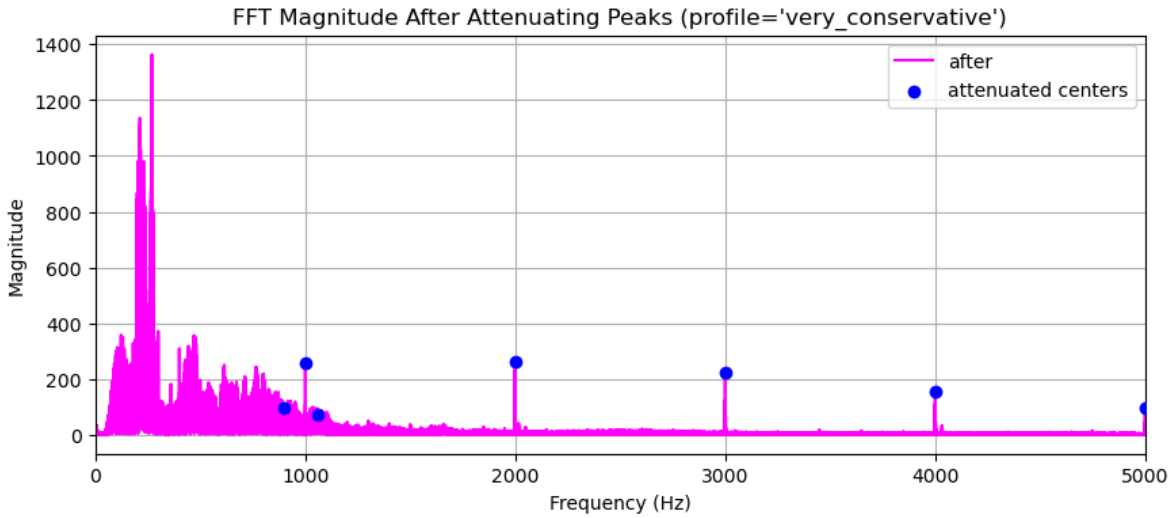

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\4_2_sample_strong.wav
Processing with profile: very_strong
```



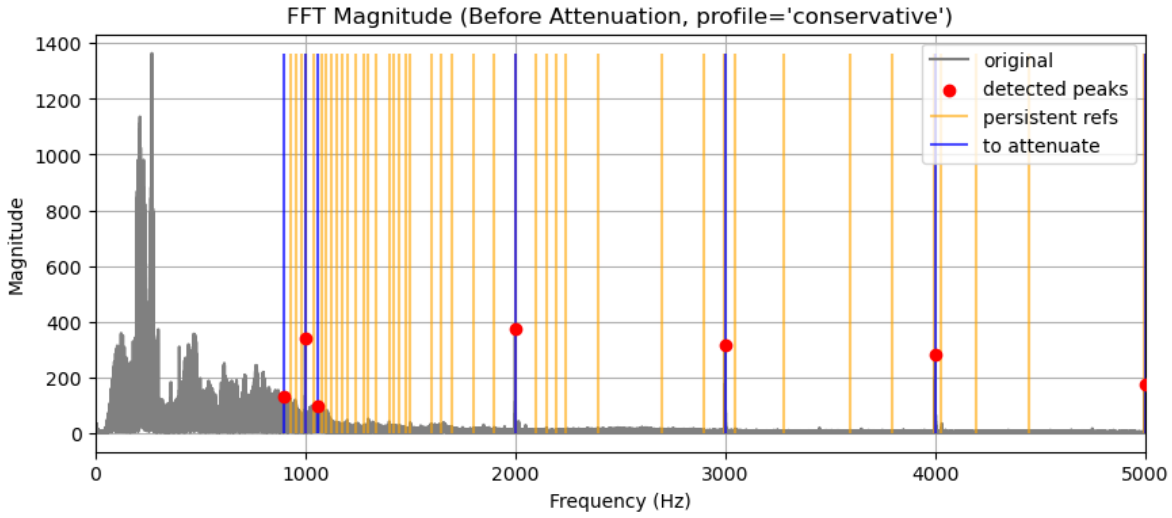


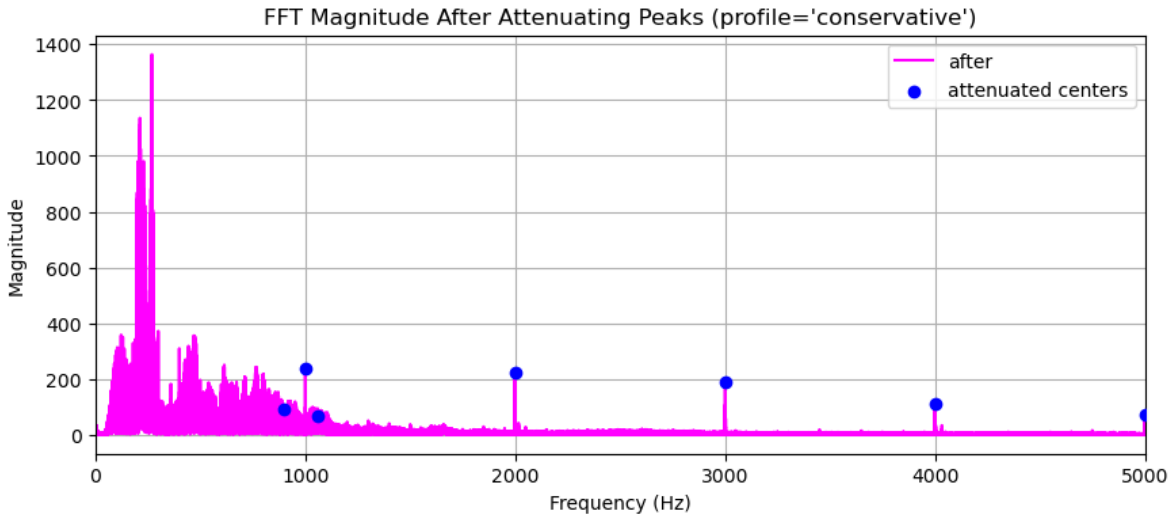

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\4_2_sample_very_strong.wav
Processing with profile: very_conservative
```



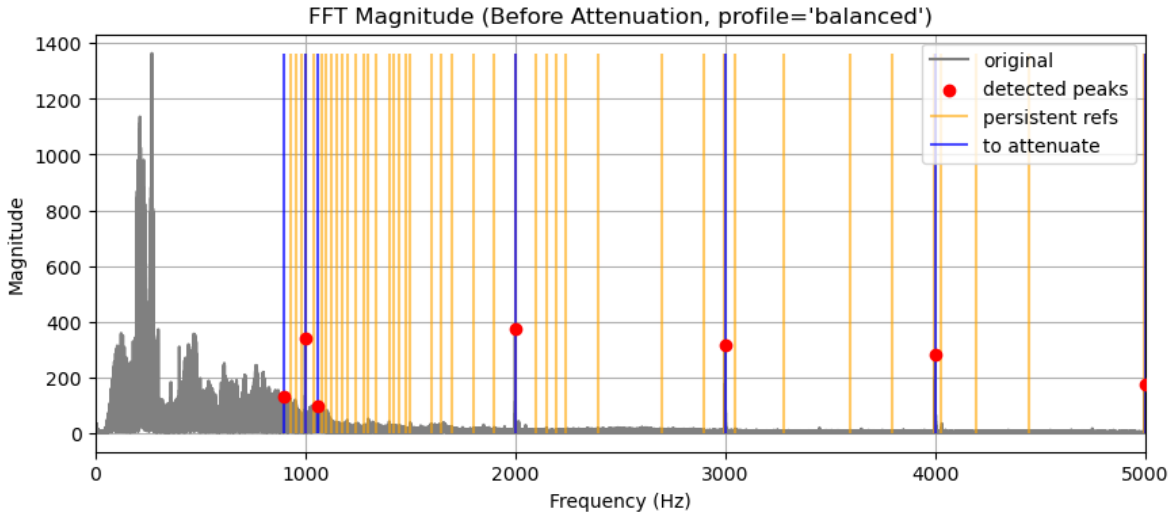


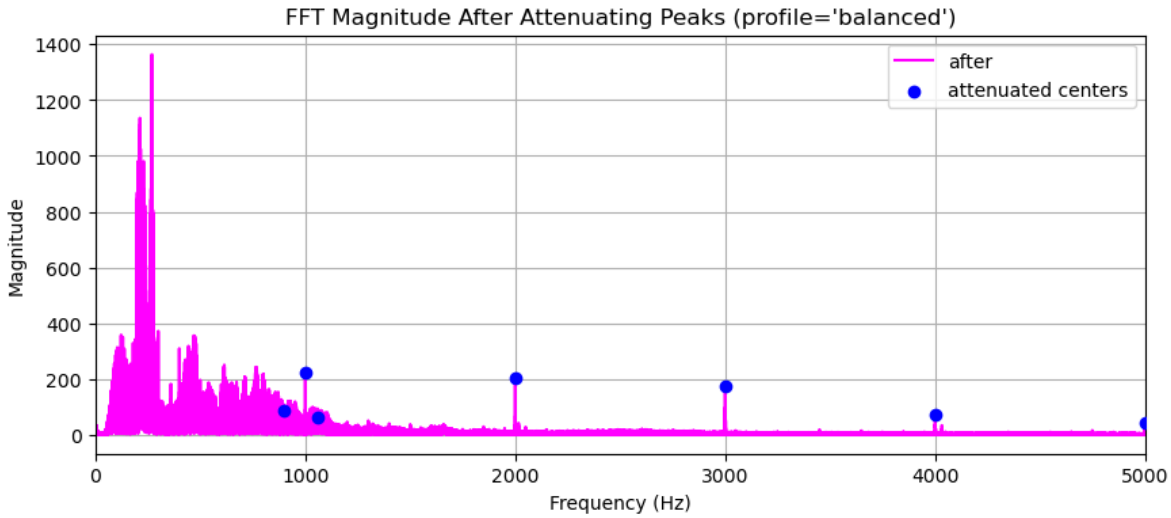

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\5_1_sample_very_conservative.w
Processing with profile: conservative
```



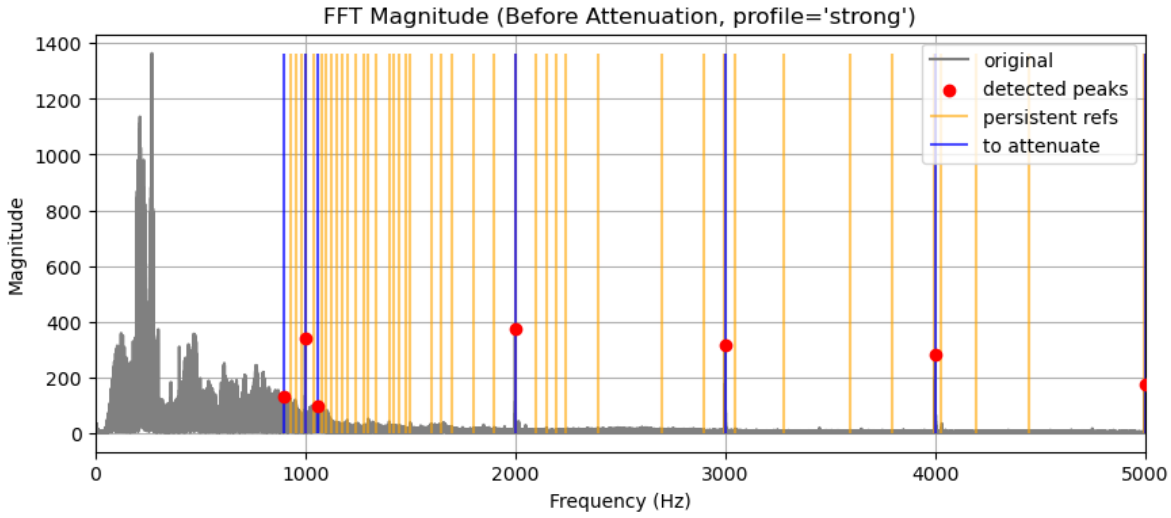


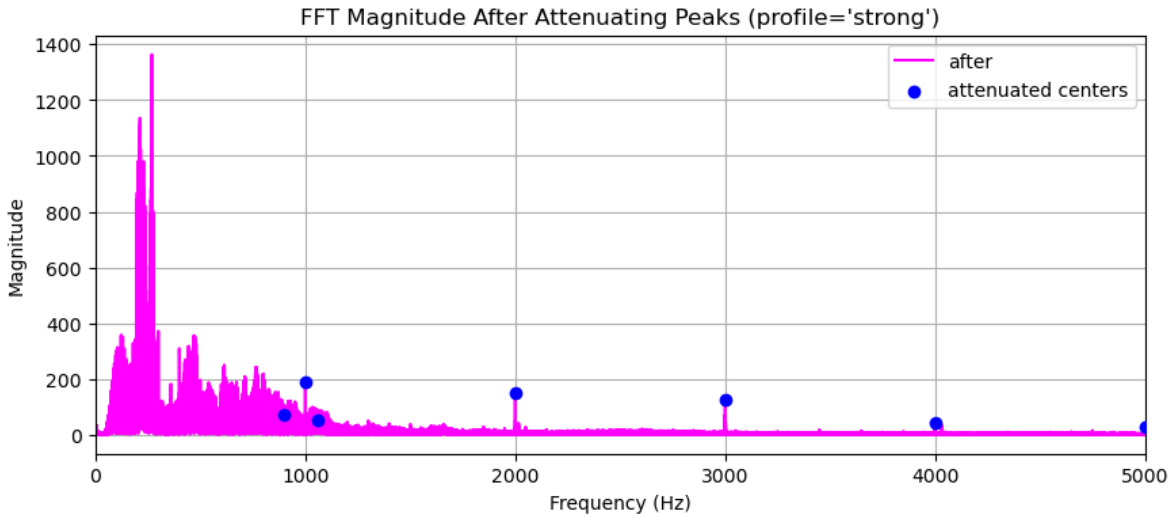

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\5_1_sample_conservative.wav
Processing with profile: balanced
```



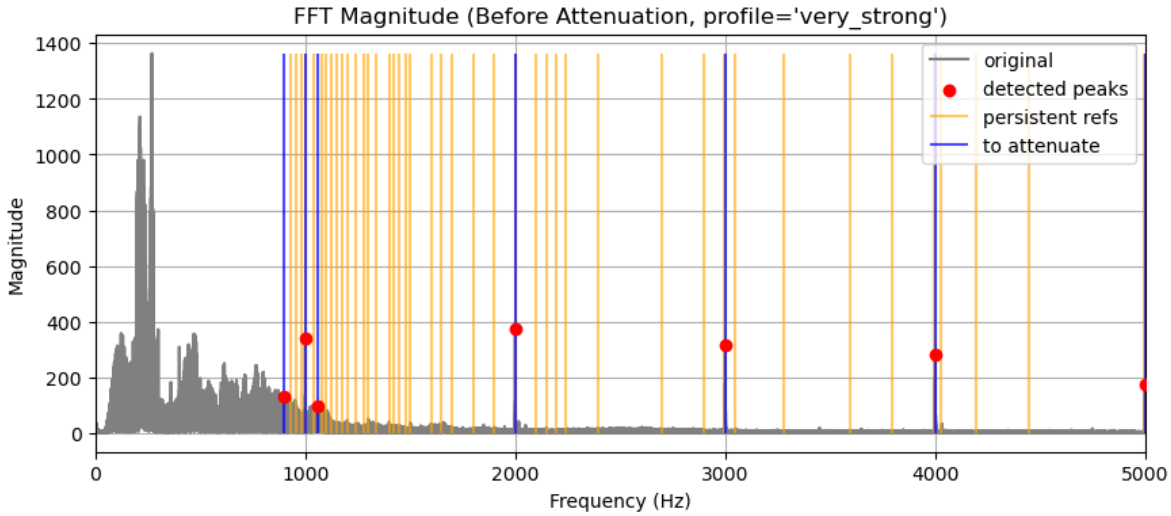


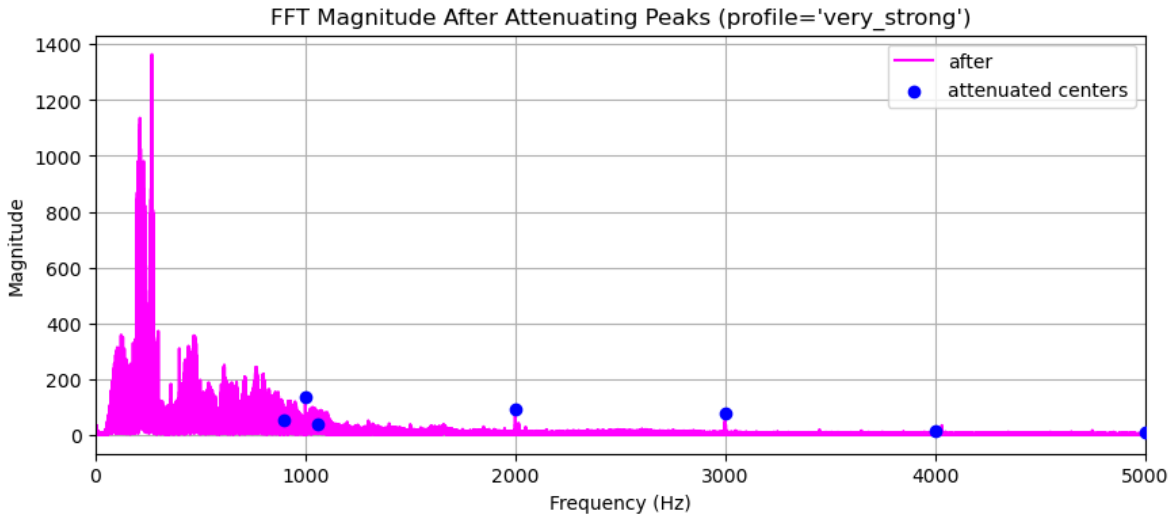

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\5_1_sample_balanced.wav
Processing with profile: strong
```



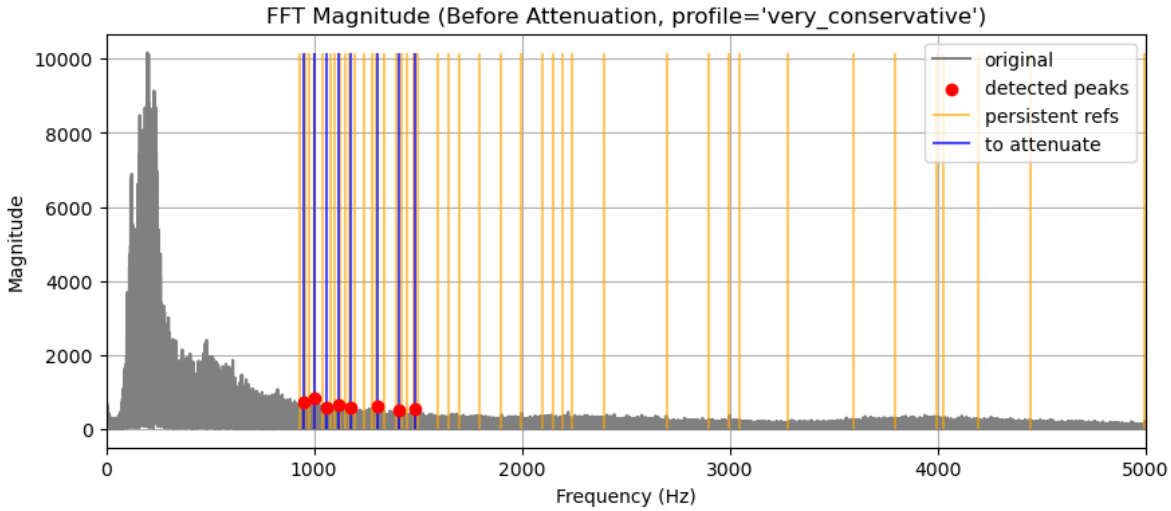


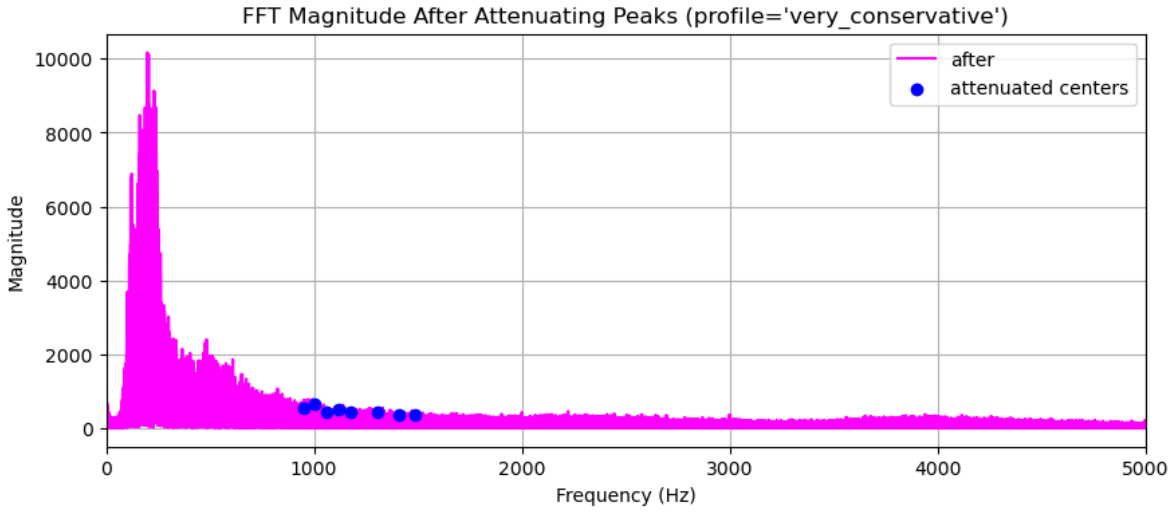

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\5_1_sample_strong.wav
Processing with profile: very_strong
```



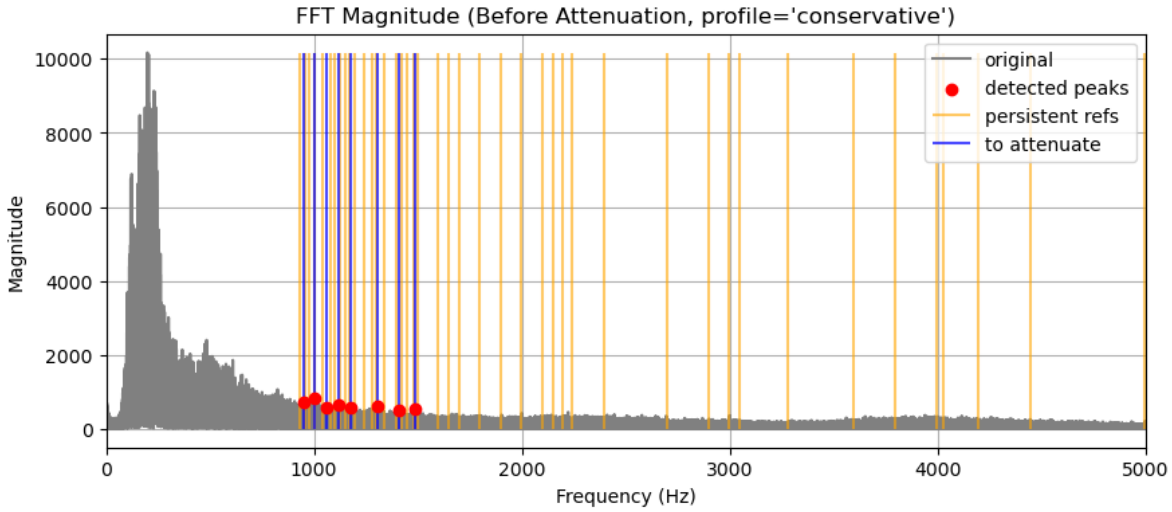


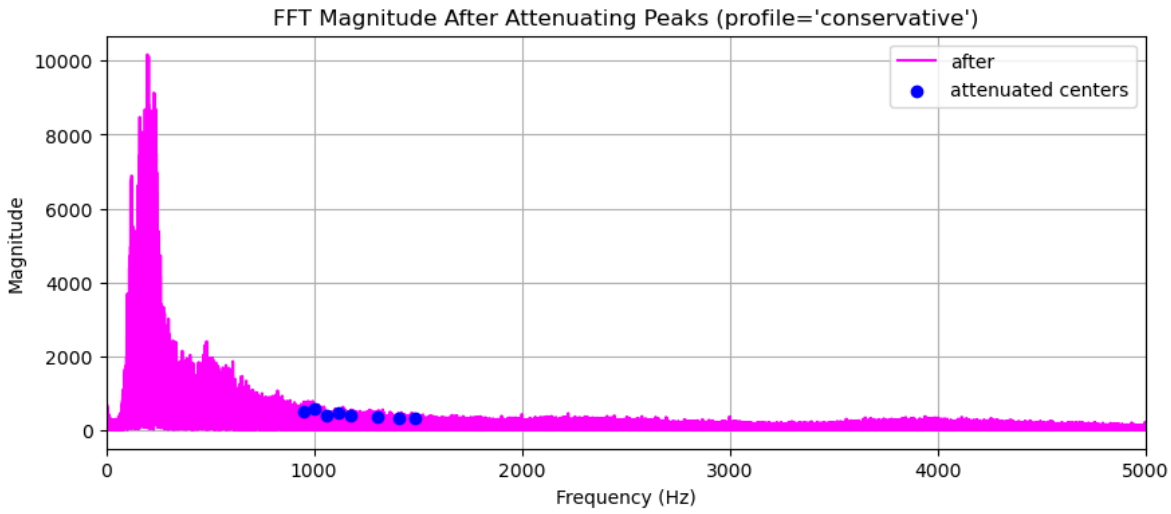

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\5_1_sample_very_strong.wav
Processing with profile: very_conservative
```



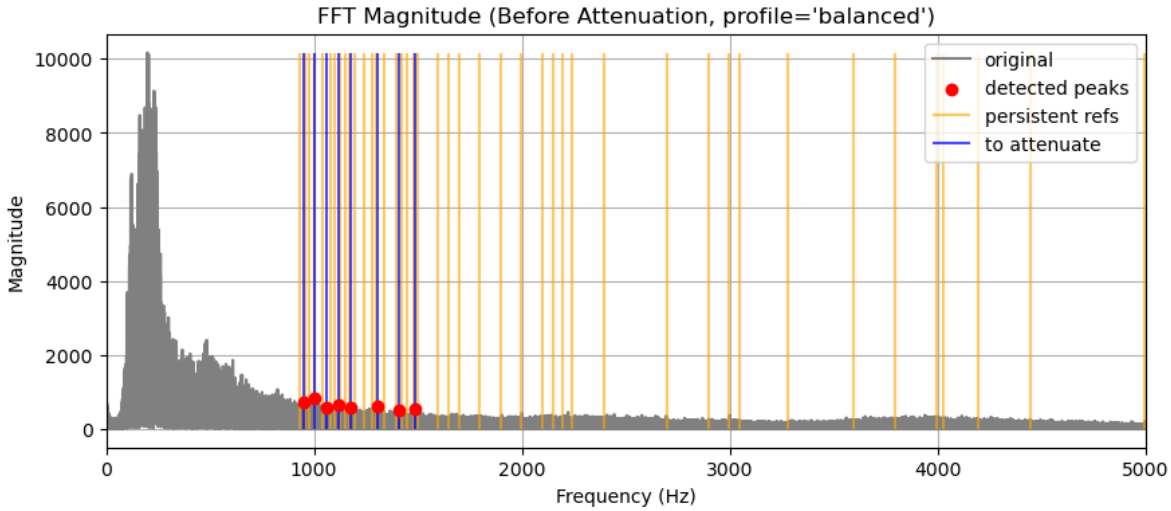


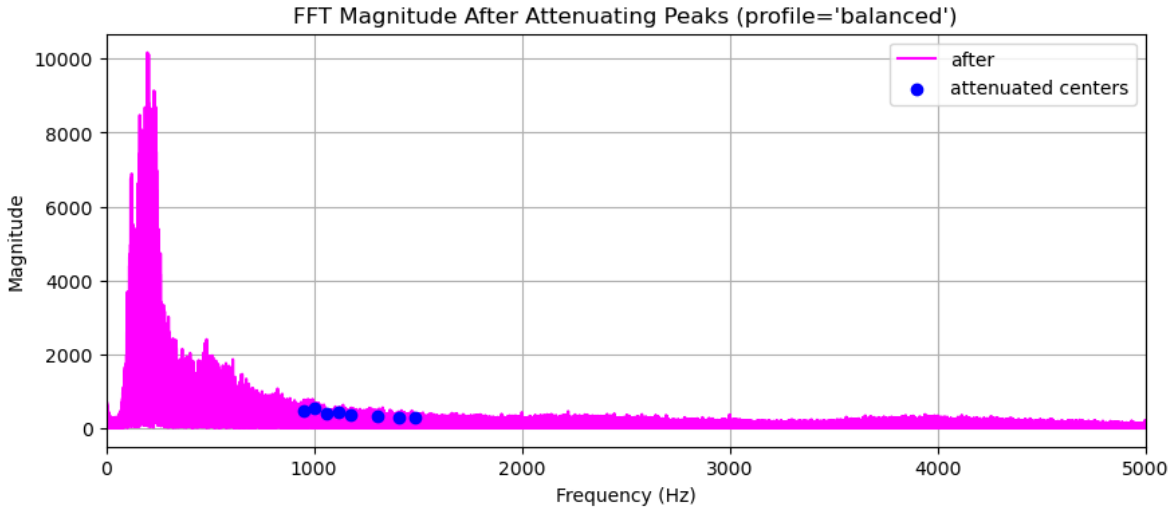

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\6_2_sample_very_conservative.w
Processing with profile: conservative
```



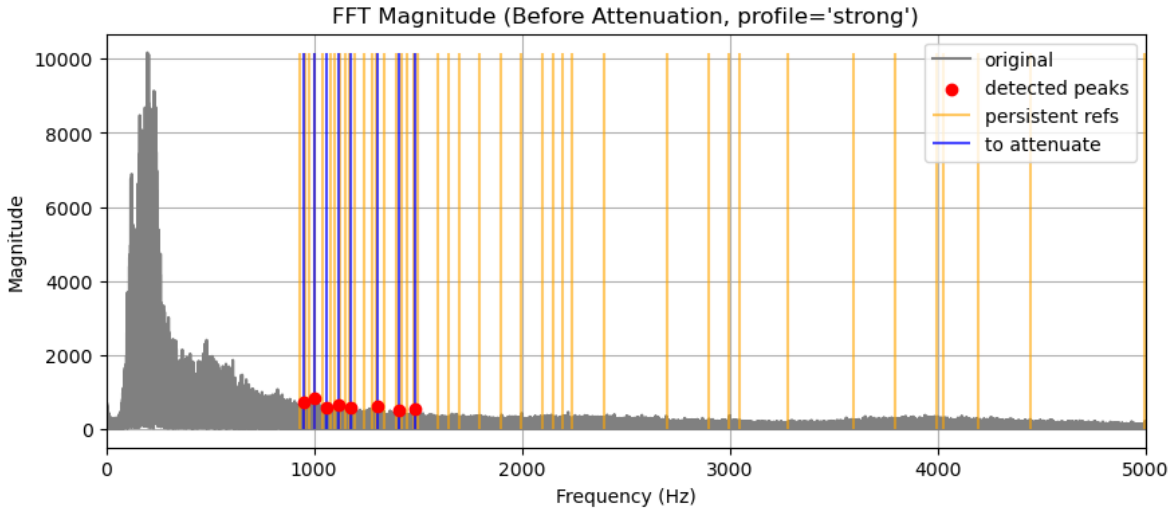


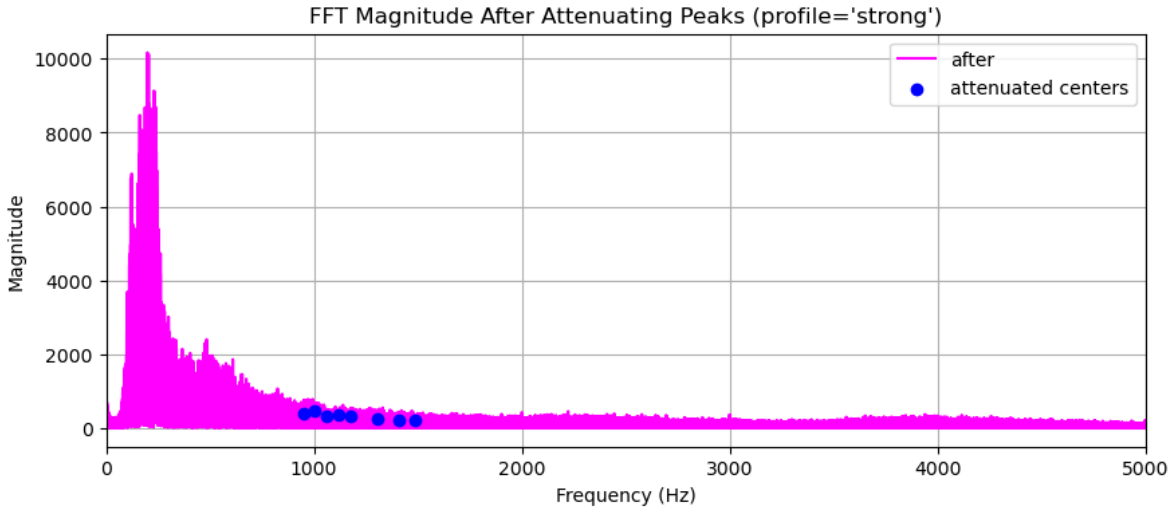

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\6_2_sample_conservative.wav
Processing with profile: balanced
```



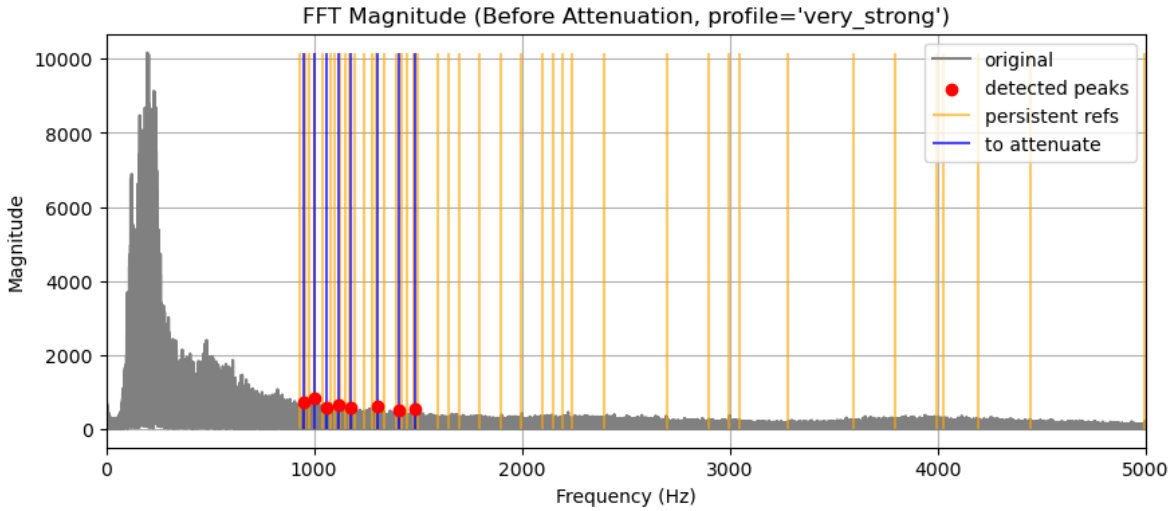


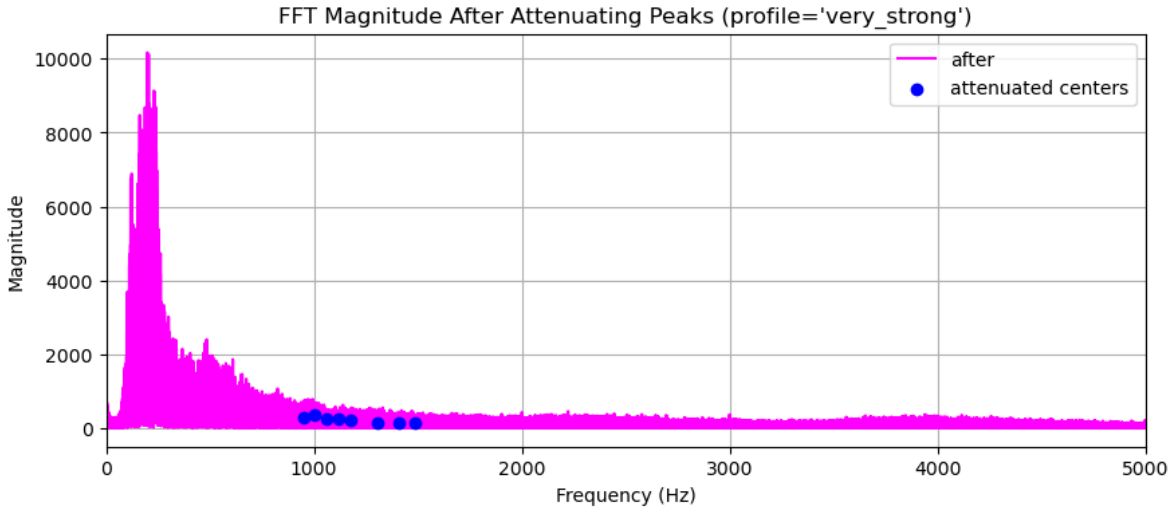

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\6_2_sample_balanced.wav
Processing with profile: strong
```






```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\6_2_sample_strong.wav
Processing with profile: very_strong
```






```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\6_2_sample_very_strong.wav
```

For complete comparison, we also create a profile that skips FFT denoising and uses only noisereduce. This will allow us to see how much distortion is introduced by noisereduce alone, compared to the combined FFT+noisereduce approach.

```
noisyfiles = os.path.join(curfolder, 'noisyfiles')
filestoprocess = glob.glob(os.path.join(noisyfiles, '*.wav'))

# get rid of those files that have conservative, strong or balanced in name
filestoprocess = [f for f in filestoprocess if all(x not in f for x in ['conservative', 'str

for sample in filestoprocess:

    data, rate = sf.read(sample)


    # then your noisereduce, if you still use it:
    #noise_clip = out[:int(rate * 0.5)]
    reduced = nr.reduce_noise(
        y=data,
        sr=rate,
        stationary=True,
        n_std_thresh_stationary=1.5,
        prop_decrease=0.8,
    )
```

```
    out_path = f"{os.path.splitext(sample)[0]}_justNR.wav"
    sf.write(out_path, reduced, rate)
    print("Saved", out_path)
```

```
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\10_1_sample_justNR.wav
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\13_1_sample_justNR.wav
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\16_1_sample_justNR.wav
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\3_1_sample_justNR.wav
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\4_2_sample_justNR.wav
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\5_1_sample_justNR.wav
Saved c:\Users\kadava\Downloads\audiohiss\audiohiss\noisyfiles\6_2_sample_justNR.wav
```

Comparison of denoising profiles

Now...

Pre-processing: extraction of acoustic features from denoised files

We extract amplitude envelope and fundamental frequency (f0) from each denoised file + use Praat script from Chris Carignan to extract formants

```
import os
import glob
import numpy as np
import pandas as pd
import librosa
import parselmouth
import scipy
from scipy.signal import butter, filtfilt


ACfolder = os.path.join(curfolder, 'noisyfiles')
processedfolder = os.path.join(curfolder, 'acoustics')

# All wav files
actotrack = glob.glob(os.path.join(ACfolder, '*.wav'))

print("Found", len(actotrack), "files to process.")
```

```
Found 0 files to process.
```

```
from scipy.signal import savgol_filter

def chunk_and_smooth(df, var, window=25, order=3):
    """
    Smooths `var` within contiguous non-NaN chunks using Savitzky-Golay.
    """
    df['chunk'] = None
    chunk = 0
    for index, row in df.iterrows():
        if np.isnan(row[var]):
            continue
        else:
            df.loc[index, 'chunk'] = chunk
            if index == len(df) - 1:
                continue
            elif np.isnan(df.loc[index + 1, var]):
                chunk += 1

    chunks = df['chunk'].unique()
    if len(chunks) > 1:
        chunks = chunks[1:]  # ignore None
        for chunk in chunks:
            chunkrows = df[df['chunk'] == chunk].copy()
            if len(chunkrows) < 5:
                continue
            chunkrows[var] = savgol_filter(chunkrows[var], window, order)
            df.loc[df['chunk'] == chunk, var] = chunkrows[var]

    df = df.drop('chunk', axis=1)
    return df

# Universal F0 range for all speakers
f0min_global = 60.0   # Hz
f0max_global = 450.0  # Hz

def extract_f0(path):
    audio, sr = librosa.load(path, sr=48000)
    snd = parselmouth.Sound(audio, sampling_frequency=sr)

    pitch = snd.to_pitch(
        time_step=0.002,
        pitch_floor=f0min_global,
```

```
        pitch_ceiling=f0max_global
    )
    f0_values = pitch.selected_array['frequency']
    return snd, f0_values

#### envelope

def butter_bandpass(lowcut, highcut, fs, order=2):
    nyq = 0.5 * fs
    low = lowcut / nyq
    high = highcut / nyq
    b, a = butter(order, [low, high], btype='band')
    return b, a

def butter_bandpass_filtfilt(data, lowcut, highcut, fs, order=2):
    b, a = butter_bandpass(lowcut, highcut, fs, order=order)
    y = filtfilt(b, a, data)
    return y

def butter_lowpass(cutoff, fs, order=2):
    nyq = 0.5 * fs
    normal_cutoff = cutoff / nyq
    b, a = butter(order, normal_cutoff, btype='low')
    return b, a

def butter_lowpass_filtfilt(data, cutoff, fs, order=2):
    b, a = butter_lowpass(cutoff, fs, order=order)
    y = filtfilt(b, a, data)
    return y

def amp_envelope(audiofilename):
    """
    Your amplitude envelope:
      - bandpass 400-4000 Hz
      - rectify
      - lowpass at 10 Hz
      - scale to 0-1
    """
    audio, sr = librosa.load(audiofilename, sr=None, mono=True)
    data = butter_bandpass_filtfilt(audio, 400, 4000, sr, order=2)
    data = butter_lowpass_filtfilt(np.abs(data), 10, sr, order=2)
    data = (data - np.min(data)) / (np.max(data) - np.min(data) + 1e-12)
```

```
    return data, sr
```

Full loop: extraction of acoustics

```
for audiofile in actotrack:

    name = os.path.basename(audiofile).split('.')[0]
    print(f"\n=== Working on {name} ===")

    # Amplitude envelope
    ampv, sr = amp_envelope(audiofile)
    rawaudio, sr_raw = librosa.load(audiofile, sr=None)
    time_env = np.arange(0, len(rawaudio) / sr_raw, 1 / sr_raw)

    # match lengths
    L = min(len(ampv), len(rawaudio), len(time_env))
    ampv = ampv[:L]
    rawaudio = rawaudio[:L]
    time_env = time_env[:L]

    env_df = pd.DataFrame({
        'time_ms': time_env * 1000,
        'audio': rawaudio,
        'envelope': ampv,
        'file': name
    })

    # Envelope change
    env_df['envelope_change'] = np.insert(np.diff(env_df['envelope']), 0, 0)
    env_df['envelope_change'] = butter_lowpass_filtfilt(
        np.abs(env_df['envelope_change']),
        10, sr_raw, order=2
    )

    env_out = os.path.join(processedfolder, f'env_{name}.csv')
    env_df.to_csv(env_out, index=False)
    print("  Saved envelope to", env_out)

    # F0 extraction
    snd, f0_vals = extract_f0(audiofile)
```

```
    f0_vals = np.where(f0_vals == 0, np.nan, f0_vals)
    f0_time = np.linspace(0, snd.duration, len(f0_vals)) * 1000  # ms

    f0_df = pd.DataFrame({
        'time_ms': f0_time,
        'f0': f0_vals,
        'name': name
    })

    # smooth F0 contour
    try:
        f0_df = chunk_and_smooth(f0_df, 'f0')
    except ValueError:
        print(f"  F0 trace very short for {name}, using smaller window=5")
        f0_df = chunk_and_smooth(f0_df, 'f0', window=5)

    f0_out = os.path.join(processedfolder, f'f0_{name}.csv')
    f0_df.to_csv(f0_out, index=False)
    print("  Saved F0 to", f0_out)
```

```
=== Working on 10_1_sample_justNR ===
  Saved envelope to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\env_10_1_sample_
  F0 trace very short for 10_1_sample_justNR, using smaller window=5
  Saved F0 to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\f0_10_1_sample_justNR.

=== Working on 13_1_sample_justNR ===
  Saved envelope to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\env_13_1_sample_
  F0 trace very short for 13_1_sample_justNR, using smaller window=5
  Saved F0 to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\f0_13_1_sample_justNR.

=== Working on 16_1_sample_justNR ===
  Saved envelope to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\env_16_1_sample_
  F0 trace very short for 16_1_sample_justNR, using smaller window=5
  Saved F0 to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\f0_16_1_sample_justNR.

=== Working on 3_1_sample_justNR ===
  Saved envelope to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\env_3_1_sample_ju
  F0 trace very short for 3_1_sample_justNR, using smaller window=5
  Saved F0 to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\f0_3_1_sample_justNR.cs

=== Working on 4_2_sample_justNR ===
```

```
  Saved envelope to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\env_4_2_sample_ju
  F0 trace very short for 4_2_sample_justNR, using smaller window=5
  Saved F0 to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\f0_4_2_sample_justNR.cs

=== Working on 5_1_sample_justNR ===
  Saved envelope to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\env_5_1_sample_ju
  F0 trace very short for 5_1_sample_justNR, using smaller window=5
  Saved F0 to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\f0_5_1_sample_justNR.cs

=== Working on 6_2_sample_justNR ===
  Saved envelope to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\env_6_2_sample_ju
  F0 trace very short for 6_2_sample_justNR, using smaller window=5
  Saved F0 to c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\f0_6_2_sample_justNR.cs
```

Timeseries comparison

```python
import os
import glob
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

ACfolder_processed = os.path.join(curfolder, "acoustics")

PROFILES = [
    "original",
    "balanced",
    "conservative",
    "strong",
    "very_conservative",
    "very_strong",
    "justNR",
]

def get_env_path(trialid, profile):
    if profile == "original":
        return os.path.join(ACfolder_processed, f"env_{trialid}.csv")
    else:
        return os.path.join(ACfolder_processed, f"env_{trialid}_{profile}.csv")

def get_f0_path(trialid, profile):
```

```
    if profile == "original":
        return os.path.join(ACfolder_processed, f"f0_{trialid}.csv")
    else:
        return os.path.join(ACfolder_processed, f"f0_{trialid}_{profile}.csv")

def get_formant_paths(trialid, profile):
    if profile == "original":
        pattern = os.path.join(ACfolder_processed, f"{trialid}_table_formants*.csv")
    else:
        pattern = os.path.join(ACfolder_processed, f"{trialid}_{profile}_table_formants*.csv
    return glob.glob(pattern)

def downsample_series(t_ms, y, step):
    """
    Keep every `step`-th sample to lighten plotting.
    """
    if step <= 1:
        return t_ms, y
    return t_ms[::step], y[::step]
```

```
def plot_profiles_quick(
    trialid,
    profiles=None,
    t_min_ms=None,
    t_max_ms=None,
    decim_env=20,
    decim_form=10,
):
    """
    Plot envelope, F0 and F1-F3 across selected profiles.

    Parameters:
    -----------
    trialid : str
        Trial ID like '10_1_sample'.

    profiles : list or None
        Which profiles to plot. If None → uses all profiles.

    t_min_ms, t_max_ms : float or None
        Crop to this time range in milliseconds.
```

```
    decim_env : int
        Downsampling factor for envelope.

    decim_form : int
        Downsampling factor for formants.
    """

    # full list of profiles if none specified
    if profiles is None:
        profiles = [
            "original",
            "balanced",
            "conservative",
            "strong",
            "very_conservative",
            "very_strong",
            "justNR"
        ]

    # colors for consistency
    colors = {
        "original": "k",
        "balanced": "C0",
        "conservative": "C1",
        "strong": "C2",
        "very_conservative": "C3",
        "very_strong": "C4",
        "justNR": "C5",
    }

    # ------------------------------------------------
    # 1) Envelope
    # ------------------------------------------------
    plt.figure(figsize=(10, 4))
    for profile in profiles:
        env_path = get_env_path(trialid, profile)
        if not os.path.exists(env_path):
            print(f"[env] Missing for {trialid}, profile={profile}")
            continue

        env_df = pd.read_csv(env_path)
        t = env_df["time_ms"].values
```

```
        env = env_df["envelope"].values

        # time cropping
        if t_min_ms is not None or t_max_ms is not None:
            mask = np.ones_like(t, dtype=bool)
            if t_min_ms is not None:
                mask &= (t >= t_min_ms)
            if t_max_ms is not None:
                mask &= (t <= t_max_ms)
            t = t[mask]
            env = env[mask]

        # downsample
        t_ds, env_ds = downsample_series(t, env, decim_env)

        plt.plot(
            t_ds,
            env_ds,
            label=profile,
            alpha=0.8,
            linewidth=1.0,
            color=colors.get(profile, None),
        )

    plt.title(f"Envelope across profiles - {trialid}")
    plt.xlabel("Time (ms)")
    plt.ylabel("Envelope (0-1)")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    # ------------------------------------------------
    # 2) F0
    # ------------------------------------------------
    plt.figure(figsize=(10, 4))
    for profile in profiles:
        f0_path = get_f0_path(trialid, profile)
        if not os.path.exists(f0_path):
            print(f"[f0] Missing for {trialid}, profile={profile}")
            continue
```

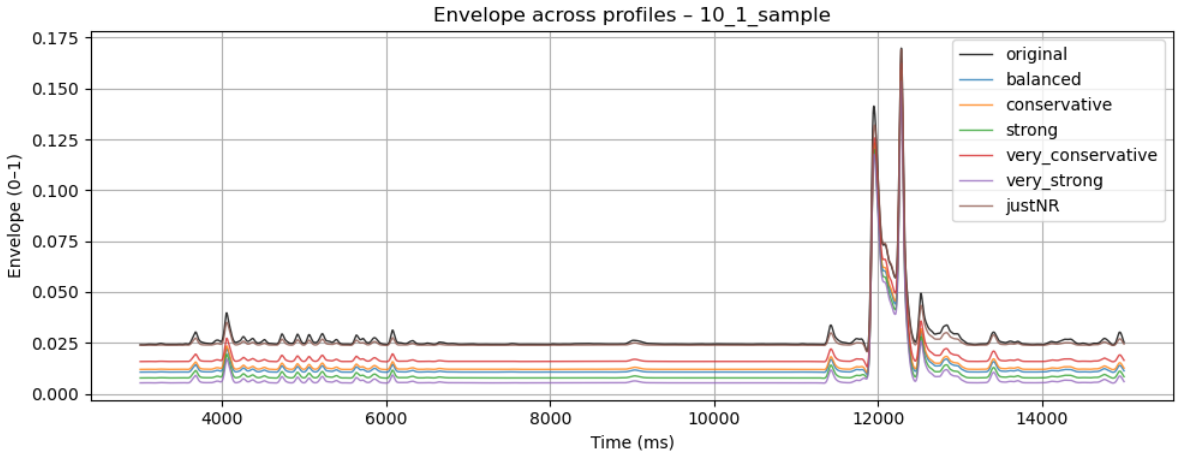
```
        f0_df = pd.read_csv(f0_path)
        t = f0_df["time_ms"].values
        f0 = f0_df["f0"].values

        if t_min_ms is not None or t_max_ms is not None:
            mask = np.ones_like(t, dtype=bool)
            if t_min_ms is not None:
                mask &= (t >= t_min_ms)
            if t_max_ms is not None:
                mask &= (t <= t_max_ms)
            t = t[mask]
            f0 = f0[mask]

        plt.plot(
            t,
            f0,
            label=profile,
            alpha=0.8,
            linewidth=1.0,
            color=colors.get(profile, None),
        )

    plt.title(f"F0 across profiles - {trialid}")
    plt.xlabel("Time (ms)")
    plt.ylabel("F0 (Hz)")
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

    # ------------------------------------------------
    # 3) Formants F1-F3
    # ------------------------------------------------
    formant_names = ["f1", "f2", "f3"]
    fig, axes = plt.subplots(3, 1, figsize=(10, 9), sharex=True)
    any_formant = False

    for profile in profiles:
        formant_paths = get_formant_paths(trialid, profile)
        if not formant_paths:
            print(f"[formants] Missing for {trialid}, profile={profile}")
            continue
```

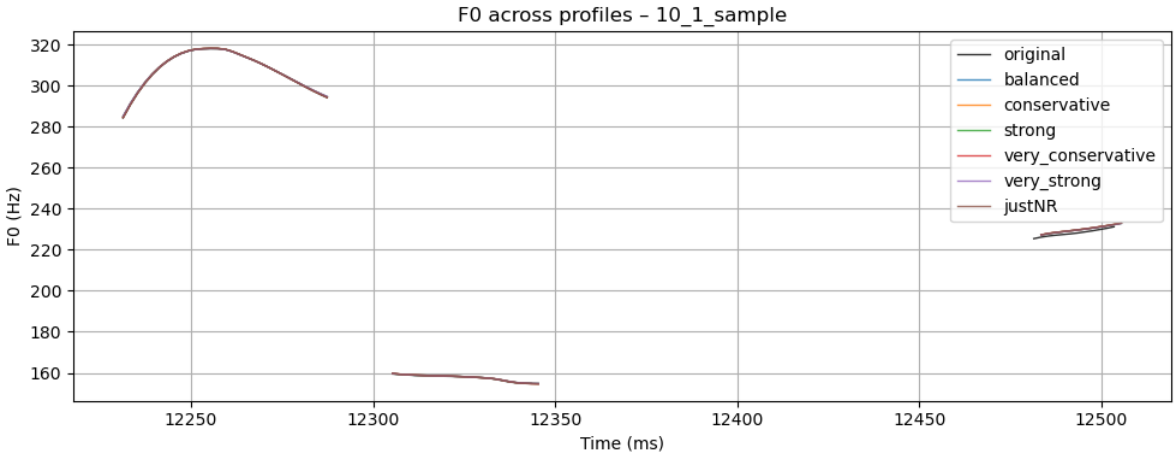
```
        frames = []
        for fp in formant_paths:
            try:
                df = pd.read_csv(fp)
                if "time_ms" not in df.columns and "time" in df.columns:
                    df["time_ms"] = df["time"] * 1000.0
                frames.append(df)
            except Exception as e:
                print("  Error reading formant file:", fp, e)

        if not frames:
            continue

        fdf = pd.concat(frames, ignore_index=True)
        t = fdf["time_ms"].values

        # time cropping
        if t_min_ms is not None or t_max_ms is not None:
            mask = np.ones_like(t, dtype=bool)
            if t_min_ms is not None:
                mask &= (t >= t_min_ms)
            if t_max_ms is not None:
                mask &= (t <= t_max_ms)
            fdf = fdf[mask]
            t = fdf["time_ms"].values

        # downsample time axis for formants
        t_ds, _ = downsample_series(t, t, decim_form)

        for ax, F in zip(axes, formant_names):
            if F not in fdf.columns:
                continue

            y = fdf[F].values
            _, y_ds = downsample_series(t, y, decim_form)

            ax.plot(
                t_ds,
                y_ds,
                label=profile,
                alpha=0.7,
                linewidth=1.0,
```

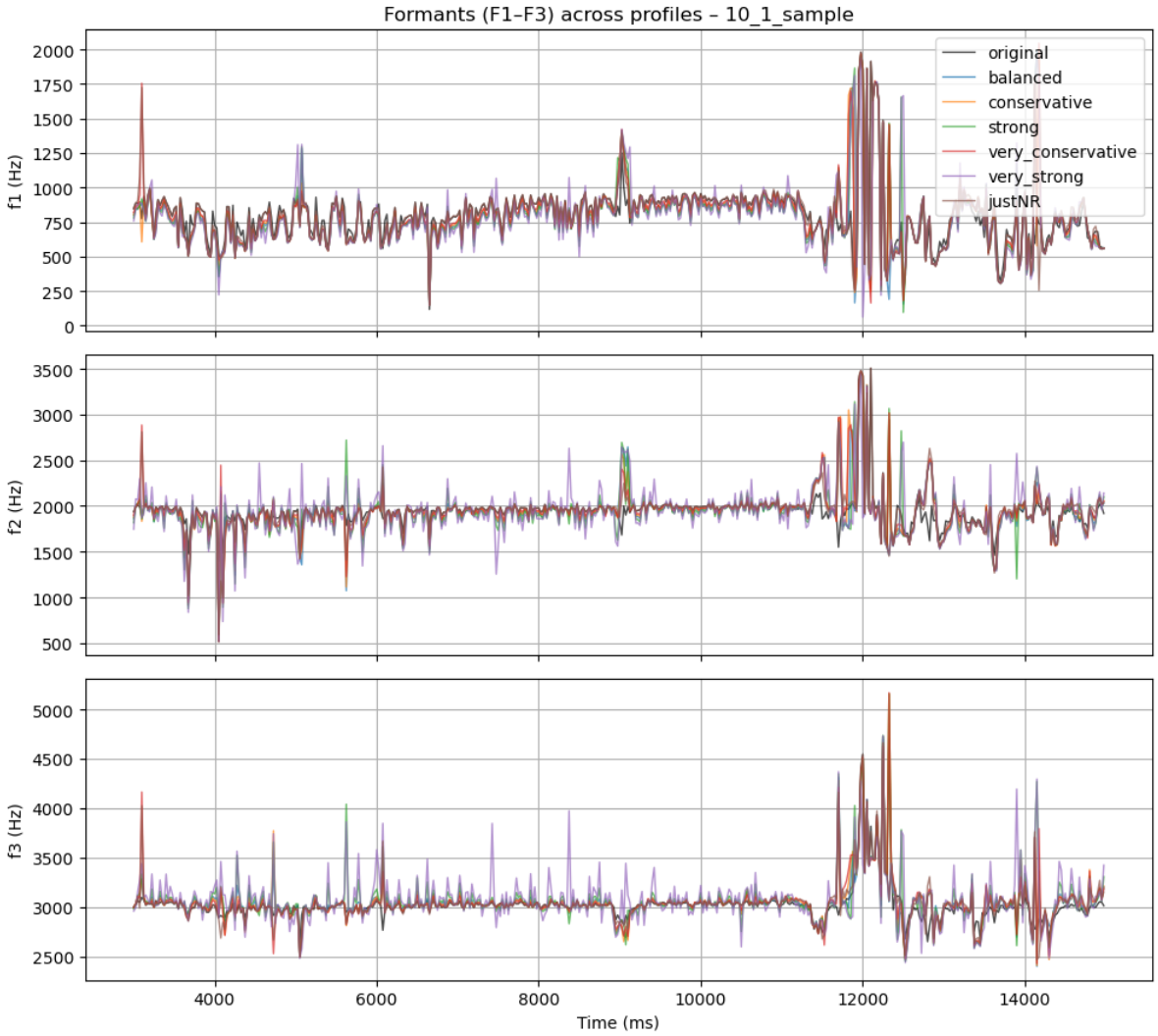
```
                color=colors.get(profile, None),
            )
            any_formant = True

    if not any_formant:
        print(f"No formant data to plot for {trialid}")
        plt.close(fig)
        return

    for ax, F in zip(axes, formant_names):
        ax.set_ylabel(F + " (Hz)")
        ax.grid(True)

    axes[-1].set_xlabel("Time (ms)")
    axes[0].set_title(f"Formants (F1-F3) across profiles - {trialid}")
    axes[0].legend()

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

Example 1

```
plot_profiles_quick("10_1_sample", t_min_ms=3000, t_max_ms=15000, decim_env=5, decim_form=5)
```

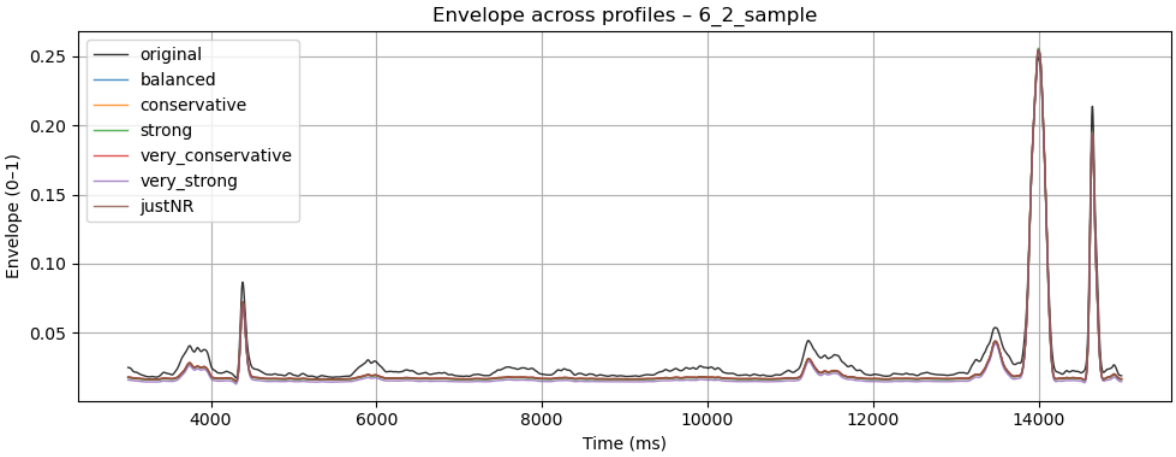


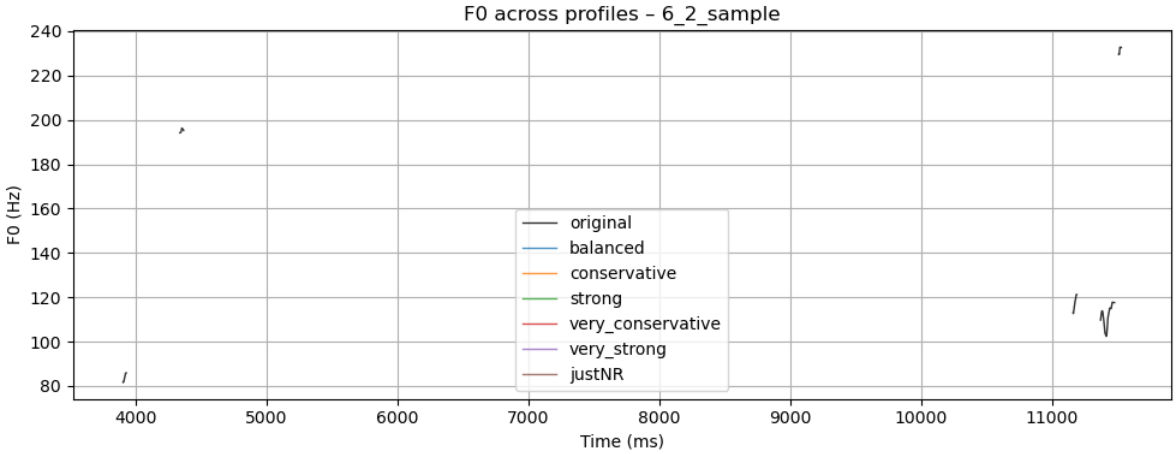

F0 across profiles – 10_1_sample
F0 (Hz)
320
300
280
260
240
220
200
180
160
12250
12300
12350
12400
12450
12500
Time (ms)
original
balanced
conservative
strong
very_conservative
very_strong
justNR

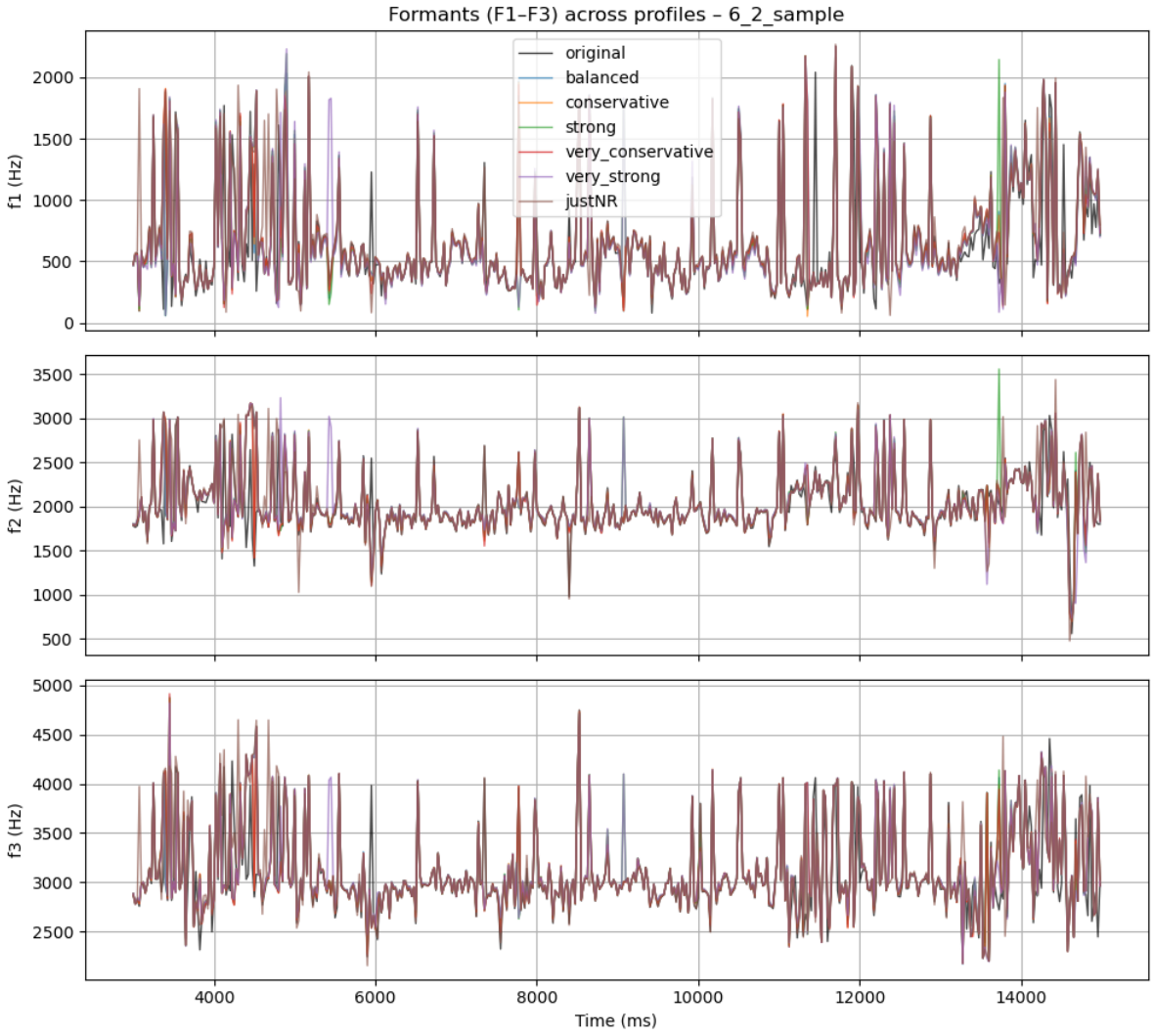


Example 2

```
plot_profiles_quick("6_2_sample", t_min_ms=3000, t_max_ms=15000, decim_env=5, decim_form=5)
```

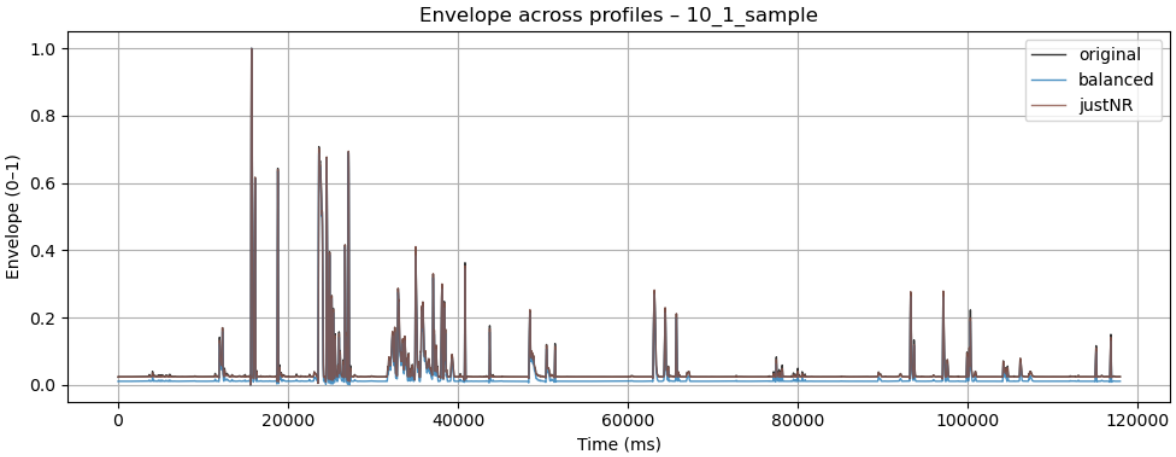

Envelope across profiles – 6_2_sample
original
balanced
conservative
strong
very_conservative
very_strong
justNR
Envelope (0–1)
0.25
0.20
0.15
0.10
0.05
4000
6000
8000
10000
12000
14000
Time (ms)

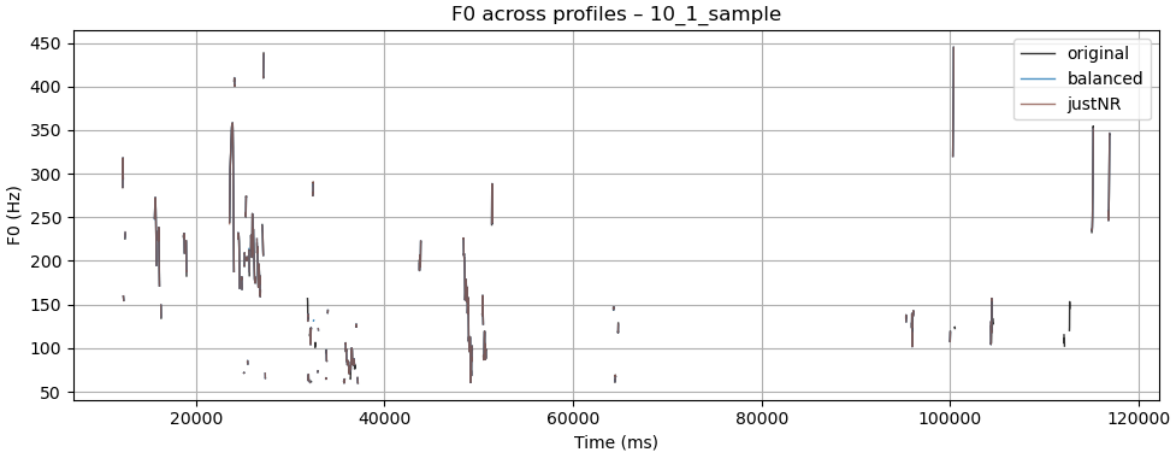

F0 across profiles – 6_2_sample
original
balanced
conservative
strong
very_conservative
very_strong
justNR
F0 (Hz)
240
220
200
180
160
140
120
100
80
4000
5000
6000
7000
8000
9000
10000
11000
Time (ms)

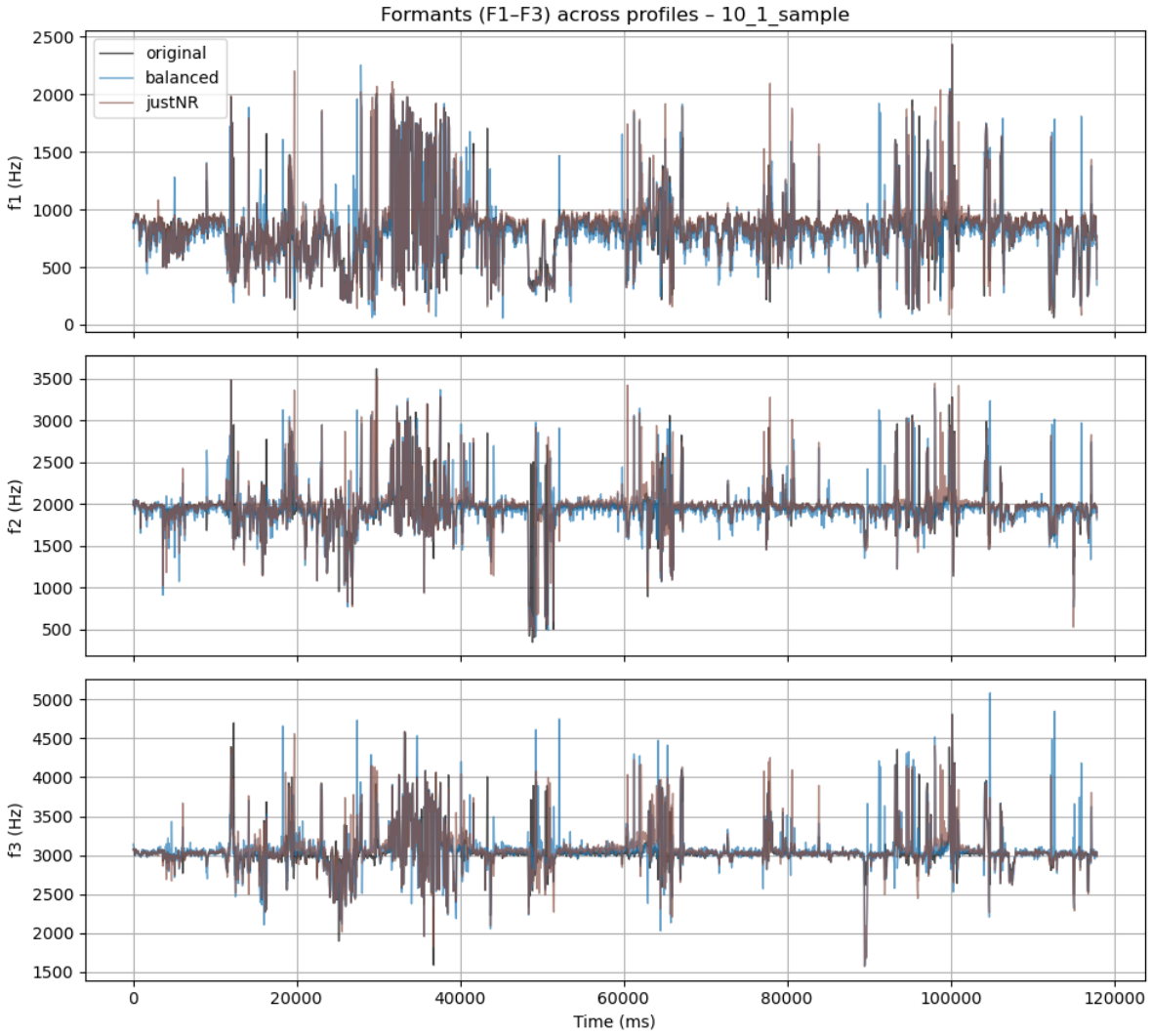


Example 3: plot just selected profiles

```
plot_profiles_quick("10_1_sample", profiles=["original", "balanced", "justNR"])
```


Envelope across profiles – 10_1_sample
original
balanced
justNR
Envelope (0–1)
Time (ms)
0.0
0.2
0.4
0.6
0.8
1.0
0
20000
40000
60000
80000
100000
120000


F0 across profiles – 10_1_sample
original
balanced
justNR
F0 (Hz)
Time (ms)
50
100
150
200
250
300
350
400
450
20000
40000
60000
80000
100000
120000



Compare features across files

```
import os
import glob
import numpy as np
import pandas as pd
import scipy.stats

ACfolder_processed = os.path.join(curfolder, 'acoustics')
```

```
def extract_summary_features(env_path, f0_path, formant_paths):
    summary = {}

    # ------------ ENV ------------
    env_df = pd.read_csv(env_path)
    env = env_df['envelope'].values
    env_ch = env_df['envelope_change'].values

    summary['env_mean']   = np.nanmean(env)
    summary['env_sd']     = np.nanstd(env)
    summary['env_range']  = np.nanmax(env) - np.nanmin(env)
    summary['env_kurtosis'] = scipy.stats.kurtosis(env, nan_policy='omit')
    summary['env_slope_mean'] = np.nanmean(env_ch)
    summary['env_slope_sd']   = np.nanstd(env_ch)

    # ------------ F0 ------------
    f0_df = pd.read_csv(f0_path)
    f0 = f0_df['f0'].values
    f0_nonan = f0[~np.isnan(f0)]

    if f0_nonan.size > 0:
        summary['f0_mean']  = np.nanmean(f0_nonan)
        summary['f0_sd']    = np.nanstd(f0_nonan)
        summary['f0_range'] = np.nanmax(f0_nonan) - np.nanmin(f0_nonan)
    else:
        summary['f0_mean'] = summary['f0_sd'] = summary['f0_range'] = np.nan

    summary['f0_percent_voiced'] = float(np.mean(~np.isnan(f0))) if f0.size > 0 else np.nan

    # ------------ FORMANTS ------------
    frames = []
    for fp in formant_paths:
        try:
            df = pd.read_csv(fp)
            frames.append(df)
        except Exception as e:
            print("Error reading formant file:", fp, e)

    if frames:
        formants = pd.concat(frames, ignore_index=True)
    else:
        formants = pd.DataFrame(columns=['f1','f2','f3','f4','f5'])
```

```
    for F in ['f1', 'f2', 'f3', 'f4', 'f5']:
        if F in formants.columns:
            col = formants[F].values
            col_nonan = col[~np.isnan(col)]
            if col_nonan.size > 0:
                summary[f'{F}_mean']  = np.nanmean(col_nonan)
                summary[f'{F}_sd']    = np.nanstd(col_nonan)
                summary[f'{F}_range'] = np.nanmax(col_nonan) - np.nanmin(col_nonan)
            else:
                summary[f'{F}_mean'] = summary[f'{F}_sd'] = summary[f'{F}_range'] = np.nan
        else:
            summary[f'{F}_mean'] = summary[f'{F}_sd'] = summary[f'{F}_range'] = np.nan

    return summary
```

```
KNOWN_PROFILES = {
    "strong",
    "very_strong",
    "conservative",
    "very_conservative",
    "balanced",
    "justNR",
}

summary_rows = []

env_files = glob.glob(os.path.join(ACfolder_processed, "env_*.csv"))

for env_path in env_files:
    base = os.path.splitext(os.path.basename(env_path))[0]
    # e.g. "env_10_1_sample", "env_10_1_sample_strong", "env_10_1_sample_very_conservative"

    if not base.startswith("env_"):
        continue

    rest = base[len("env_"):]              # e.g. "10_1_sample_strong"
    parts = rest.split("_")

    if len(parts) < 3:
        print("Skipping weird name:", base)
        continue
```

```
    # first 3 tokens = trialid
    trialid = "_".join(parts[:3])          # "10_1_sample"

    # remaining tokens = profile tokens
    if len(parts) > 3:
        profile = "_".join(parts[3:])      # e.g. "strong", "very_conservative"
    else:
        profile = ""                       # no suffix → original

    if profile == "":
        profile = "original"
    elif profile not in KNOWN_PROFILES:
        print(f"  Warning: unknown profile '{profile}' in {base}; treating as 'original'")
        profile = "original"

    print(f"\n=== {trialid} | profile: {profile} ===")

    # ----------------- build f0 path -----------------
    if profile == "original":
        f0_filename = f"f0_{trialid}.csv"
    else:
        f0_filename = f"f0_{trialid}_{profile}.csv"

    f0_path = os.path.join(ACfolder_processed, f0_filename)
    if not os.path.exists(f0_path):
        print("  Missing f0 file:", f0_path)
        continue

    # ----------------- find formant files -----------------
    if profile == "original":
        formant_pattern = os.path.join(
            ACfolder_processed,
            f"{trialid}_table_formants*.csv"
        )
    else:
        formant_pattern = os.path.join(
            ACfolder_processed,
            f"{trialid}_{profile}_table_formants*.csv"
        )

    formant_paths = glob.glob(formant_pattern)
    if not formant_paths:
```

```
        print("  No formant files for", trialid, profile)

    # ----------------- extract features -----------------
    feats = extract_summary_features(env_path, f0_path, formant_paths)
    feats["trialid"] = trialid
    feats["profile"] = profile

    summary_rows.append(feats)

summary_df = pd.DataFrame(summary_rows)
out_summary = os.path.join(ACfolder_processed, "summary_features_by_profile.csv")
summary_df.to_csv(out_summary, index=False)
print("\nSaved summary features to:", out_summary)
```

```
=== 10_1_sample | profile: original ===

=== 10_1_sample | profile: balanced ===

=== 10_1_sample | profile: conservative ===

=== 10_1_sample | profile: justNR ===

=== 10_1_sample | profile: strong ===

=== 10_1_sample | profile: very_conservative ===

=== 10_1_sample | profile: very_strong ===

=== 13_1_sample | profile: original ===

=== 13_1_sample | profile: balanced ===

=== 13_1_sample | profile: conservative ===

=== 13_1_sample | profile: justNR ===

=== 13_1_sample | profile: strong ===

=== 13_1_sample | profile: very_conservative ===

=== 13_1_sample | profile: very_strong ===
```

```
=== 16_1_sample | profile: original ===

=== 16_1_sample | profile: balanced ===

=== 16_1_sample | profile: conservative ===

=== 16_1_sample | profile: justNR ===

=== 16_1_sample | profile: strong ===

=== 16_1_sample | profile: very_conservative ===

=== 16_1_sample | profile: very_strong ===

=== 3_1_sample | profile: original ===

=== 3_1_sample | profile: balanced ===

=== 3_1_sample | profile: conservative ===

=== 3_1_sample | profile: justNR ===

=== 3_1_sample | profile: strong ===

=== 3_1_sample | profile: very_conservative ===

=== 3_1_sample | profile: very_strong ===

=== 4_2_sample | profile: original ===

=== 4_2_sample | profile: balanced ===

=== 4_2_sample | profile: conservative ===

=== 4_2_sample | profile: justNR ===

=== 4_2_sample | profile: strong ===

=== 4_2_sample | profile: very_conservative ===

=== 4_2_sample | profile: very_strong ===
```

```
=== 5_1_sample | profile: original ===

=== 5_1_sample | profile: balanced ===

=== 5_1_sample | profile: conservative ===

=== 5_1_sample | profile: justNR ===

=== 5_1_sample | profile: strong ===

=== 5_1_sample | profile: very_conservative ===

=== 5_1_sample | profile: very_strong ===

=== 6_2_sample | profile: original ===

=== 6_2_sample | profile: balanced ===

=== 6_2_sample | profile: conservative ===

=== 6_2_sample | profile: justNR ===

=== 6_2_sample | profile: strong ===

=== 6_2_sample | profile: very_conservative ===

=== 6_2_sample | profile: very_strong ===

Saved summary features to: c:\Users\kadava\Downloads\audiohiss\audiohiss\acoustics\summary_fe
```

```
display(summary_df)
```

	env_mean	env_sd	env_range	env_kurtosis	env_slope_mean	env_slope_sd	f0_mean	f0_
0	0.039775	0.066599	1.0	67.175044	0.000005	0.000018	175.348096	85.4
1	0.026148	0.066765	1.0	67.937055	0.000004	0.000018	179.133654	86.3
2	0.027469	0.066727	1.0	67.778432	0.000004	0.000018	179.133379	86.3
3	0.039047	0.066008	1.0	67.362230	0.000004	0.000018	179.662757	86.0
4	0.023368	0.066887	1.0	67.984284	0.000004	0.000017	178.777687	86.4
5	0.031251	0.066494	1.0	67.622074	0.000004	0.000018	179.202414	86.3
6	0.021040	0.066927	1.0	68.022981	0.000004	0.000017	178.609395	86.5
7	0.037209	0.061627	1.0	96.460316	0.000004	0.000018	202.829675	75.0

	env_mean	env_sd	env_range	env_kurtosis	env_slope_mean	env_slope_sd	f0_mean	f0_
8	0.037833	0.061911	1.0	96.920424	0.000004	0.000018	207.177845	77.7
9	0.037662	0.061613	1.0	97.748030	0.000004	0.000018	207.036140	77.7
10	0.037386	0.060597	1.0	100.644088	0.000004	0.000018	206.866419	77.7
11	0.037953	0.062194	1.0	95.714869	0.000004	0.000018	206.173550	77.6
12	0.037521	0.061178	1.0	98.837560	0.000004	0.000018	207.035527	77.7
13	0.037029	0.062510	1.0	94.510359	0.000004	0.000018	206.551003	77.6
14	0.071731	0.123736	1.0	15.857855	0.000013	0.000034	149.410723	42.8
15	0.062964	0.118449	1.0	18.302483	0.000012	0.000034	153.870551	43.0
16	0.063147	0.118430	1.0	18.303526	0.000012	0.000034	153.874341	43.0
17	0.065265	0.117889	1.0	18.326382	0.000012	0.000034	153.891121	43.0
18	0.061591	0.118551	1.0	18.293096	0.000012	0.000034	153.870349	43.0
19	0.064483	0.118277	1.0	18.308789	0.000012	0.000034	153.884392	43.0
20	0.060063	0.118785	1.0	18.274124	0.000012	0.000034	153.876063	43.0
21	0.100275	0.151550	1.0	7.815476	0.000019	0.000039	213.131388	27.5
22	0.085075	0.137695	1.0	9.440148	0.000017	0.000035	214.373315	27.1
23	0.085828	0.138824	1.0	9.427500	0.000017	0.000036	214.357880	27.0
24	0.086162	0.139774	1.0	9.453002	0.000017	0.000036	214.341462	27.0
25	0.084348	0.136746	1.0	9.446022	0.000016	0.000035	214.378051	27.1
26	0.085822	0.138782	1.0	9.438190	0.000017	0.000036	214.362558	27.0
27	0.083706	0.135994	1.0	9.450556	0.000016	0.000035	214.365900	27.1
28	0.025071	0.057757	1.0	124.493654	0.000003	0.000012	245.579248	59.5
29	0.023109	0.055236	1.0	130.773033	0.000002	0.000012	246.962402	59.5
30	0.023006	0.055347	1.0	130.545025	0.000002	0.000012	247.107794	59.5
31	0.023279	0.055563	1.0	130.486706	0.000002	0.000012	247.091442	59.4
32	0.023152	0.055182	1.0	130.879163	0.000002	0.000012	247.102858	59.4
33	0.022955	0.055459	1.0	130.275106	0.000002	0.000012	247.079154	59.4
34	0.023131	0.055214	1.0	131.221110	0.000002	0.000012	247.073456	59.4
35	0.031687	0.052797	1.0	143.991048	0.000004	0.000021	220.591219	34.4
36	0.027661	0.051367	1.0	156.748034	0.000003	0.000021	219.923161	37.7
37	0.027675	0.051278	1.0	157.549644	0.000003	0.000021	219.925840	37.7
38	0.027796	0.051455	1.0	158.604540	0.000003	0.000021	219.899365	37.7
39	0.027478	0.051221	1.0	155.994477	0.000003	0.000021	219.920311	37.7
40	0.027691	0.051197	1.0	158.069823	0.000003	0.000021	219.898963	37.7
41	0.027242	0.051052	1.0	155.481894	0.000003	0.000021	219.888913	37.7
42	0.050565	0.093896	1.0	22.367047	0.000007	0.000021	153.589915	53.6
43	0.046148	0.095135	1.0	22.762051	0.000006	0.000021	156.957367	54.3
44	0.046194	0.095045	1.0	22.783496	0.000006	0.000021	156.998420	54.3
45	0.046174	0.094925	1.0	22.840370	0.000006	0.000021	156.923332	54.3
46	0.045965	0.095564	1.0	22.692545	0.000006	0.000021	156.831261	54.3
47	0.046198	0.094999	1.0	22.802747	0.000006	0.000021	156.957974	54.3

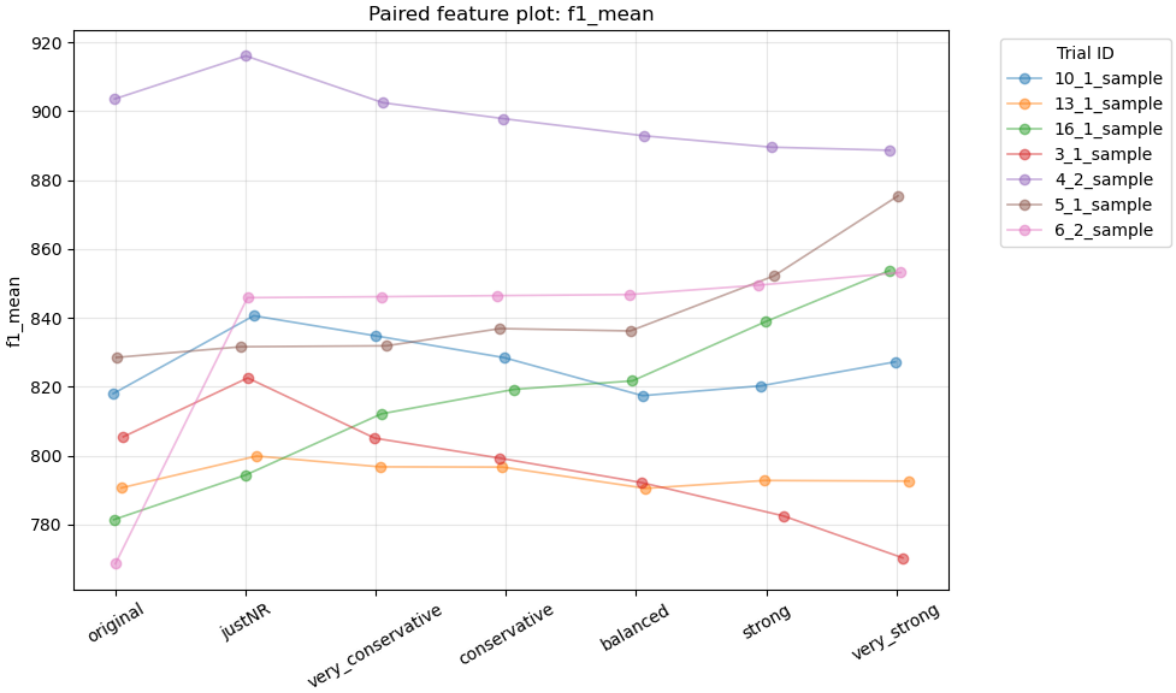
48	0.044489	0.095699	1.0	22.626786	0.000006	0.000021	157.111482	54.5

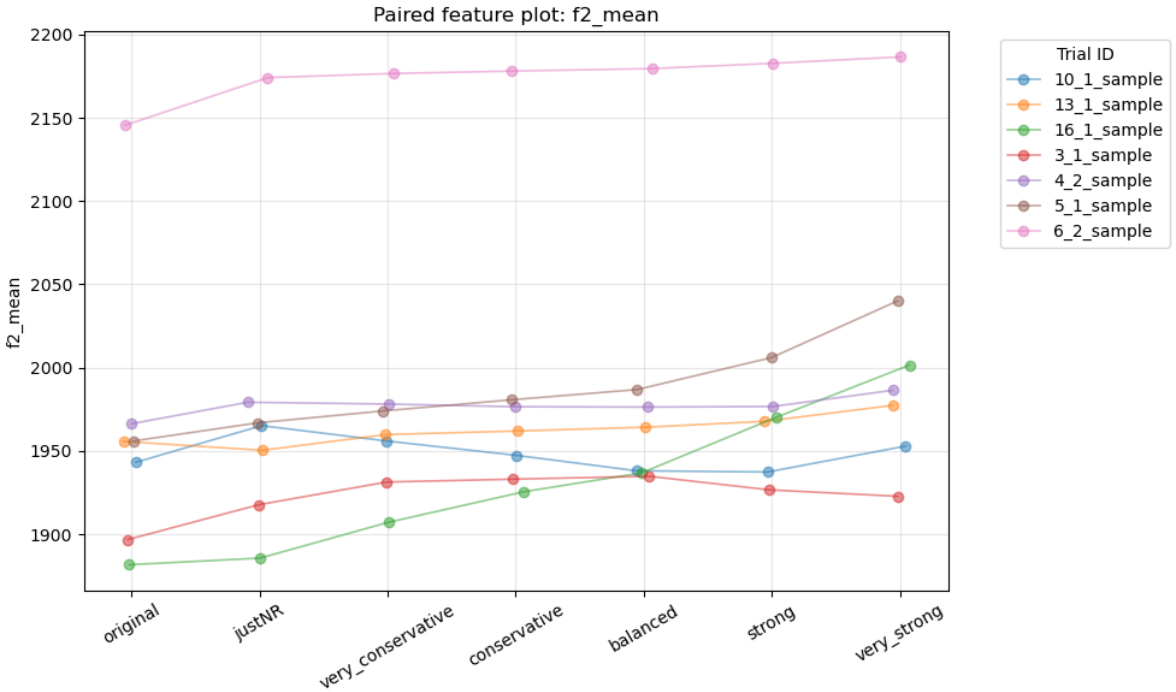
Plot differences

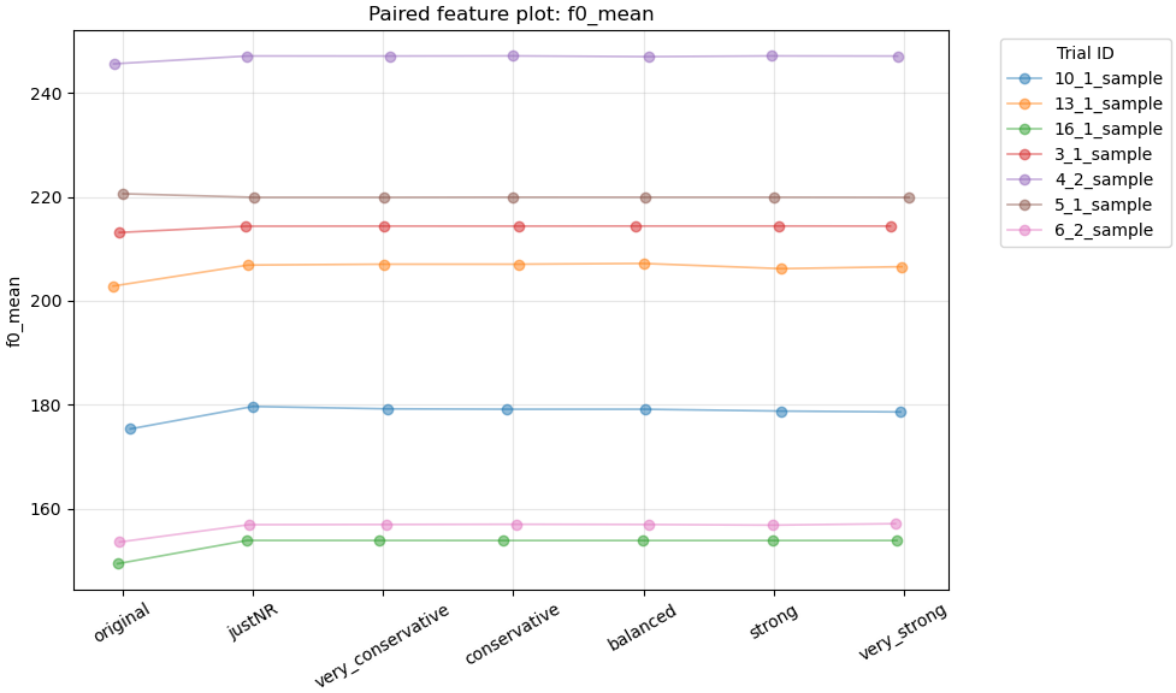
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

PROFILE_ORDER = [
    "original",
    "justNR",
    "very_conservative",
    "conservative",
    "balanced",
    "strong",
    "very_strong",
]

def plot_paired_feature(summary_df, feature_col, profile_order=PROFILE_ORDER):
    """
    For each trial, connect the feature values across profiles with a line.
    """
    df = summary_df.copy()

    # drop NaNs
    df = df.dropna(subset=[feature_col])

    # pivot: rows = trialid, columns = profiles
    wide = df.pivot_table(
        index='trialid',
        columns='profile',
        values=feature_col
    )

    # reorder columns to match desired profile order
    wide = wide.reindex(columns=profile_order)

    x = np.arange(len(profile_order))

    plt.figure(figsize=(10, 6))
```

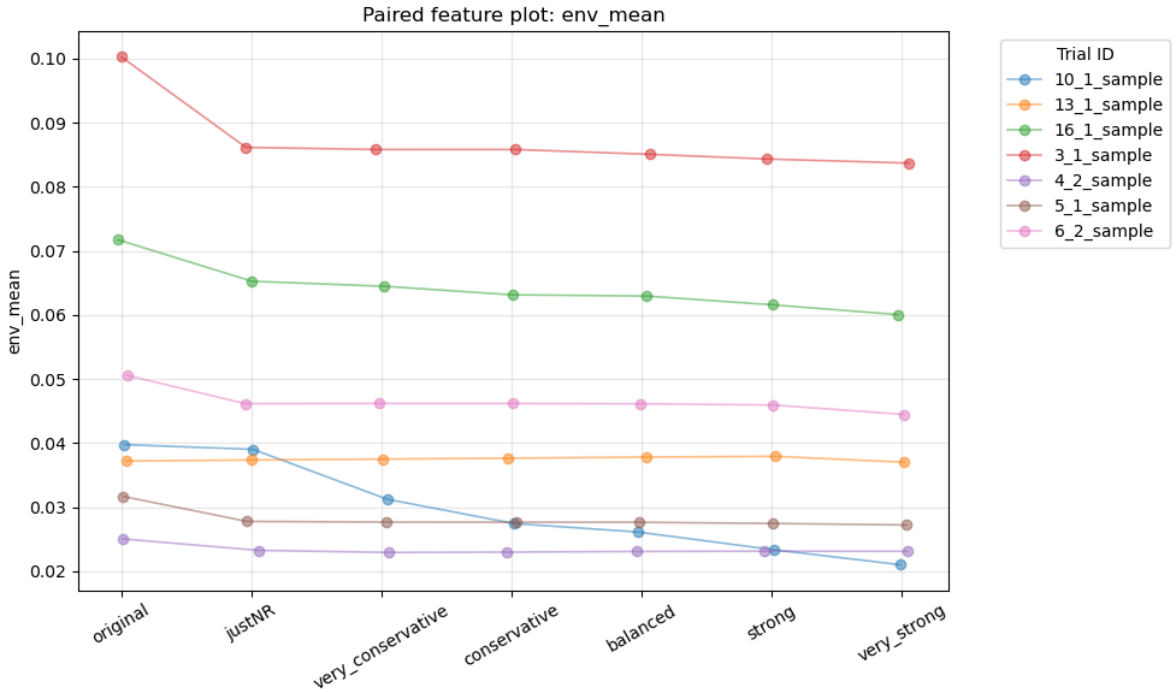
```
    # plot one line per trial
    for trialid, row in wide.iterrows():
        y = row.values
        # mask NaNs
        mask = ~np.isnan(y)
        if mask.sum() < 2:
            continue
        # small jitter to help see overlapping points
        jitter_x = x + np.random.normal(scale=0.04, size=len(x))
        plt.plot(jitter_x[mask], y[mask],
                 marker='o', alpha=0.5, linewidth=1.2)

    plt.xticks(x, profile_order, rotation=30)
    plt.ylabel(feature_col)
    plt.title(f"Paired feature plot: {feature_col}")
    # add legend for trialid
    plt.legend(wide.index, title="Trial ID", bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.grid(alpha=0.3)
    plt.tight_layout()
    plt.show()
```

```
plot_paired_feature(summary_df, "f1_mean")
plot_paired_feature(summary_df, "f2_mean")
plot_paired_feature(summary_df, "f0_mean")
plot_paired_feature(summary_df, "env_mean")
```


Paired feature plot: f1_mean
f1_mean
920
900
880
860
840
820
800
780
original
justNR
very_conservative
conservative
balanced
strong
very_strong
Trial ID
10_1_sample
13_1_sample
16_1_sample
3_1_sample
4_2_sample
5_1_sample
6_2_sample


Paired feature plot: f2_mean
f2_mean
2200
2150
2100
2050
2000
1950
1900
original
justNR
very_conservative
conservative
balanced
strong
very_strong
Trial ID
10_1_sample
13_1_sample
16_1_sample
3_1_sample
4_2_sample
5_1_sample
6_2_sample


Paired feature plot: f0_mean
f0_mean
240
220
200
180
160
original
justNR
very_conservative
conservative
balanced
strong
very_strong
Trial ID
10_1_sample
13_1_sample
16_1_sample
3_1_sample
4_2_sample
5_1_sample
6_2_sample


Paired feature plot: env_mean
env_mean
0.10
0.09
0.08
0.07
0.06
0.05
0.04
0.03
0.02
original
justNR
very_conservative
conservative
balanced
strong
very_strong
Trial ID
10_1_sample
13_1_sample
16_1_sample
3_1_sample
4_2_sample
5_1_sample
6_2_sample

Full test of denoising profiles

We will do three tests across the profiles

1. Bias in acoustic features: How much do the denoised features deviate from the original (noisy) features? For each mean of a feature (f0, F1, F2, F3, amplitude), we compute the deviation from the original. This tells us how much the average shifts with each denoising profile. If these shifts are large or systematic, we can conclude that the profile is significantly distorting or biasing the features.

2. Preservation of acoustic shape: How well do the denoised features preserve the overall shape of the original features? For F1 and F2, we compute the pairwise distance between all samples in the original and denoised data. Then we compute the correlation between the two distance matrices. The closer the correlation is to 1, the better the shape is preserved. Very low valuers of r would reveal serious distortion.

If one works with speech, it is also possible to compare the denoised samples to a fixed vowel targets

3. Stability and consistency: How noise reduction affects variability? In this test, we evaluate the standard deviation of formants, f0 and amplitude. We also compare the percentage of voiced frames. Here, we look whether noise removal largely increases or decreases variability. If a profile introduces massive formant jitter or kills voiced frames, it becomes unusable.

In sum, test 1 ensures that global feature values remain as similar as possible; test 2 ensures that samples remain comparable to each other; and test 3 ensures that temporal structure and vocal dynamics are preserved.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy.spatial.distance import pdist, squareform
from scipy import stats

PROFILE_ORDER = [
    "original",
    "justNR",
    "very_conservative",
    "conservative",
    "balanced",
    "strong",
    "very_strong",
]

# Test 1: global mean bias
```

```
def compute_bias_table(summary_df, feature_cols, profile_order=PROFILE_ORDER):
    """
    Compute bias = feature(profile) - feature(original) for each trial & profile.
    Returns long-form table with columns:
        trialid, profile, feature, delta
    and a summary table with mean/sd of delta per profile & feature.
    """
    df = summary_df.copy()
    results_long = []

    for feat in feature_cols:
        if feat not in df.columns:
            print(f"[Test1] Skipping feature {feat}: not in summary_df")
            continue

        # pivot to wide: index = trialid, columns = profile
        wide = df.pivot_table(index="trialid", columns="profile", values=feat)
        if "original" not in wide.columns:
            raise ValueError("No 'original' profile found in summary_df for Test 1.")

        # subtract original from each profile
        deltas = wide.sub(wide["original"], axis=0)

        # melt to long
        long = (
            deltas.reset_index()
                  .melt(id_vars="trialid", var_name="profile", value_name="delta")
        )
        # drop original itself (delta is 0) if you want
        long = long[long["profile"] != "original"]

        long["feature"] = feat
        results_long.append(long)

    if not results_long:
        raise ValueError("No valid features found in Test 1.")

    long_all = pd.concat(results_long, ignore_index=True)

    # summary stats
    summary = (
        long_all.groupby(["feature", "profile"])["delta"]
```

```
                .agg(["mean", "std", "count"])
                .reset_index()
    )
    summary = summary.sort_values(["feature", "profile"])

    return long_all, summary


def plot_bias_paired(summary_df, feature_col, profile_order=PROFILE_ORDER):
    """
    Paired lines for feature across profiles, plus mean trend.
    Uses absolute values (not deltas).
    """
    df = summary_df.copy()
    if feature_col not in df.columns:
        raise ValueError(f"{feature_col} not in summary_df.")

    df = df.dropna(subset=[feature_col])

    wide = df.pivot_table(index="trialid", columns="profile", values=feature_col)
    # keep only profiles that exist
    profs = [p for p in profile_order if p in wide.columns]
    wide = wide.reindex(columns=profs)

    x = np.arange(len(profs))

    plt.figure(figsize=(10, 6))

    # individual lines
    for trialid, row in wide.iterrows():
        y = row.values.astype(float)
        m = ~np.isnan(y)
        if m.sum() < 2:
            continue
        plt.plot(x[m], y[m], alpha=0.3, color="gray", linewidth=1.0)

    # mean trend
    means = wide.mean(axis=0, skipna=True)
    plt.plot(x, means.values, marker="o", color="red", linewidth=3, label="mean")

    plt.xticks(x, profs, rotation=30)
    plt.ylabel(feature_col)
```

```
    plt.title(f"Test 1 - Global bias: {feature_col} across profiles")
    plt.grid(alpha=0.3)
    plt.legend()
    plt.tight_layout()
    plt.show()


def plot_bias_deltas(long_bias, feature_col, profile_order=PROFILE_ORDER):
    """
    Plot deltas (profile - original) as paired dots per trial + mean ± SD per profile.
    """
    df = long_bias[long_bias["feature"] == feature_col].copy()
    profs = [p for p in profile_order if p in df["profile"].unique()]
    x = np.arange(len(profs))

    plt.figure(figsize=(10, 6))
    plt.axhline(0, color="black", linestyle="--", linewidth=1)

    # individual dots with vertical pairing
    for i, prof in enumerate(profs):
        vals = df.loc[df["profile"] == prof, "delta"].values
        jitter_x = np.random.normal(loc=i, scale=0.04, size=len(vals))
        plt.scatter(jitter_x, vals, s=15, alpha=0.5, color="gray")

    # means + SD
    stats_df = (
        df.groupby("profile")["delta"]
          .agg(["mean", "std"])
          .reindex(profs)
    )

    plt.errorbar(
        x,
        stats_df["mean"].values,
        yerr=stats_df["std"].values,
        fmt="-o",
        color="red",
        capsize=4,
        label="mean ± SD",
    )

    plt.xticks(x, profs, rotation=30)
```

```
    plt.ylabel(f"{feature_col} (profile - original)")
    plt.title(f"Test 1 - Global bias in {feature_col}")
    plt.grid(alpha=0.3)
    plt.legend()
    plt.tight_layout()
    plt.show()


# Test 2: shape preservation (distance geometry)

def _pairwise_dist_matrix(X):
    """
    X: N x D matrix
    returns N x N pairwise Euclidean distance matrix
    """
    D = squareform(pdist(X, metric="euclidean"))
    return D

def compute_shape_correlation(summary_df, features, profile_order=PROFILE_ORDER):
    """
    For each non-original profile:
      - build feature matrix of (trialid x features)
      - compute pairwise distance matrix
      - correlate with original's distance matrix
    Returns a DataFrame with columns:
      profile, r, n_trials
    """
    df = summary_df.copy()

    # restrict to rows with all requested features
    df = df.dropna(subset=features + ["profile", "trialid"])

    # original matrix
    base = df[df["profile"] == "original"].copy()
    if base.empty:
        raise ValueError("No 'original' profile found in summary_df for Test 2.")

    # pivot to trial x features
    base_mat = (
        base.groupby("trialid")[features]
            .mean()
    )
```

```
    base_trials = base_mat.index.to_list()
    X_base = base_mat.values
    D_base = _pairwise_dist_matrix(X_base)

    rows = []

    for prof in profile_order:
        if prof == "original":
            continue
        sub = df[df["profile"] == prof].copy()
        if sub.empty:
            continue

        # pivot and align to the same trialids as original
        mat_prof = (
            sub.groupby("trialid")[features]
               .mean()
        )

        # intersect trialids
        common_trials = base_mat.index.intersection(mat_prof.index)
        if len(common_trials) < 4:
            # not enough overlapping trials
            rows.append({"profile": prof, "r": np.nan, "n_trials": len(common_trials)})
            continue

        X0 = base_mat.loc[common_trials].values
        Xp = mat_prof.loc[common_trials].values

        D0 = _pairwise_dist_matrix(X0)
        Dp = _pairwise_dist_matrix(Xp)

        # correlate upper triangle
        iu = np.triu_indices_from(D0, k=1)
        r = np.corrcoef(D0[iu], Dp[iu])[0, 1]

        rows.append({"profile": prof, "r": r, "n_trials": len(common_trials)})

    return pd.DataFrame(rows)


def plot_shape_correlation(shape_df):
```

```
    """
    Simple barplot of distance-matrix correlation per profile (Test 2).
    """
    shape_df = shape_df.copy().dropna(subset=["r"])
    if shape_df.empty:
        print("No valid correlations to plot in Test 2.")
        return

    x = np.arange(len(shape_df))
    plt.figure(figsize=(6, 4))
    plt.bar(x, shape_df["r"].values, alpha=0.7)
    plt.xticks(x, shape_df["profile"].values, rotation=30)
    plt.ylim(0, 1.05)
    plt.ylabel("Correlation r")
    plt.title("Test 2 - Shape preservation (distance geometry)")
    plt.grid(axis="y", alpha=0.3)
    plt.tight_layout()
    plt.show()


# Test 3: stability / consistency (variability, voiced %)

def compute_stability_deltas(summary_df,
                             variability_features=None,
                             profile_order=PROFILE_ORDER):
    """
    Compare variability-related features between profiles and original.
    E.g. F1_sd, F2_sd, F3_sd, f0_sd, env_slope_sd, f0_percent_voiced.
    Returns long and summary DataFrames similar to Test 1.
    """
    if variability_features is None:
        variability_features = [
            "f1_sd", "f2_sd", "f3_sd",
            "f0_sd",
            "env_sd", "env_slope_sd",
            "f0_percent_voiced",
        ]

    df = summary_df.copy()
    results_long = []

    for feat in variability_features:
```

```
        if feat not in df.columns:
            print(f"[Test3] Skipping {feat}: not in summary_df")
            continue

        df_feat = df.dropna(subset=[feat])
        wide = df_feat.pivot_table(index="trialid", columns="profile", values=feat)
        if "original" not in wide.columns:
            raise ValueError("No 'original' profile for Test 3.")

        deltas = wide.sub(wide["original"], axis=0)
        long = (
            deltas.reset_index()
                  .melt(id_vars="trialid", var_name="profile", value_name="delta")
        )
        long = long[long["profile"] != "original"]
        long["feature"] = feat
        results_long.append(long)

    if not results_long:
        raise ValueError("No variability features found for Test 3.")

    long_all = pd.concat(results_long, ignore_index=True)

    summary = (
        long_all.groupby(["feature", "profile"])["delta"]
                .agg(["mean", "std", "count"])
                .reset_index()
    )
    summary = summary.sort_values(["feature", "profile"])

    return long_all, summary


def plot_stability_feature(long_stab, feature_col, profile_order=PROFILE_ORDER):
    """
    Plot per-profile deltas for a stability feature (e.g. F1_sd, f0_sd, f0_percent_voiced).
    """
    df = long_stab[long_stab["feature"] == feature_col].copy()
    profs = [p for p in profile_order if p in df["profile"].unique()]
    x = np.arange(len(profs))

    plt.figure(figsize=(10, 6))
```

```
    plt.axhline(0, color="black", linestyle="--", linewidth=1)

    for i, prof in enumerate(profs):
        vals = df.loc[df["profile"] == prof, "delta"].values
        jitter_x = np.random.normal(loc=i, scale=0.04, size=len(vals))
        plt.scatter(jitter_x, vals, s=15, alpha=0.6, color="gray")

    stats_df = (
        df.groupby("profile")["delta"]
          .agg(["mean", "std"])
          .reindex(profs)
    )

    plt.errorbar(
        x,
        stats_df["mean"].values,
        yerr=stats_df["std"].values,
        fmt="-o",
        color="red",
        capsize=4,
        label="mean ± SD",
    )

    plt.xticks(x, profs, rotation=30)
    plt.ylabel(f"{feature_col} (profile - original)")
    plt.title(f"Test 3 - Stability: change in {feature_col}")
    plt.grid(alpha=0.3)
    plt.legend()
    plt.tight_layout()
    plt.show()


# Compile in one function

def run_all_tests(summary_df):
    # --- Test 1: global bias in means ---
    bias_features = ["env_mean", "f0_mean", "f1_mean", "f2_mean", "f3_mean"]
    bias_long, bias_summary = compute_bias_table(summary_df, bias_features)
    print("\n=== Test 1 - Global bias summary ===")
    print(bias_summary)

    # Example plots:
```

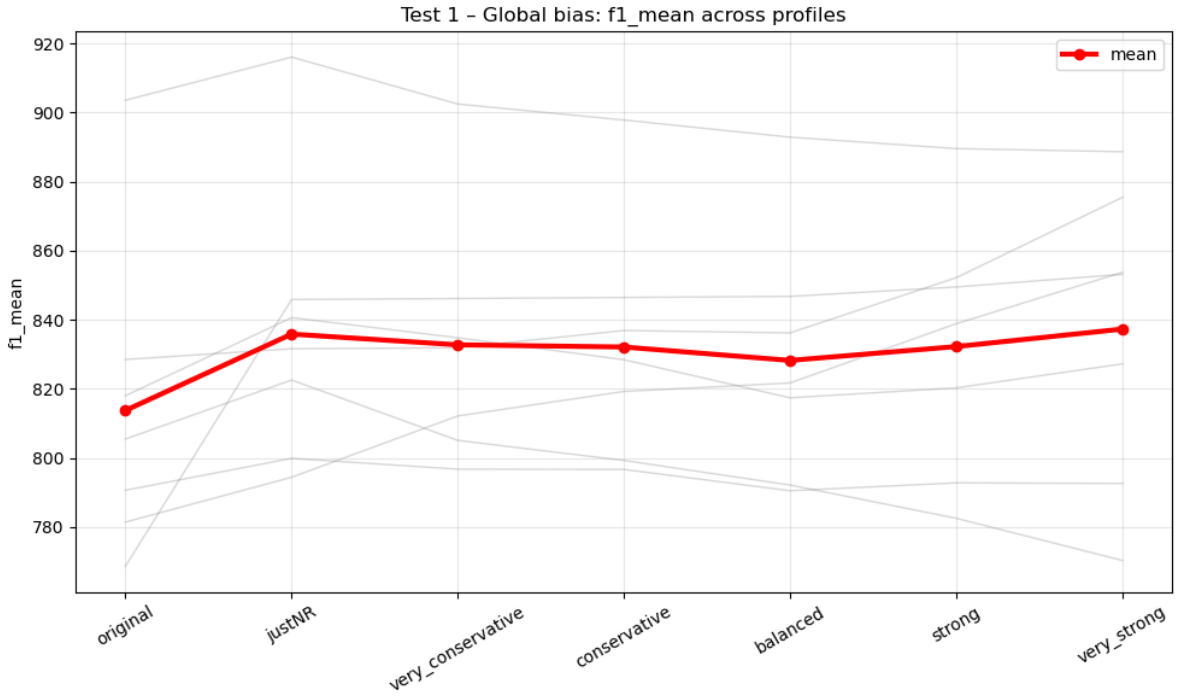
```
    plot_bias_paired(summary_df, "f1_mean")
    plot_bias_deltas(bias_long, "f1_mean")

    # --- Test 2: shape preservation in (F1_mean, F2_mean) space ---
    shape_df = compute_shape_correlation(summary_df, ["f1_mean", "f2_mean"])
    print("\n=== Test 2 - Shape preservation (F1_mean, F2_mean) ===")
    print(shape_df)
    plot_shape_correlation(shape_df)

    # --- Test 3: stability of variability and voicing ---
    stab_long, stab_summary = compute_stability_deltas(summary_df)
    print("\n=== Test 3 - Stability summary ===")
    print(stab_summary)

    # Example plots:
    # variability in f1_sd, f2_sd, f0_sd, and voiced %
    for feat in ["f1_sd", "f2_sd", "f0_sd", "f0_percent_voiced"]:
        if feat in stab_long["feature"].unique():
            plot_stability_feature(stab_long, feat)

    return {
        "bias_long": bias_long,
        "bias_summary": bias_summary,
        "shape_df": shape_df,
        "stability_long": stab_long,
        "stability_summary": stab_summary,
    }
```

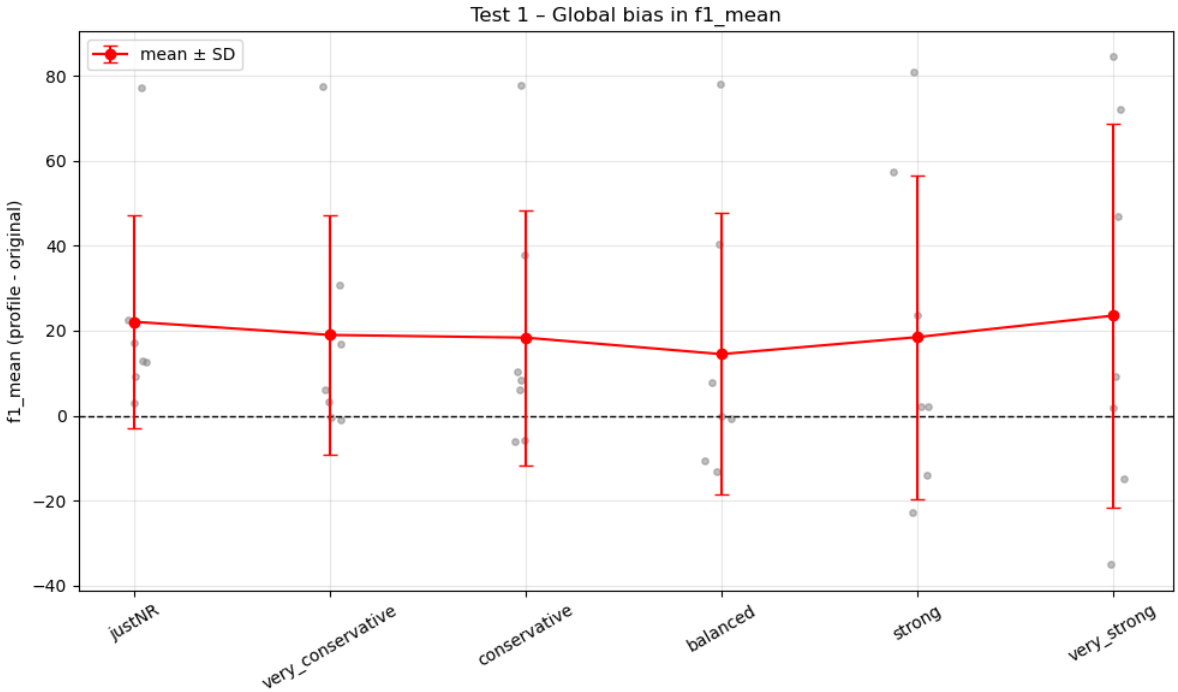
Results

```
results = run_all_tests(summary_df)
```

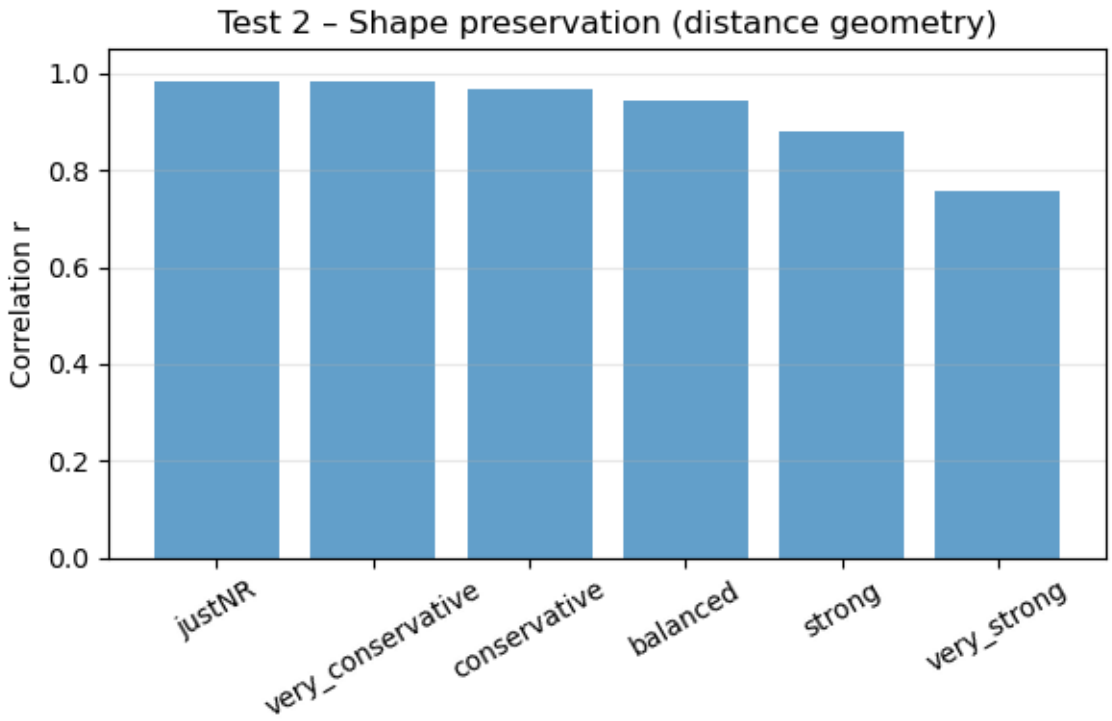
```
=== Test 1 - Global bias summary ===
     feature            profile        mean        std  count
0   env_mean           balanced   -0.006768   0.005957      7
1   env_mean       conservative   -0.006476   0.005476      7
2   env_mean             justNR   -0.004458   0.004832      7
3   env_mean             strong   -0.007494   0.006778      7
4   env_mean  very_conservative   -0.005770   0.004841      7
5   env_mean        very_strong   -0.008516   0.007240      7
```

6	f0_mean	balanced	2.559718	1.936643	7
7	f0_mean	conservative	2.564790	1.906171	7
8	f0_mean	justNR	2.599376	1.959150	7
9	f0_mean	strong	2.367686	1.752251	7
10	f0_mean	very_conservative	2.562959	1.922059	7
11	f0_mean	very_strong	2.427992	1.817000	7
12	f1_mean	balanced	14.481596	33.193869	7
13	f1_mean	conservative	18.367369	30.032737	7
14	f1_mean	justNR	22.104072	25.055028	7
15	f1_mean	strong	18.489886	38.157287	7
16	f1_mean	very_conservative	19.001013	28.154746	7
17	f1_mean	very_strong	23.555468	45.156269	7
18	f2_mean	balanced	24.504884	20.778027	7
19	f2_mean	conservative	22.581070	15.798284	7
20	f2_mean	justNR	13.429761	11.604336	7
21	f2_mean	strong	31.748902	31.220631	7
22	f2_mean	very_conservative	19.702669	11.092125	7
23	f2_mean	very_strong	46.073134	40.631383	7
24	f3_mean	balanced	60.663749	32.257688	7
25	f3_mean	conservative	54.959501	26.360058	7
26	f3_mean	justNR	23.408542	18.625770	7
27	f3_mean	strong	80.574423	48.419947	7
28	f3_mean	very_conservative	44.863761	16.818229	7
29	f3_mean	very_strong	109.487343	62.906073	7


Test 1 – Global bias: f1_mean across profiles
mean
f1_mean
920
900
880
860
840
820
800
780
original
justNR
very_conservative
conservative
balanced
strong
very_strong

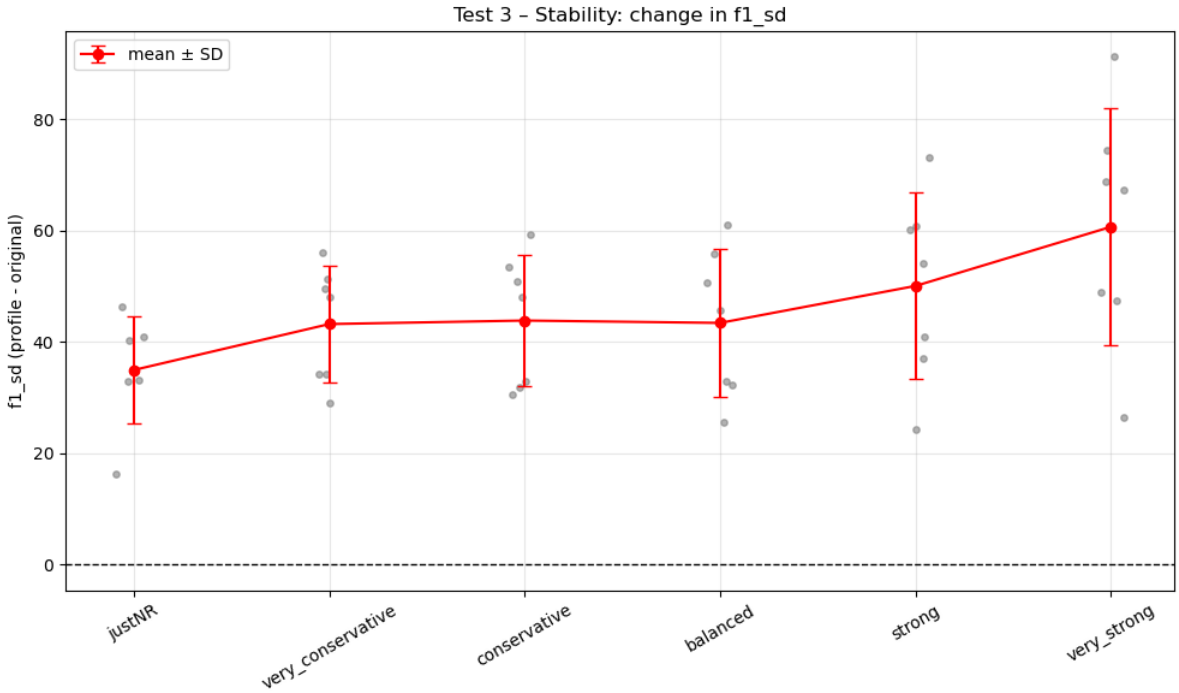

Test 1 – Global bias in f1_mean
mean ± SD
f1_mean (profile - original)
80
60
40
20
0
−20
−40
justNR
very_conservative
conservative
balanced
strong
very_strong

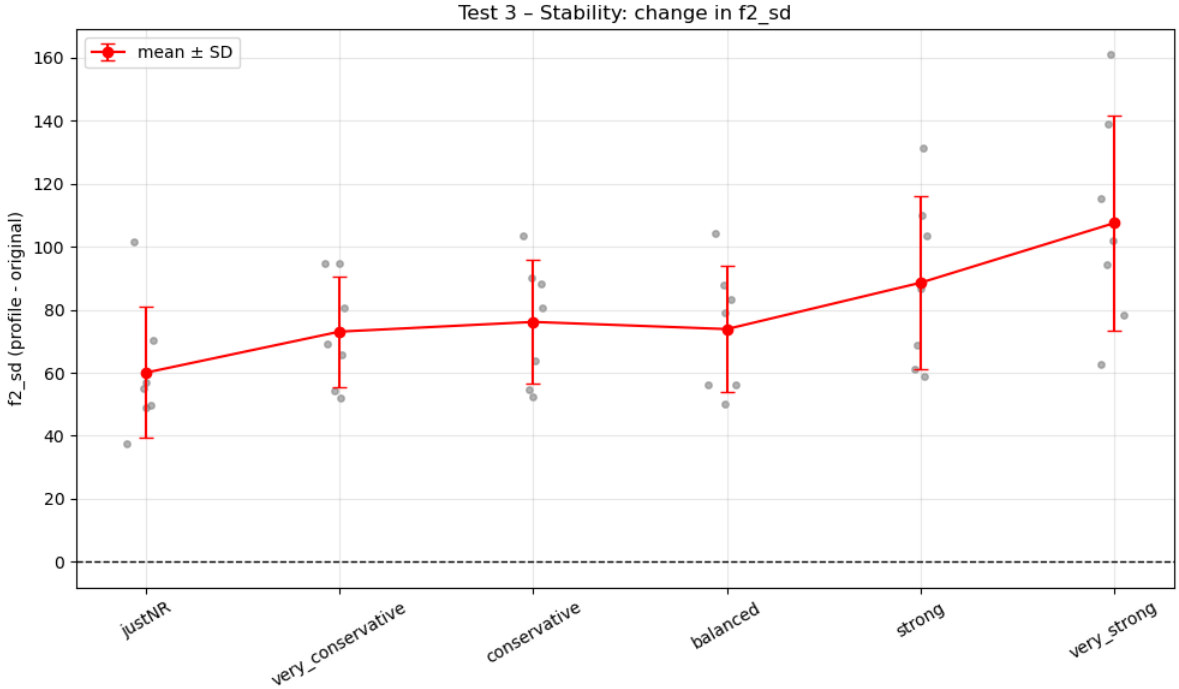
```
=== Test 2 - Shape preservation (F1_mean, F2_mean) ===
             profile         r  n_trials
0            justNR  0.985512         7
1  very_conservative  0.984835         7
2       conservative  0.967831         7
3           balanced  0.943017         7
4             strong  0.880396         7
5        very_strong  0.758274         7
```

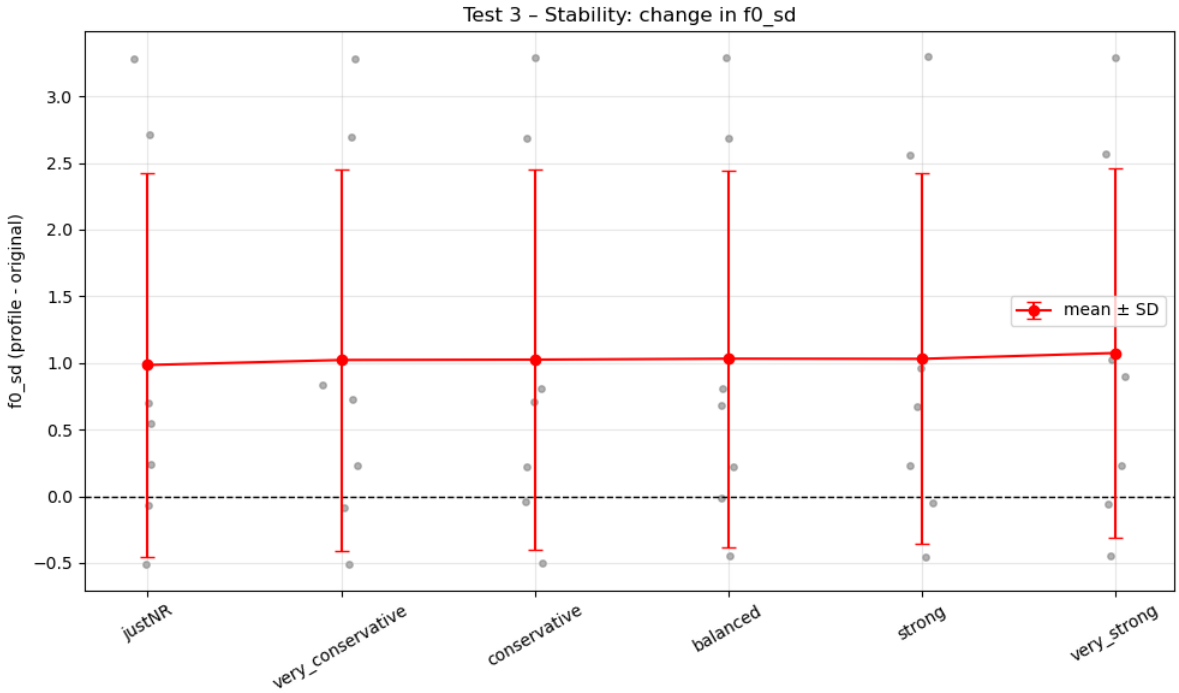


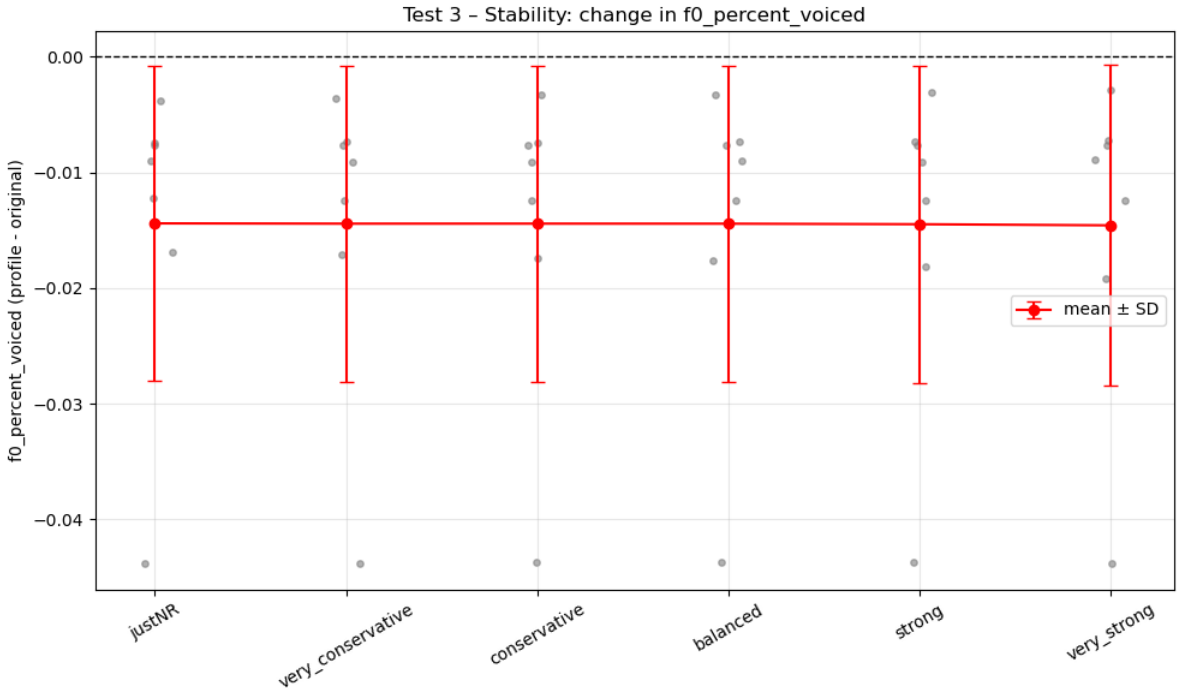

```
=== Test 3 - Stability summary ===
             feature            profile          mean           std  count
0             env_sd           balanced -3.057501e-03  5.236404e-03      7
1             env_sd       conservative -2.956665e-03  4.802161e-03      7
2             env_sd             justNR -3.107135e-03  4.365801e-03      7
3             env_sd             strong -3.088170e-03  5.650884e-03      7
4             env_sd  very_conservative -3.082286e-03  4.753843e-03      7
5             env_sd        very_strong -3.111511e-03  5.949848e-03      7
6        env_slope_sd           balanced -6.932875e-07  1.198613e-06      7
7        env_slope_sd       conservative -6.632894e-07  1.089905e-06      7
```

8	env_slope_sd	justNR	-6.720791e-07	9.911239e-07	7
9	env_slope_sd	strong	-7.537608e-07	1.298047e-06	7
10	env_slope_sd	very_conservative	-6.818537e-07	1.086247e-06	7
11	env_slope_sd	very_strong	-8.426174e-07	1.373723e-06	7
12	f0_percent_voiced	balanced	-1.443596e-02	1.368945e-02	7
13	f0_percent_voiced	conservative	-1.443127e-02	1.366371e-02	7
14	f0_percent_voiced	justNR	-1.440842e-02	1.363803e-02	7
15	f0_percent_voiced	strong	-1.448222e-02	1.373787e-02	7
16	f0_percent_voiced	very_conservative	-1.443351e-02	1.364137e-02	7
17	f0_percent_voiced	very_strong	-1.457852e-02	1.384923e-02	7
18	f0_sd	balanced	1.032295e+00	1.410819e+00	7
19	f0_sd	conservative	1.025131e+00	1.422855e+00	7
20	f0_sd	justNR	9.849238e-01	1.438775e+00	7
21	f0_sd	strong	1.031080e+00	1.391162e+00	7
22	f0_sd	very_conservative	1.022431e+00	1.427807e+00	7
23	f0_sd	very_strong	1.074769e+00	1.383592e+00	7
24	f1_sd	balanced	4.342068e+01	1.337026e+01	7
25	f1_sd	conservative	4.384610e+01	1.185328e+01	7
26	f1_sd	justNR	3.495798e+01	9.557112e+00	7
27	f1_sd	strong	5.006658e+01	1.672925e+01	7
28	f1_sd	very_conservative	4.321308e+01	1.050425e+01	7
29	f1_sd	very_strong	6.068316e+01	2.125511e+01	7
30	f2_sd	balanced	7.385880e+01	2.012930e+01	7
31	f2_sd	conservative	7.613559e+01	1.955393e+01	7
32	f2_sd	justNR	6.004736e+01	2.076466e+01	7
33	f2_sd	strong	8.856366e+01	2.746439e+01	7
34	f2_sd	very_conservative	7.305332e+01	1.760143e+01	7
35	f2_sd	very_strong	1.074819e+02	3.403830e+01	7
36	f3_sd	balanced	6.924413e+01	3.742038e+01	7
37	f3_sd	conservative	6.926540e+01	3.473599e+01	7
38	f3_sd	justNR	5.082027e+01	2.183084e+01	7
39	f3_sd	strong	8.677689e+01	4.798853e+01	7
40	f3_sd	very_conservative	6.559505e+01	2.982529e+01	7
41	f3_sd	very_strong	1.111769e+02	5.905221e+01	7


Test 3 – Stability: change in f1_sd
mean ± SD
f1_sd (profile - original)
0
20
40
60
80
justNR
very_conservative
conservative
balanced
strong
very_strong


Test 3 – Stability: change in f2_sd
mean ± SD
f2_sd (profile - original)
0
20
40
60
80
100
120
140
160
justNR
very_conservative
conservative
balanced
strong
very_strong


Test 3 – Stability: change in f0_sd
f0_sd (profile - original)
3.0
2.5
2.0
1.5
1.0
0.5
0.0
−0.5
mean ± SD
justNR
very_conservative
conservative
balanced
strong
very_strong


Test 3 – Stability: change in f0_percent_voiced
f0_percent_voiced (profile - original)
0.00
−0.01
−0.02
−0.03
−0.04
mean ± SD
justNR
very_conservative
conservative
balanced
strong
very_strong

```
bias_summary = results['bias_summary']
shape_df = results['shape_df']
stability_summary = results['stability_summary']
```

```
bias_summary.to_csv("bias_summary.csv")
shape_df.to_csv("shape_df.csv")
stability_summary.to_csv("stability_summary.csv")
```

Conclusion