Step 1: Import Libraries

```
import numpy as np
import matplotlib.pyplot as plt
import os
import librosa
import librosa.display
import soundfile as sf
from tqdm import tqdm
import json
```

Step 2: Settings for File Paths

```
# Input dataset directory (original IRMAS training data)
INPUT_DIR = r"E:\InstruNet-AI\data\IRMAS-TrainingData"

# Output directories
CLEAN_AUDIO_DIR = r"E:\InstruNet-AI\data\Processed Audio"
SPEC_DIR = r"E:\InstruNet-AI\data\Mel_Spectrograms"
LABEL_MAP_PATH = r"E:\InstruNet-AI\data\label_map.json"

# Audio settings
TARGET_SR = 16000          # Resample to 16 kHz
FIXED_DURATION = 3.0       # seconds
MIN_AMPLITUDE = 0.02       # threshold for silence trimming
N_MELS = 128               # Number of mel bands

os.makedirs(CLEAN_AUDIO_DIR, exist_ok=True)
os.makedirs(SPEC_DIR, exist_ok=True)
```

Step 3: Helper Functions

(a) Load audio

```
def load_audio(path, sr=TARGET_SR):
    y, orig_sr = librosa.load(path, sr=None, mono=False)
    return y, orig_sr
```

(b) Convert stereo → mono

```
def stereo_to_mono(audio):
    """Converts stereo audio [2, n] to mono [n] manually if needed."""
    if audio.ndim == 1:
```

```
        return audio  # already mono
    return np.mean(audio, axis=0)
```

(c) Resample to 16 kHz

```
def resample_audio(audio, orig_sr, target_sr=TARGET_SR):
    return librosa.resample(audio, orig_sr=orig_sr,
target_sr=target_sr)
```

(d) Normalize Audio

```
def peak_normalize(audio):
    peak = np.max(np.abs(audio))
    return audio / peak if peak != 0 else audio
```

(e) Trim Silence (Amplitude Thresholding)

```
def trim_silence(audio, thresh=0.02):
    idx = np.where(np.abs(audio) > thresh)[0]
    if len(idx) == 0:
        return audio  # silence only
    return audio[idx[0]: idx[-1]]
```

(f) Pad/Clip to Fixed Duration

```
def fix_duration(audio, sr=TARGET_SR, duration=FIXED_DURATION):
    target_len = int(sr * duration)
    cur_len = len(audio)

    if cur_len > target_len:
        return audio[:target_len]       # clipping
    else:
        pad_len = target_len - cur_len
        return np.pad(audio, (0, pad_len), mode='constant')  # padding
```

(g) Export Cleaned Audio

```
def save_audio(audio, sr, out_path):
    sf.write(out_path, audio, sr)
```

Step 4: Mel Spectrogram Generation + Save as Image

```
def generate_mel_spectrogram(audio, sr=TARGET_SR, n_mels=N_MELS,
out_path=None):
```

```
    S = librosa.feature.melspectrogram(y=audio, sr=sr, n_mels=n_mels)
    S_db = librosa.power_to_db(S, ref=np.max)

    if out_path:
        plt.figure(figsize=(3, 3))
        plt.axis("off")
        librosa.display.specshow(S_db, sr=sr, cmap='magma')
        plt.savefig(out_path, bbox_inches='tight', pad_inches=0)
        plt.close()

    return S_db
```

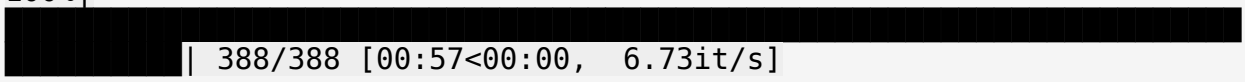
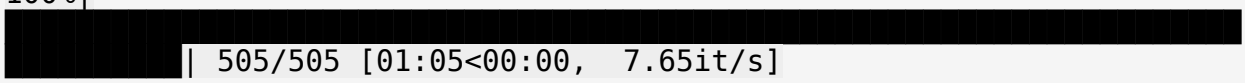
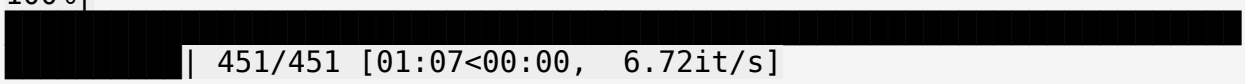
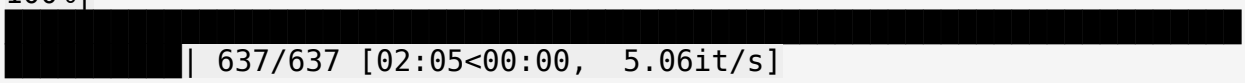
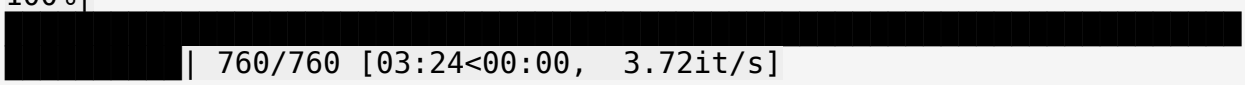
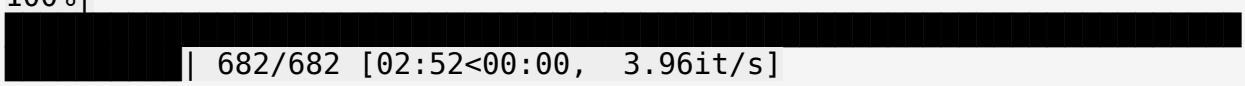
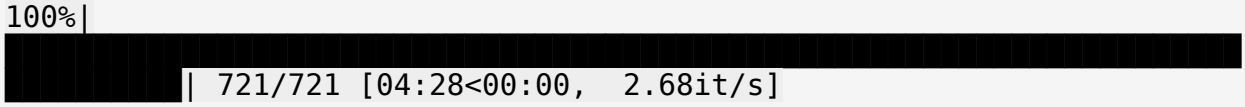
Step 5: Pipeline Wrapper (One File → Fully Processed Output)

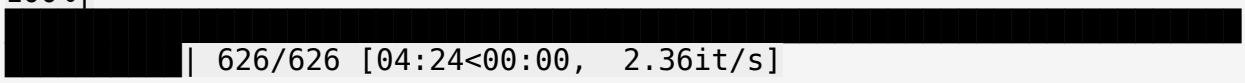
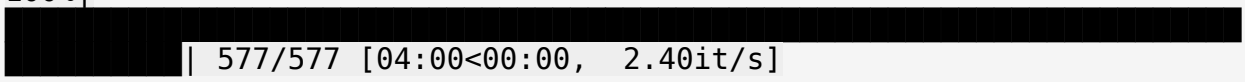
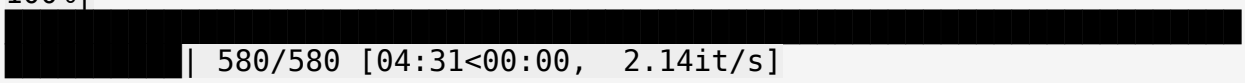
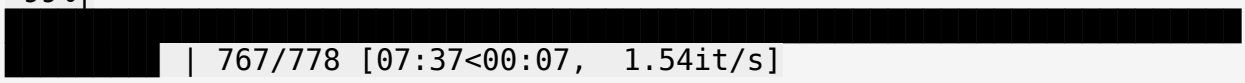
```
def preprocess_audio_file(
    file_path,
    out_audio_path,
    out_mel_img_path,
    label_mapping
):
    # (a) Load
    audio, sr = load_audio(file_path, sr=None)

    # (b) Stereo → Mono
    audio = stereo_to_mono(audio)

    # (c) Resample
    audio = resample_audio(audio, orig_sr=sr, target_sr=TARGET_SR)
    sr = TARGET_SR

    # (d) Peak normalize
    audio = peak_normalize(audio)

    # (e) Trim silence
    audio = trim_silence(audio)

    # (f) Fix duration to EXACT length
    audio = fix_duration(audio, sr, duration=FIXED_DURATION)

    # (g) Save audio
    save_audio(audio, sr, out_audio_path)

    # (h) Mel-spectrogram image
    generate_mel_spectrogram(audio, sr, out_path=out_mel_img_path)

    return label_mapping
```

Step 6: Main Processing Loop

```
# Initialize label mapping
label_map = {}
label_id = 0

# Loop over each instrument folder inside input directory
for class_name in os.listdir(INPUT_DIR):
    class_path = os.path.join(INPUT_DIR, class_name)
    if not os.path.isdir(class_path):
        continue

    print(f"\nProcessing class: {class_name}")
    label_map[label_id] = class_name

    # Prepare output directories for this class
    cleaned_class_dir = os.path.join(CLEAN_AUDIO_DIR, class_name)
    spec_class_dir = os.path.join(SPEC_DIR, class_name)

    os.makedirs(cleaned_class_dir, exist_ok=True)
    os.makedirs(spec_class_dir, exist_ok=True)

    # Process each .wav file inside the class folder
    for file in tqdm(os.listdir(class_path)):
        if not file.lower().endswith(".wav"):
            continue

        file_path = os.path.join(class_path, file)

        # Output file paths
        out_audio_path = os.path.join(cleaned_class_dir, file)
        out_mel_img_path = os.path.join(
            spec_class_dir,
            file.replace(".wav", ".png")
        )

        # Complete preprocessing pipeline
        preprocess_audio_file(
            file_path=file_path,
            out_audio_path=out_audio_path,
            out_mel_img_path=out_mel_img_path,
            label_mapping=label_map
        )

    label_id += 1

# Save label → class name mapping
with open(LABEL_MAP_PATH, "w") as f:
    json.dump(label_map, f, indent=4)
```

```
print("\nPreprocessing Completed Successfully.")
```

```
Processing class: cel
100%|
```

```
| 388/388 [00:57<00:00,  6.73it/s]

Processing class: cla
100%|
```

```
| 505/505 [01:05<00:00,  7.65it/s]

Processing class: flu
100%|
```

```
| 451/451 [01:07<00:00,  6.72it/s]

Processing class: gac
100%|
```

```
| 637/637 [02:05<00:00,  5.06it/s]

Processing class: gel
100%|
```

```
| 760/760 [03:24<00:00,  3.72it/s]

Processing class: org
100%|
```

```
| 682/682 [02:52<00:00,  3.96it/s]

Processing class: pia
100%|
```

```
| 721/721 [04:28<00:00,  2.68it/s]
```

```
Processing class: sax
100%|
```

```
| 626/626 [04:24<00:00,  2.36it/s]

Processing class: tru
100%|
```

```
| 577/577 [04:00<00:00,  2.40it/s]

Processing class: vio
100%|
```

```
| 580/580 [04:31<00:00,  2.14it/s]

Processing class: voi
 99%|
```

```
 | 767/778 [07:37<00:07,  1.54it/s]
```