```python
import os

# Path to your dataset root
dataset_root = r"E:\Publications\Paper 6\Datasets\New folder\LAV-DF\LAV-DF"

# Dictionary to store video files by split
video_files = {}

for split in ["train", "dev", "test"]:
    split_path = os.path.join(dataset_root, split)
    if os.path.exists(split_path):
        files = [f for f in os.listdir(split_path) if f.endswith(".mp4")]
        video_files[split] = files
        print(f"{split.upper()} videos: {len(files)} files found")
        print(f"First 5 videos in {split}: {files[:5]}")
    else:
        print(f"⚠️ {split} folder not found!")
```

```
TRAIN videos: 56776 files found
First 5 videos in train: ['000469.mp4', '000470.mp4', '000471.mp4', '000472.mp4', '000473.mp4']
DEV videos: 31501 files found
First 5 videos in dev: ['004053.mp4', '004054.mp4', '004055.mp4', '004056.mp4', '004057.mp4']
TEST videos: 26100 files found
First 5 videos in test: ['000000.mp4', '000001.mp4', '000002.mp4', '000003.mp4', '000004.mp4']
```

```python
import os

# Base path to save preprocessed data
preprocessed_base = r"E:\Publications\Paper 6\Datasets\Preprocessed"

# Create folders for each split and for faces/audio
for split in ["train", "dev", "test"]:
    faces_dir = os.path.join(preprocessed_base, split, "faces")
    audio_dir = os.path.join(preprocessed_base, split, "audio")
    os.makedirs(faces_dir, exist_ok=True)
    os.makedirs(audio_dir, exist_ok=True)
    print(f"✅ Created folders for {split}:")
    print(f"   Faces: {faces_dir}")
    print(f"   Audio: {audio_dir}")
```

```
✅ Created folders for train:
   Faces: E:\Publications\Paper 6\Datasets\Preprocessed\train\faces
   Audio: E:\Publications\Paper 6\Datasets\Preprocessed\train\audio
✅ Created folders for dev:
   Faces: E:\Publications\Paper 6\Datasets\Preprocessed\dev\faces
   Audio: E:\Publications\Paper 6\Datasets\Preprocessed\dev\audio
✅ Created folders for test:
   Faces: E:\Publications\Paper 6\Datasets\Preprocessed\test\faces
   Audio: E:\Publications\Paper 6\Datasets\Preprocessed\test\audio
```

```python
import cv2
import mediapipe as mp

# Mediapipe face detection
mp_face = mp.solutions.face_detection
mp_drawing = mp.solutions.drawing_utils

# Number of frames to extract per video (adjustable)
frames_per_video = 10

for split in ["train", "dev", "test"]:
    split_path = os.path.join(dataset_root, split)
    faces_dir = os.path.join(preprocessed_base, split, "faces")

    video_files = [f for f in os.listdir(split_path) if f.endswith(".mp4")]

    print(f"\nProcessing {split.upper()} videos: {len(video_files)} found")

    with mp_face.FaceDetection(model_selection=0, min_detection_confidence=0.5) as face_detector:
        for vid_file in video_files[:5]:  # For now, just process first 5 videos as test
            video_path = os.path.join(split_path, vid_file)
            try:
                cap = cv2.VideoCapture(video_path)
                frame_count = 0
                saved_count = 0
                while frame_count < frames_per_video:
                    ret, frame = cap.read()
```

```
                while frame_count < frames_per_video:
                    ret, frame = cap.read()
                    if not ret:
                        break
                    # Convert to RGB for mediapipe
                    rgb_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                    results = face_detector.process(rgb_frame)

                    if results.detections:
                        for i, detection in enumerate(results.detections):
                            bboxC = detection.location_data.relative_bounding_box
                            ih, iw, _ = frame.shape
                            x = int(bboxC.xmin * iw)
                            y = int(bboxC.ymin * ih)
                            w = int(bboxC.width * iw)
                            h = int(bboxC.height * ih)
                            face_crop = frame[y:y+h, x:x+w]

                            face_filename = f"{vid_file.replace('.mp4','')}_frame{frame_count}.jpg"
                            cv2.imwrite(os.path.join(faces_dir, face_filename), face_crop)
                            saved_count += 1
                    frame_count += 1
                cap.release()
                print(f"✅ Processed {vid_file}: {saved_count} face crops saved")
            except Exception as e:
                print(f"⚠️ Skipping {vid_file} due to error: {e}")
```

```
Processing TRAIN videos: 56776 found
⚠️ Skipping 000469.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 000470.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 000471.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 000472.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 000473.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'

Processing DEV videos: 31501 found
⚠️ Skipping 004053.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 004054.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 004055.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 004056.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 004057.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'

Processing TEST videos: 26100 found
⚠️ Skipping 000000.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 000001.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 000002.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 000003.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
⚠️ Skipping 000004.mp4 due to error: 'SymbolDatabase' object has no attribute 'GetPrototype'
```

```
!pip install librosa soundfile
```

```
Requirement already satisfied: librosa in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (0.11.0)
Requirement already satisfied: soundfile in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (0.13.1)
Requirement already satisfied: audioread>=2.1.9 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (3.0.1)
Requirement already satisfied: numba>=0.51.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (0.62.1)
Requirement already satisfied: numpy>=1.22.3 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (2.2.6)
Requirement already satisfied: scipy>=1.6.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.15.3)
Requirement already satisfied: scikit-learn>=1.1.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.7.2)
Requirement already satisfied: joblib>=1.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.5.2)
Requirement already satisfied: decorator>=4.3.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (5.2.1)
Requirement already satisfied: pooch>=1.1 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.8.2)
Requirement already satisfied: soxr>=0.3.2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.0.0)
Requirement already satisfied: typing_extensions>=4.1.1 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (4.15.0)
Requirement already satisfied: lazy_loader>=0.1 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (0.4)
Requirement already satisfied: msgpack>=1.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.1.2)
Requirement already satisfied: cffi>=1.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from soundfile) (2.0.0)
Requirement already satisfied: pycparser in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from cffi>=1.0->soundfile) (2.23)
Requirement already satisfied: packaging in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from lazy_loader>=0.1->librosa) (25.0)
Requirement already satisfied: llvmlite<0.46,>=0.45.0dev0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from numba>=0.51.0->librosa) (0.45.1)
Requirement already satisfied: platformdirs>=2.5.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from pooch>=1.1->librosa) (4.5.0)
Requirement already satisfied: requests>=2.19.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from pooch>=1.1->librosa) (2.32.5)
Requirement already satisfied: charset_normalizer<4,>=2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from requests>=2.19.0->pooch>=1.1->librosa) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from requests>=2.19.0->pooch>=1.1->librosa) (3.11)
Requirement already satisfied: urllib3<3,>=1.21.1 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from requests>=2.19.0->pooch>=1.1->librosa) (2.5.0)
Requirement already satisfied: certifi>=2017.4.17 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from requests>=2.19.0->pooch>=1.1->librosa) (2025.10.5)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from scikit-learn>=1.1.0->librosa) (3.6.0)
```

```
!pip install moviepy --upgrade
```

```
Requirement already satisfied: moviepy in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (2.2.1)
Requirement already satisfied: decorator<6.0,>=4.0.2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (5.2.1)
Requirement already satisfied: imageio<3.0,>=2.5 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (2.37.0)
Requirement already satisfied: imageio_ffmpeg>=0.2.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (0.6.0)
Requirement already satisfied: numpy>=1.25.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (2.2.6)
Requirement already satisfied: proglog<=1.0.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (0.1.12)
Requirement already satisfied: python-dotenv>=0.10 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (1.1.1)
Requirement already satisfied: pillow<12.0,>=9.2.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (11.3.0)
Requirement already satisfied: tqdm in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from proglog<=1.0.0->moviepy) (4.67.1)
Requirement already satisfied: colorama in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tqdm->proglog<=1.0.0->moviepy) (0.4.6)
```

```
import sys
!{sys.executable} -m pip install moviepy
```

```
Requirement already satisfied: moviepy in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (2.2.1)
Requirement already satisfied: decorator<6.0,>=4.0.2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (5.2.1)
Requirement already satisfied: imageio<3.0,>=2.5 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (2.37.0)
Requirement already satisfied: imageio_ffmpeg>=0.2.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (0.6.0)
Requirement already satisfied: numpy>=1.25.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (2.2.6)
Requirement already satisfied: proglog<=1.0.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (0.1.12)
Requirement already satisfied: python-dotenv>=0.10 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (1.1.1)
Requirement already satisfied: pillow<12.0,>=9.2.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from moviepy) (11.3.0)
Requirement already satisfied: tqdm in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from proglog<=1.0.0->moviepy) (4.67.1)
Requirement already satisfied: colorama in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tqdm->proglog<=1.0.0->moviepy) (0.4.6)
```

```
import sys
print(sys.executable)
```

```
C:\Users\HP\anaconda3\envs\colab-local\python.exe
```

```
import os

# Root folder where your extracted videos are
dataset_root = r"E:\Publications\Paper 6\Datasets\New folder\LAV-DF\LAV-DF"

# Base folder where preprocessed faces/audio will be saved
preprocessed_base = r"E:\Publications\Paper 6\Datasets\Preprocessed"

# Make sure preprocessed audio folders exist
for split in ["train", "dev", "test"]:
    audio_dir = os.path.join(preprocessed_base, split, "audio")
    os.makedirs(audio_dir, exist_ok=True)
```

```
import os
import subprocess
import librosa
import numpy as np

# Full path to ffmpeg executable
ffmpeg_path = r"C:\Users\HP\Downloads\ffmpeg-8.0-full_build\bin\ffmpeg.exe"

# Loop through each split
for split in ["train", "dev", "test"]:
    split_path = os.path.join(dataset_root, split)
    audio_dir = os.path.join(preprocessed_base, split, "audio")

    video_files = [f for f in os.listdir(split_path) if f.endswith(".mp4")]

    print(f"\nProcessing {split.upper()} videos for audio: {len(video_files)} found")

    for idx, vid_file in enumerate(video_files[:5]):  # first 5 videos per split for testing
        video_path = os.path.join(split_path, vid_file)
        temp_wav = os.path.join(audio_dir, "temp.wav")
        try:
            # Extract audio to WAV using full ffmpeg path
            cmd = f'"{ffmpeg_path}" -y -i "{video_path}" -vn -ac 1 -ar 22050 "{temp_wav}"'
            subprocess.run(cmd, shell=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

            # Load audio and compute MFCC
            y, sr = librosa.load(temp_wav, sr=22050)
            mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
```

```
            print(f"✅ {vid_file}: MFCC shape {mfcc.shape} saved")

            # Remove temporary WAV
            os.remove(temp_wav)
        except Exception as e:
            print(f"⚠️ Skipping {vid_file} due to error: {e}")
```

```
Processing TRAIN videos for audio: 56776 found
✅ 000469.mp4: MFCC shape (13, 218) saved
✅ 000470.mp4: MFCC shape (13, 254) saved
✅ 000471.mp4: MFCC shape (13, 224) saved
✅ 000472.mp4: MFCC shape (13, 260) saved
✅ 000473.mp4: MFCC shape (13, 177) saved

Processing DEV videos for audio: 31501 found
✅ 004053.mp4: MFCC shape (13, 328) saved
✅ 004054.mp4: MFCC shape (13, 342) saved
✅ 004055.mp4: MFCC shape (13, 334) saved
✅ 004056.mp4: MFCC shape (13, 340) saved
✅ 004057.mp4: MFCC shape (13, 249) saved

Processing TEST videos for audio: 26100 found
✅ 000000.mp4: MFCC shape (13, 199) saved
✅ 000001.mp4: MFCC shape (13, 180) saved
✅ 000002.mp4: MFCC shape (13, 204) saved
✅ 000003.mp4: MFCC shape (13, 185) saved
✅ 000004.mp4: MFCC shape (13, 202) saved
```

```
import os
import subprocess
import librosa
import numpy as np

# Full path to ffmpeg executable
ffmpeg_path = r"C:\Users\HP\Downloads\ffmpeg-8.0-full_build\bin\ffmpeg.exe"

# Loop through each split
for split in ["train", "dev", "test"]:
    split_path = os.path.join(dataset_root, split)
    audio_dir = os.path.join(preprocessed_base, split, "audio")

    video_files = [f for f in os.listdir(split_path) if f.endswith(".mp4")]

    print(f"\nProcessing {split.upper()} videos for audio: {len(video_files)} found")

    for idx, vid_file in enumerate(video_files):  # <-- all videos
        video_path = os.path.join(split_path, vid_file)
        temp_wav = os.path.join(audio_dir, "temp.wav")
        try:
            # Extract audio to WAV using full ffmpeg path
            cmd = f'"{ffmpeg_path}" -y -i "{video_path}" -vn -ac 1 -ar 22050 "{temp_wav}"'
            subprocess.run(cmd, shell=True, stdout=subprocess.DEVNULL, stderr=subprocess.DEVNULL)

            # Load audio and compute MFCC
            y, sr = librosa.load(temp_wav, sr=22050)
            mfcc = librosa.feature.mfcc(y=y, sr=sr, n_mfcc=13)
```

```
            # Save MFCC as .npy
            audio_out_path = os.path.join(audio_dir, vid_file.replace(".mp4", ".npy"))
            np.save(audio_out_path, mfcc)

            print(f"✅ {vid_file}: MFCC shape {mfcc.shape} saved")

            # Remove temporary WAV
            os.remove(temp_wav)
        except Exception as e:
            print(f"⚠️ Skipping {vid_file} due to error: {e}")
```

```
Streaming output truncated to the last 5000 lines.
✅ 086889.mp4: MFCC shape (13, 295) saved
✅ 086890.mp4: MFCC shape (13, 309) saved
✅ 086891.mp4: MFCC shape (13, 604) saved
✅ 086892.mp4: MFCC shape (13, 690) saved
✅ 086893.mp4: MFCC shape (13, 612) saved
✅ 086894.mp4: MFCC shape (13, 684) saved
✅ 086896.mp4: MFCC shape (13, 610) saved
✅ 086898.mp4: MFCC shape (13, 210) saved
✅ 086899.mp4: MFCC shape (13, 585) saved
✅ 086900.mp4: MFCC shape (13, 654) saved
✅ 086901.mp4: MFCC shape (13, 596) saved
✅ 086902.mp4: MFCC shape (13, 654) saved
✅ 086903.mp4: MFCC shape (13, 623) saved
✅ 086904.mp4: MFCC shape (13, 590) saved
✅ 086905.mp4: MFCC shape (13, 629) saved
✅ 086906.mp4: MFCC shape (13, 257) saved
✅ 086907.mp4: MFCC shape (13, 271) saved
```

```
✅ 135196.mp4: MFCC shape (13, 240) saved
✅ 135197.mp4: MFCC shape (13, 218) saved
✅ 135198.mp4: MFCC shape (13, 240) saved
✅ 135199.mp4: MFCC shape (13, 273) saved
✅ 135200.mp4: MFCC shape (13, 193) saved
✅ 135201.mp4: MFCC shape (13, 218) saved
✅ 135202.mp4: MFCC shape (13, 232) saved
✅ 135203.mp4: MFCC shape (13, 221) saved
✅ 135204.mp4: MFCC shape (13, 232) saved
✅ 135205.mp4: MFCC shape (13, 174) saved
✅ 135206.mp4: MFCC shape (13, 202) saved
✅ 135207.mp4: MFCC shape (13, 180) saved
✅ 135208.mp4: MFCC shape (13, 199) saved
✅ 135209.mp4: MFCC shape (13, 367) saved
```

```python
import os

base_path = r"E:\Publications\Paper 6\Datasets\Preprocessed"

for split in ["train", "dev", "test"]:
    audio_dir = os.path.join(base_path, split, "audio")
    num_files = len([f for f in os.listdir(audio_dir) if f.endswith(".npy")])
    print(f"{split.upper()} audio features: {num_files} .npy files found")
```

```
TRAIN audio features: 56775 .npy files found
DEV audio features: 31501 .npy files found
TEST audio features: 26100 .npy files found
```

```python
import os
import numpy as np
import cv2

# Update this path to your preprocessed base directory
preprocessed_base = r"E:\Publications\Paper 6\Datasets\Preprocessed"

def load_face_embedding(face_path):
    """Convert an image to a flattened embedding (placeholder)."""
    img = cv2.imread(face_path)
    if img is None:
        return None
    img = cv2.resize(img, (128, 128))
    return img.flatten() / 255.0  # normalize

def build_combined_features(split):
    face_dir = os.path.join(preprocessed_base, split, "faces")
    audio_dir = os.path.join(preprocessed_base, split, "audio")
    out_path = os.path.join(preprocessed_base, f"{split}_features.npz")

    combined = []
    face_files = sorted([f for f in os.listdir(face_dir) if f.endswith(".jpg")])
    audio_files = sorted([f for f in os.listdir(audio_dir) if f.endswith(".npy")])

    for i, face_file in enumerate(face_files[:50]):  # take 50 samples for testing
        audio_idx = i % len(audio_files)
        face_path = os.path.join(face_dir, face_file)
        audio_path = os.path.join(audio_dir, audio_files[audio_idx])
        face_emb = load_face_embedding(face_path)
        audio_emb = np.load(audio_path)
        if face_emb is not None:
            combined.append({"face": face_emb, "audio": audio_emb})

    np.savez(out_path, data=combined)
    print(f"✅ {split} combined feature file saved:", out_path)

# Build all three splits
for split in ["train", "dev", "test"]:
    build_combined_features(split)
```

```
✅ train combined feature file saved: E:\Publications\Paper 6\Datasets\Preprocessed\train_features.npz
✅ dev combined feature file saved: E:\Publications\Paper 6\Datasets\Preprocessed\dev_features.npz
✅ test combined feature file saved: E:\Publications\Paper 6\Datasets\Preprocessed\test_features.npz
```

```
!pip install pandas
```

```
Requirement already satisfied: pandas in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (2.3.3)
Requirement already satisfied: numpy>=1.22.4 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from pandas) (2.2.6)
Requirement already satisfied: python-dateutil>=2.8.2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from pandas) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from pandas) (2025.2)
Requirement already satisfied: six>=1.5 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from python-dateutil>=2.8.2->pandas) (1.17.0)
```

```python
import os

root = r"E:\Publications\Paper 6\Datasets"
for path, dirs, files in os.walk(root):
    if "metadata.json" in files:
        print("✅ Found metadata.json here:", path)
```

```
✅ Found metadata.json here: E:\Publications\Paper 6\Datasets\New folder\LAV-DF\LAV-DF
```

```python
import json
import pandas as pd
import os

# --- Paths ---
metadata_path = r"E:\Publications\Paper 6\Datasets\New folder\LAV-DF\LAV-DF\metadata.json"
output_csv_path = r"E:\Publications\Paper 6\Datasets\Preprocessed\metadata_clean.csv"

# --- Ensure output directory exists ---
os.makedirs(os.path.dirname(output_csv_path), exist_ok=True)

# --- Load JSON ---
with open(metadata_path, "r", encoding="utf-8") as f:
    metadata = json.load(f)

# --- Flatten JSON to DataFrame ---
df = pd.json_normalize(metadata)

# --- Save to CSV ---
df.to_csv(output_csv_path, index=False, encoding="utf-8")

print(f"✅ Metadata successfully preprocessed and saved to:\n{output_csv_path}")
```

```
✅ Metadata successfully preprocessed and saved to:
E:\Publications\Paper 6\Datasets\Preprocessed\metadata_clean.csv
```

```python
import pandas as pd

# Path to the cleaned CSV
csv_path = r"E:\Publications\Paper 6\Datasets\Preprocessed\metadata_clean.csv"

# Load the CSV
df = pd.read_csv(csv_path)

# Quick overview
print("Shape of the dataset:", df.shape)
print("\nColumns:")
print(df.columns.tolist())

# Display first 5 rows
print("\nFirst 5 rows:")
print(df.head())

# Optional: Check for missing values
print("\nMissing values per column:")
print(df.isnull().sum())
```

```
Shape of the dataset: (136304, 13)

Columns:
['file', 'n_fakes', 'fake_periods', 'timestamps', 'duration', 'transcript', 'original', 'modify_video', 'modify_audio', 'split', 'video_frames', 'audio_channels', 'audio_frames']

First 5 rows:
             file  n_fakes    fake_periods  \
0  test/000001.mp4        0              []
1  test/000000.mp4        0              []
2  test/000002.mp4        1  [[2.5, 3.224]]
3  test/000003.mp4        1   [[2.5, 2.78]]
4  test/000004.mp4        1  [[2.5, 3.204]]

                                      timestamps  duration  \
0  [['not', 0.0, 0.2], ['the', 0.2, 0.4], ['point...     4.224
1  [['I', 0.0, 0.1], ['side', 0.1, 0.4], ['with',...     4.672
2  [['not', 0.0, 0.2], ['the', 0.2, 0.4], ['point...     4.800
3  [['not', 0.0, 0.2], ['the', 0.2, 0.4], ['point...     4.352
4  [['not', 0.0, 0.2], ['the', 0.2, 0.4], ['point...     4.736

                                      transcript         original  \
0  not the point the point is that she died satis...          NaN
1  I side with what she's done with this gift of ...          NaN
2  not the point the point is that she be_born sa...  test/000001.mp4
3  not the point the point is that she be_born sa...  test/000001.mp4
```

```
4  not the point the point is that she be born sa...  test/000001.mp4

   modify_video  modify_audio split  video_frames  audio_channels  \
0         False         False  test           103               1
1         False         False  test           115               1
2          True          True  test           116               1
3          True         False  test           105               1
4         False          True  test           114               1

   audio_frames
0         65536
1         72704
2         74752
3         67584
4         73728

Missing values per column:
file                0
n_fakes             0
fake_periods        0
timestamps          0
duration            0
transcript          0
original        36431
modify_video        0
modify_audio        0
split               0
video_frames        0
audio_channels      0
audio_frames        0
dtype: int64
```

```
# For Colab or pip environment
!pip install mtcnn opencv-python-headless tqdm librosa soundfile
```

```
Requirement already satisfied: mtcnn in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (1.0.0)
Requirement already satisfied: opencv-python-headless in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (4.12.0.88)
Requirement already satisfied: tqdm in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (4.66.4)
Requirement already satisfied: librosa in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (0.10.1)
Requirement already satisfied: soundfile in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (0.13.1)
Requirement already satisfied: joblib>=1.4.2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from mtcnn) (1.5.2)
Requirement already satisfied: lz4>=4.3.3 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from mtcnn) (4.4.4)
Requirement already satisfied: numpy<2.3.0,>=2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from opencv-python-headless) (2.2.6)
Requirement already satisfied: colorama in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tqdm) (0.4.6)
Requirement already satisfied: audioread>=2.1.9 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (3.0.1)
Requirement already satisfied: scipy>=1.2.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.15.3)
Requirement already satisfied: scikit-learn>=0.20.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.5.2)
Requirement already satisfied: decorator>=4.3.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (5.2.1)
Requirement already satisfied: numba>=0.51.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (0.62.1)
Requirement already satisfied: pooch>=1.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.8.2)
Requirement already satisfied: soxr>=0.3.2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.0.0)
Requirement already satisfied: typing-extensions>=4.1.1 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (4.15.0)
Requirement already satisfied: lazy-loader>=0.1 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (0.4)
Requirement already satisfied: msgpack>=1.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from librosa) (1.1.2)
Requirement already satisfied: cffi>=1.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from soundfile) (2.0.0)
Requirement already satisfied: pycparser in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from cffi>=1.0->soundfile) (2.23)
Requirement already satisfied: packaging in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from lazy-loader>=0.1->librosa) (25.0)
Requirement already satisfied: llvmlite<0.46,>=0.45.0dev0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from numba>=0.51.0->librosa) (0.45.1)
Requirement already satisfied: platformdirs>=2.5.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from pooch>=1.0->librosa) (4.5.0)
Requirement already satisfied: requests>=2.19.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from pooch>=1.0->librosa) (2.32.5)
Requirement already satisfied: charset_normalizer<4,>=2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (3.4.4)
Requirement already satisfied: idna<4,>=2.5 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from requests>=2.19.0->pooch>=1.0->librosa) (3.11)
```

```
pip uninstall -y tensorflow tensorflow-cpu tensorflow-gpu
```

```
Found existing installation: tensorflow 2.13.0
Uninstalling tensorflow-2.13.0:
  Successfully uninstalled tensorflow-2.13.0
Note: you may need to restart the kernel to use updated packages.
WARNING: Skipping tensorflow-cpu as it is not installed.
WARNING: Skipping tensorflow-gpu as it is not installed.
```

```
pip install tensorflow==2.13.0
```

```
Collecting tensorflow==2.13.0
  Using cached tensorflow-2.13.0-cp310-cp310-win_amd64.whl.metadata (2.6 kB)
Requirement already satisfied: tensorflow-intel==2.13.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow==2.13.0) (2.13.0)
Requirement already satisfied: absl-py>=1.0.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (2.3.1)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (1.6.3)
Requirement already satisfied: flatbuffers>=23.1.21 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (25.9.23)
Requirement already satisfied: gast<=0.4.0,>=0.2.1 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (0.4.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (0.2.0)
Requirement already satisfied: h5py>=2.9.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (3.15.0)
Requirement already satisfied: libclang>=13.0.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (18.1.1)
Requirement already satisfied: numpy<=1.24.3,>=1.22 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (1.24.3)
Requirement already satisfied: opt-einsum>=2.3.2 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (3.4.0)
Requirement already satisfied: packaging in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (25.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from ter
Requirement already satisfied: setuptools in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (80.9.0)
Requirement already satisfied: six>=1.12.0 in c:\users\hp\anaconda3\envs\colab-local\lib\site-packages (from tensorflow-intel==2.13.0->tensorflow==2.13.0) (1.17.0)
```

```python
import sys
print(sys.executable)   # Make sure this is the correct Python environment

import tensorflow as tf
print("TensorFlow version:", tf.__version__)
print("tf.keras available:", hasattr(tf, 'keras'))
```

```
C:\Users\HP\anaconda3\envs\colab-local\python.exe
TensorFlow version: 2.13.0
tf.keras available: True
```

```python
# --- Step 1: Import required libraries ---
import os
import cv2
import pandas as pd
from mtcnn import MTCNN
from tqdm import tqdm
import numpy as np

# --- Verify versions ---
import tensorflow as tf
print("TensorFlow version:", tf.__version__)
print("tf.keras available:", hasattr(tf, 'keras'))
print("MTCNN imported successfully!")
```

```
TensorFlow version: 2.13.0
```

```python
# Paths
dataset_csv = r"E:\Publications\Paper 6\Datasets\Preprocessed\metadata_clean.csv"
video_root = r"E:\Publications\Paper 6\Datasets\New folder\LAV-DF\LAV-DF"
output_root = r"E:\Publications\Paper 6\Datasets\Preprocessed\Faces_Extracted"

os.makedirs(output_root, exist_ok=True)

# Load metadata
metadata = pd.read_csv(dataset_csv)

# Initialize MTCNN
detector = MTCNN()

# Function to extract faces with frame skipping
def extract_faces_fast(video_path, output_dir, frame_skip=30):
    """
    frame_skip=30 will process roughly 1 frame per second if video is 30 FPS.
    """
    cap = cv2.VideoCapture(video_path)
    frame_idx = 0
    saved_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        if frame_idx % frame_skip == 0:
            result = detector.detect_faces(frame)
            for i, face in enumerate(result):
                x, y, w, h = face['box']
                face_img = frame[y:y+h, x:x+w]
                face_filename = os.path.join(output_dir, f"frame{frame_idx}_face{i}.jpg")
                cv2.imwrite(face_filename, face_img)
                saved_count += 1
        frame_idx += 1
    cap.release()
    return saved_count

# --- Loop over all videos ---
from tqdm import tqdm

for idx, row in tqdm(metadata.iterrows(), total=len(metadata)):
    video_path = os.path.join(video_root, row['file'])
    output_dir = os.path.join(output_root, os.path.splitext(os.path.basename(row['file']))[0])
    os.makedirs(output_dir, exist_ok=True)
    extract_faces_fast(video_path, output_dir, frame_skip=30)  # adjust frame_skip as needed
```

```
96%|██████████████████████████████████████████████████ | 130254/136304 [24:17:41<2:05:32,  1.24s/it]
```

```python
import os
import pandas as pd

dataset_csv = r"E:\Publications\Paper 6\Datasets\Preprocessed\metadata_clean.csv"
output_root = r"E:\Publications\Paper 6\Datasets\Preprocessed\Faces_Extracted"

metadata = pd.read_csv(dataset_csv)
expected_total = len(metadata)
completed = len([d for d in os.listdir(output_root) if os.path.isdir(os.path.join(output_root, d))])

print(f"✅ Already completed: {completed:,}")
print(f"🔁 Remaining: {expected_total - completed:,}")
print(f"📊 Completion: {(completed / expected_total) * 100:.2f}%")
```

```
✅ Already completed: 131,171
🔁 Remaining: 5,133
📊 Completion: 96.23%
```

```python
import os
import cv2
import pandas as pd
from mtcnn import MTCNN
from tqdm import tqdm

# --- Paths ---
dataset_csv = r"E:\Publications\Paper 6\Datasets\Preprocessed\metadata_clean.csv"
video_root = r"E:\Publications\Paper 6\Datasets\New folder\LAV-DF\LAV-DF"
output_root = r"E:\Publications\Paper 6\Datasets\Preprocessed\Faces_Extracted"
min_frames_per_video = 3  # minimum number of face images to consider a video completed

os.makedirs(output_root, exist_ok=True)

# --- Load metadata ---
metadata = pd.read_csv(dataset_csv)

# --- Initialize MTCNN ---
detector = MTCNN()

# --- Function to extract faces with frame skipping ---
def extract_faces_fast(video_path, output_dir, frame_skip=30):
    cap = cv2.VideoCapture(video_path)
    frame_idx = 0
    saved_count = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            break
        if frame_idx % frame_skip == 0:
            result = detector.detect_faces(frame)
            for i, face in enumerate(result):
                x, y, w, h = face['box']
                face_img = frame[y:y+h, x:x+w]
                face_filename = os.path.join(output_dir, f"frame{frame_idx}_face{i}.jpg")
                cv2.imwrite(face_filename, face_img)
                saved_count += 1
        frame_idx += 1
    cap.release()
    return saved_count

# --- Resume face extraction ---
completed = 0
remaining_videos = []

for idx, row in metadata.iterrows():
    video_name = os.path.splitext(os.path.basename(row['file']))[0]
    output_dir = os.path.join(output_root, video_name)

    # Count number of frames saved
    frames_saved = len(os.listdir(output_dir)) if os.path.exists(output_dir) else 0

    if frames_saved >= min_frames_per_video:
        completed += 1
    else:
        remaining_videos.append((row['file'], output_dir))
```

```python
total_videos = len(metadata)
print(f"✅ Already completed videos: {completed}")
print(f"🔁 Remaining videos to process: {len(remaining_videos)}")

# --- Process remaining videos with progress bar ---
for i, (video_file, output_dir) in enumerate(tqdm(remaining_videos, desc="Resuming Face Extraction")):
    os.makedirs(output_dir, exist_ok=True)
    extract_faces_fast(os.path.join(video_root, video_file), output_dir, frame_skip=30)

    if (i+1) % 500 == 0 or (i+1) == len(remaining_videos):
        percent_done = (completed + i + 1) / total_videos * 100
        print(f"Completion: {percent_done:.2f}% ({completed + i + 1}/{total_videos})")
```

```
✅ Already completed videos: 106948
🔁 Remaining videos to process: 29356
Resuming Face Extraction:   2%|▏         | 500/29356 [05:05<4:19:13,  1.86it/s]Completion: 78.83% (107448/136304)
Resuming Face Extraction:   3%|▎         | 1000/29356 [09:27<4:13:23,  1.87it/s]Completion: 79.20% (107948/136304)
Resuming Face Extraction:   5%|▍         | 1500/29356 [14:02<3:07:01,  2.48it/s]Completion: 79.56% (108448/136304)
Resuming Face Extraction:   7%|▋         | 2000/29356 [18:31<3:05:36,  2.46it/s]Completion: 79.93% (108948/136304)
Resuming Face Extraction:   9%|▊         | 2500/29356 [23:41<4:59:04,  1.50it/s]Completion: 80.30% (109448/136304)
Resuming Face Extraction:  10%|▉         | 3000/29356 [29:11<3:17:22,  2.23it/s]Completion: 80.66% (109948/136304)
Resuming Face Extraction:  12%|█▏        | 3500/29356 [34:03<3:43:12,  1.93it/s]Completion: 81.03% (110448/136304)
Resuming Face Extraction:  15%|█▍        | 4437/29356 [37:54<01:13, 337.97it/s]Completion: 81.40% (110948/136304)
Completion: 81.76% (111448/136304)
Resuming Face Extraction:  19%|█▊        | 5462/29356 [42:06<00:57, 418.05it/s]Completion: 82.13% (111948/136304)
Completion: 82.50% (112448/136304)
Resuming Face Extraction:  22%|██▏       | 6421/29356 [42:07<00:15, 1439.89it/s]Completion: 82.86% (112948/136304)
Resuming Face Extraction:  24%|██▎       | 6924/29356 [42:09<00:49, 451.36it/s]Completion: 83.23% (113448/136304)
```

```
Resuming Face Extraction:  50%|                                  | 14804/29356 [42:19<00:07, 2010.34it/s]Completion: 89.10% (121448/136304)
Completion: 89.47% (121948/136304)
Resuming Face Extraction:  54%|                                  | 15915/29356 [42:20<00:05, 2532.83it/s]Completion: 89.83% (122448/136304)
Completion: 90.20% (122948/136304)
Resuming Face Extraction:  58%|                                  | 17020/29356 [42:44<02:20, 87.68it/s]Completion: 90.57% (123448/136304)
Completion: 90.93% (123948/136304)
Resuming Face Extraction:  61%|                                  | 17832/29356 [42:45<00:41, 278.84it/s]Completion: 91.30% (124448/136304)
Completion: 91.67% (124948/136304)
Resuming Face Extraction:  64%|                                  | 18870/29356 [42:45<00:11, 885.60it/s]Completion: 92.04% (125448/136304)
Completion: 92.40% (125948/136304)
Resuming Face Extraction:  68%|                                  | 19882/29356 [43:20<02:53, 54.75it/s]Completion: 92.77% (126448/136304)
Completion: 93.14% (126948/136304)
Resuming Face Extraction:  71%|                                  | 20888/29356 [43:29<01:02, 136.06it/s]Completion: 93.50% (127448/136304)
Resuming Face Extraction:  72%|                                  | 20996/29356 [43:45<04:15, 32.74it/s]Completion: 93.87% (127948/136304)
Resuming Face Extraction:  75%|                                  | 21982/29356 [43:55<01:06, 110.61it/s]Completion: 94.24% (128448/136304)
Completion: 94.60% (128948/136304)
Resuming Face Extraction:  78%|                                  | 23024/29356 [43:56<00:14, 436.38it/s]Completion: 94.97% (129448/136304)
Completion: 95.34% (129948/136304)
Resuming Face Extraction:  80%|                                  | 23553/29356 [43:56<00:07, 774.24it/s]Completion: 95.70% (130448/136304)
Resuming Face Extraction:  82%|                                  | 24067/29356 [44:12<01:11, 74.18it/s]Completion: 96.07% (130948/136304)
Resuming Face Extraction:  83%|                                 | 24500/29356 [48:49<1:23:04,  1.03s/it]Completion: 96.44% (131448/136304)
Resuming Face Extraction:  85%|                                 | 25000/29356 [55:32<50:10,  1.45it/s]Completion: 96.80% (131948/136304)
Resuming Face Extraction:  87%|                                 | 25500/29356 [1:01:56<48:44,  1.32it/s]Completion: 97.17% (132448/136304)
Resuming Face Extraction:  89%|                                 | 26000/29356 [1:09:16<36:12,  1.54it/s]Completion: 97.54% (132948/136304)
Resuming Face Extraction:  90%|                                 | 26500/29356 [1:15:19<36:16,  1.31it/s]Completion: 97.90% (133448/136304)
Resuming Face Extraction:  92%|                                 | 26949/29356 [1:16:15<00:13, 182.74it/s]Completion: 98.27% (133948/136304)
Resuming Face Extraction:  94%|                                 | 27500/29356 [1:20:35<19:53,  1.55it/s]Completion: 98.64% (134448/136304)
Resuming Face Extraction:  97%|                                 | 28594/29356 [1:22:22<00:01, 712.04it/s]Completion: 99.01% (134948/136304)
Completion: 99.37% (135448/136304)
Resuming Face Extraction: 100%|                                 | 29356/29356 [1:22:22<00:00,  5.94it/s]Completion: 99.74% (135948/136304)
Completion: 100.00% (136304/136304)
```

```python
import os
from tqdm import tqdm
import cv2

output_root = r"E:\Publications\Paper 6\Datasets\Preprocessed\Faces_Extracted"
video_folders = os.listdir(output_root)

total_videos = len(video_folders)
print(f"Total videos with extracted faces: {total_videos}")

# Count total face images
total_faces = 0
for folder in tqdm(video_folders, desc="Checking videos"):
    folder_path = os.path.join(output_root, folder)
    if os.path.isdir(folder_path):
        face_files = [f for f in os.listdir(folder_path) if f.endswith(".jpg")]
        total_faces += len(face_files)
        if len(face_files) == 0:
            print(f"⚠️ No faces found in video folder: {folder}")

print(f"Total extracted face images: {total_faces}")

# Optional: preview a few face images
import random
import matplotlib.pyplot as plt

sample_video = random.choice(video_folders)
```

```python
sample_video = random.choice(video_folders)
sample_folder = os.path.join(output_root, sample_video)
sample_faces = [f for f in os.listdir(sample_folder) if f.endswith(".jpg")]
sample_faces = random.sample(sample_faces, min(5, len(sample_faces)))

plt.figure(figsize=(12,3))
for i, face_file in enumerate(sample_faces):
    img = cv2.imread(os.path.join(sample_folder, face_file))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    plt.subplot(1,5,i+1)
    plt.imshow(img)
    plt.axis('off')
plt.show()
```

```
Total videos with extracted faces: 136305
```

```python
import os

output_root = r"E:\Publications\Paper 6\Datasets\Preprocessed\Faces_Extracted"
video_folders = [f for f in os.listdir(output_root) if os.path.isdir(os.path.join(output_root, f))]

total_videos = len(video_folders)
total_faces = 0

for folder in video_folders:
    folder_path = os.path.join(output_root, folder)
    face_files = [f for f in os.listdir(folder_path) if f.endswith(".jpg")]
    total_faces += len(face_files)

print(f"✅ Total videos processed: {total_videos}")
print(f"✅ Total face images extracted: {total_faces}")
```

```
✅ Total videos processed: 136305
✅ Total face images extracted: 835991
```

```python
import os
import pandas as pd

# Paths
dataset_csv = r"E:\Publications\Paper 6\Datasets\Preprocessed\metadata_clean.csv"
output_root = r"E:\Publications\Paper 6\Datasets\Preprocessed\Faces_Extracted"
```

```python
# Load metadata
metadata = pd.read_csv(dataset_csv)

# Check for videos with no faces extracted
no_face_videos = []

for idx, row in metadata.iterrows():
    video_name = os.path.splitext(os.path.basename(row['file']))[0]
    video_folder = os.path.join(output_root, video_name)

    if not os.path.exists(video_folder) or len(os.listdir(video_folder)) == 0:
        no_face_videos.append(row['file'])

print(f"Total videos with no faces extracted: {len(no_face_videos)}")
print("Sample videos with no faces:", no_face_videos[:10])
```

```
Total videos with no faces extracted: 23628
Sample videos with no faces: ['train/000580.mp4', 'train/000581.mp4', 'train/000583.mp4', 'train/000585.mp4', 'train/000589.mp4', 'train/000590.mp4', 'train/000595.mp4', 'train/000
```

```python
import os
import numpy as np
from tqdm import tqdm

# --- Paths ---
mfcc_root = r"E:\Publications\Paper 6\Datasets\Preprocessed\MFCC_Features"
```

```python
# --- Load MFCC files ---
mfcc_files = [f for f in os.listdir(mfcc_root) if f.endswith("_mfcc.npy")]
mfcc_files.sort()
print(f"Detected {len(mfcc_files)} MFCC files.")

# --- Find max frames ---
max_frames = 0
for file in mfcc_files:
    mfcc = np.load(os.path.join(mfcc_root, file))
    max_frames = max(max_frames, mfcc.shape[1])

print("Maximum frames in any MFCC:", max_frames)

# --- Assign dummy labels (for demonstration) ---
num_files = len(mfcc_files)
half = num_files // 2
labels = [0]*half + [1]*(num_files - half)

# --- Prepare feature matrix with padding ---
X, y = [], []
for i, file in enumerate(tqdm(mfcc_files)):
    path = os.path.join(mfcc_root, file)
    mfcc = np.load(path)  # shape: (13, n_frames)
    # Pad MFCC along time axis to max_frames
    if mfcc.shape[1] < max_frames:
        pad_width = max_frames - mfcc.shape[1]
        mfcc = np.pad(mfcc, ((0,0),(0,pad_width)), mode='constant')
    X.append(mfcc.flatten())
    if mfcc.shape[1] < max_frames:
        pad_width = max_frames - mfcc.shape[1]
        mfcc = np.pad(mfcc, ((0,0),(0,pad_width)), mode='constant')
    X.append(mfcc.flatten())
    y.append(labels[i])

X = np.array(X)
y = np.array(y)

print("Feature matrix shape:", X.shape)
print("Labels shape:", y.shape)
```

```
Detected 17 MFCC files.
Maximum frames in any MFCC: 249
100%|██████████| 17/17 [00:00<00:00, 1216.90it/s]Feature matrix shape: (17, 3237)
Labels shape: (17,)
```

```python
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# --- Split into train/test ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)
```

```
print("Train set:", X_train.shape, y_train.shape)
print("Test set:", X_test.shape, y_test.shape)

# --- Train baseline model ---
clf = LogisticRegression(max_iter=1000)
clf.fit(X_train, y_train)

# --- Predictions ---
y_pred = clf.predict(X_test)

# --- Evaluation ---
acc = accuracy_score(y_test, y_pred)
print(f"\nBaseline accuracy: {acc:.4f}\n")
print("Classification report:\n", classification_report(y_test, y_pred))
```

```
Train set: (13, 3237) (13,)
Test set: (4, 3237) (4,)

Baseline accuracy: 0.7500

Classification report:
              precision    recall  f1-score   support

           0       1.00      0.50      0.67         2
           1       0.67      1.00      0.80         2

    accuracy                           0.75         4
   macro avg       0.83      0.75      0.73         4
weighted avg       0.83      0.75      0.73         4
```

```
import os

dataset_root = r"E:\Publications\Paper 6\Datasets\Preprocessed"

print("Files in Preprocessed folder:")
for f in os.listdir(dataset_root):
    print(f)
```

```
Files in Preprocessed folder:
dev
dev_features.npz
Faces_Extracted
metadata_clean.csv
MFCC_Features
MFCC_Features-20251019T120712Z-1-001.zip
test
test_features.npz
train
train_features.npz
```

```
import os
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report, accuracy_score
from tqdm import tqdm
```

```
dataset_root = r"E:\Publications\Paper 6\Datasets\Preprocessed"
mfcc_root = os.path.join(dataset_root, "MFCC_Features")
metadata_file = os.path.join(dataset_root, "metadata_clean.csv")

# --- Load metadata ---
metadata = pd.read_csv(metadata_file)  # should contain 'file' and 'label' columns
label_mapping = dict(zip(metadata['file'], metadata['label']))

# --- Load MFCC features ---
X, y = [], []
mfcc_files = [f for f in os.listdir(mfcc_root) if f.endswith(".npy")]
print(f"Detected {len(mfcc_files)} MFCC files.")

max_frames = 0
for f in mfcc_files:
    mfcc = np.load(os.path.join(mfcc_root, f))
    max_frames = max(max_frames, mfcc.shape[1])
print("Maximum frames in any MFCC:", max_frames)

for f in tqdm(mfcc_files, desc="Loading MFCC features"):
    mfcc = np.load(os.path.join(mfcc_root, f))
    # Pad/truncate to max_frames
    if mfcc.shape[1] < max_frames:
        pad_width = max_frames - mfcc.shape[1]
        mfcc = np.pad(mfcc, ((0,0),(0,pad_width)), mode='constant')
    elif mfcc.shape[1] > max_frames:
        mfcc = mfcc[:, :max_frames]
    X.append(mfcc.flatten())  # flatten to 1D
```

```
        X.append(mfcc.flatten())  # flatten to 1D
        # assign label from metadata
        file_name = f.replace("_mfcc.npy","")
        y.append(label_mapping[file_name])

X = np.array(X)
y = np.array(y)

print("Feature matrix shape:", X.shape)
print("Labels shape:", y.shape)

# --- Split train/test ---
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42, stratify=y
)

# --- Train a simple baseline classifier ---
clf = LogisticRegression(max_iter=500)
clf.fit(X_train, y_train)

y_pred = clf.predict(X_test)

acc = accuracy_score(y_test, y_pred)
print("\nBaseline accuracy:", acc)
print("\nClassification report:\n", classification_report(y_test, y_pred))
```

```
import pandas as pd
metadata_file = r"E:\Publications\Paper 6\Datasets\Preprocessed\metadata_clean.csv"

metadata = pd.read_csv(metadata_file)
print("Columns in metadata CSV:", metadata.columns)
print("First 5 rows:\n", metadata.head())
```

```
Columns in metadata CSV: Index(['file', 'n_fakes', 'fake_periods', 'timestamps', 'duration',
       'transcript', 'original', 'modify_video', 'modify_audio', 'split',
       'video_frames', 'audio_channels', 'audio_frames'],
      dtype='object')
First 5 rows:
              file  n_fakes    fake_periods  \
0  test/000001.mp4        0              []
1  test/000000.mp4        0              []
2  test/000002.mp4        1  [[2.5, 3.224]]
3  test/000003.mp4        1   [[2.5, 2.78]]
4  test/000004.mp4        1  [[2.5, 3.204]]

                                      timestamps  duration  \
0  [['not', 0.0, 0.2], ['the', 0.2, 0.4], ['point...     4.224
1  [['I', 0.0, 0.1], ['side', 0.1, 0.4], ['with',...     4.672
2  [['not', 0.0, 0.2], ['the', 0.2, 0.4], ['point...     4.800
3  [['not', 0.0, 0.2], ['the', 0.2, 0.4], ['point...     4.352
4  [['not', 0.0, 0.2], ['the', 0.2, 0.4], ['point...     4.736

                                      transcript         original  \
0  not the point the point is that she died satis...              NaN
1  I side with what she's done with this gift of ...              NaN
```

```
                                      transcript         original  \
0  not the point the point is that she died satis...              NaN
1  I side with what she's done with this gift of ...              NaN
2  not the point the point is that she be_born sa...  test/000001.mp4
3  not the point the point is that she be_born sa...  test/000001.mp4
4  not the point the point is that she be_born sa...  test/000001.mp4

   modify_video  modify_audio split  video_frames  audio_channels  \
0         False         False  test           103               1
1         False         False  test           115               1
2          True          True  test           116               1
3          True         False  test           105               1
4         False          True  test           114               1

   audio_frames
0         65536
1         72704
2         74752
3         67584
4         73728
```

```
!echo "Connected to local runtime!"
import torch
print(torch.__version__)
```

```
"Connected to local runtime!"
2.5.1
```

```python
# --------------------------------
# 0️⃣ Setup
# --------------------------------
import os
import numpy as np
import pandas as pd
from tqdm import tqdm

import torch
import torch.nn as nn
from torch.utils.data import Dataset, DataLoader

from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score, classification_report

# Device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# --------------------------------
# 1️⃣ Paths
# --------------------------------
dataset_root = r"E:\Publications\Paper 6\Datasets\Preprocessed"
metadata_file = os.path.join(dataset_root, "metadata_clean.csv")

mfcc_paths = {
    "train": os.path.join(dataset_root, "train", "audio"),
    "dev":   os.path.join(dataset_root, "dev", "audio"),
```

```python
mfcc_paths = {
    "train": os.path.join(dataset_root, "train", "audio"),
    "dev":   os.path.join(dataset_root, "dev", "audio"),
    "test":  os.path.join(dataset_root, "test", "audio")
}

# --------------------------------
# 2️⃣ Load metadata & assign labels
# --------------------------------
metadata = pd.read_csv(metadata_file)
metadata['label'] = metadata['n_fakes'].apply(lambda x: 1 if x>0 else 0)
label_mapping = {os.path.join(split, fn.replace(".npy", ".mp4")): lbl
                 for split in mfcc_paths
                 for fn, lbl in zip(os.listdir(mfcc_paths[split]),
                                    metadata['label']) if fn.endswith(".npy")}

# --------------------------------
# 3️⃣ Load MFCC features with padding
# --------------------------------
X, y = [], []
max_frames = 0

# Find max frames
for mfcc_dir in mfcc_paths.values():
    for fn in os.listdir(mfcc_dir):
        if fn.endswith(".npy"):
            mfcc = np.load(os.path.join(mfcc_dir, fn))
            max_frames = max(max_frames, mfcc.shape[1])
```

```python
print("Maximum frames in any MFCC:", max_frames)

# Load and pad
for split, mfcc_dir in mfcc_paths.items():
    files = [f for f in os.listdir(mfcc_dir) if f.endswith(".npy")]
    for fn in tqdm(files, desc=f"Loading {split} MFCCs"):
        mfcc = np.load(os.path.join(mfcc_dir, fn))
        if mfcc.shape[1] < max_frames:
            pad_width = max_frames - mfcc.shape[1]
            mfcc = np.pad(mfcc, ((0,0),(0,pad_width)), mode='constant')
        X.append(mfcc.flatten())

        video_key = os.path.join(split, fn.replace(".npy", ".mp4"))
        y.append(label_mapping.get(video_key, 0))  # fallback 0 if missing

X = np.array(X, dtype=np.float32)
y = np.array(y, dtype=np.int64)

print("Feature matrix shape:", X.shape)
print("Labels shape:", y.shape)

# --------------------------------
# 4️⃣ Optional normalization
# --------------------------------
scaler = StandardScaler()
X = scaler.fit_transform(X)
```

```
train_mask = np.array([("train" in k or "dev" in k) for k in label_mapping.keys()])
test_mask = np.array([("test" in k) for k in label_mapping.keys()])

X_train, y_train = X[train_mask], y[train_mask]
X_test, y_test = X[test_mask], y[test_mask]

print("Train set:", X_train.shape, y_train.shape)
print("Test set:", X_test.shape, y_test.shape)

# --------------------------------
# 6️⃣ PyTorch Dataset & DataLoader
# --------------------------------
class MFCCDataset(Dataset):
    def __init__(self, X, y):
        self.X = torch.tensor(X, dtype=torch.float32)
        self.y = torch.tensor(y, dtype=torch.long)
    def __len__(self):
        return len(self.y)
    def __getitem__(self, idx):
        return self.X[idx], self.y[idx]

batch_size = 64
train_loader = DataLoader(MFCCDataset(X_train, y_train), batch_size=batch_size, shuffle=True)
test_loader  = DataLoader(MFCCDataset(X_test, y_test), batch_size=batch_size, shuffle=False)
```

```
input_dim = X_train.shape[1]
hidden_dim = 512
output_dim = 2

class MLP(nn.Module):
    def __init__(self):
        super().__init__()
        self.net = nn.Sequential(
            nn.Linear(input_dim, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_dim, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_dim, output_dim)
        )
    def forward(self, x):
        return self.net(x)

model = MLP().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=1e-3)

# --------------------------------
# 7️⃣ Training loop with early stopping
# --------------------------------
epochs = 20
patience = 3
```

```
patience = 3
best_acc = 0
trigger = 0

for epoch in range(epochs):
    model.train()
    total_loss = 0
    for xb, yb in train_loader:
        xb, yb = xb.to(device), yb.to(device)
        optimizer.zero_grad()
        out = model(xb)
        loss = criterion(out, yb)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    avg_loss = total_loss / len(train_loader)

    # Evaluate on test set
    model.eval()
    all_preds, all_labels = [], []
    with torch.no_grad():
        for xb, yb in test_loader:
            xb, yb = xb.to(device), yb.to(device)
            out = model(xb)
            preds = torch.argmax(out, dim=1)
            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(yb.cpu().numpy())
    acc = accuracy_score(all_labels, all_preds)
    print(f"Epoch {epoch+1}/{epochs}, Loss: {avg_loss:.4f}, Test Acc: {acc:.4f}")
```

```python
            all_labels.extend(yb.cpu().numpy())
    acc = accuracy_score(all_labels, all_preds)
    print(f"Epoch {epoch+1}/{epochs}, Loss: {avg_loss:.4f}, Test Acc: {acc:.4f}")

    # Early stopping
    if acc > best_acc:
        best_acc = acc
        trigger = 0
    else:
        trigger += 1
        if trigger >= patience:
            print(f"Early stopping at epoch {epoch+1}")
            break

# ------------------------------
# 9 Final evaluation
# ------------------------------
print("\nFinal Test Accuracy:", best_acc)
print("\nClassification Report:\n", classification_report(all_labels, all_preds))

# ------------------------------
# 10 Save trained model
# ------------------------------
torch.save(model.state_dict(), os.path.join(dataset_root, "mlp_baseline.pth"))
print("Model saved successfully!")
```

```
Loading train MFCCs: 100%|                                    | 56775/5
Loading dev MFCCs: 100%|                                      | 31501/3
Loading test MFCCs: 100%|                                     | 26100/2
Feature matrix shape: (114376, 11154)
Labels shape: (114376,)
Train set: (88276, 11154) (88276,)
Test set: (26100, 11154) (26100,)
Epoch 1/20, Loss: 0.6042, Test Acc: 0.7333
Epoch 2/20, Loss: 0.5831, Test Acc: 0.7333
Epoch 3/20, Loss: 0.5811, Test Acc: 0.7333
Epoch 4/20, Loss: 0.5811, Test Acc: 0.7333
Early stopping at epoch 4

Final Test Accuracy: 0.7332950191570882

Classification Report:
               precision    recall  f1-score   support

           0       0.00      0.00      0.00      6961
           1       0.73      1.00      0.85     19139

    accuracy                           0.73     26100
   macro avg       0.37      0.50      0.42     26100
weighted avg       0.54      0.73      0.62     26100

C:\Users\HP\anaconda3\envs\colab-local\lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels wit
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\HP\anaconda3\envs\colab-local\lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels wit
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
C:\Users\HP\anaconda3\envs\colab-local\lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels wit
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Model saved successfully!
```

```python
import cv2
import torch
from torch.utils.data import Dataset
import os
import numpy as np

class VideoFramesDataset(Dataset):
    def __init__(self, video_files, labels, num_frames=75):
        self.video_files = video_files
        self.labels = labels
        self.num_frames = num_frames

    def __len__(self):
        return len(self.video_files)

    def __getitem__(self, idx):
        video_path = self.video_files[idx]
        label = self.labels[idx]

        # Load video frames (grayscale, resized)
        cap = cv2.VideoCapture(video_path)
        frames = []
        for i in range(self.num_frames):
            ret, frame = cap.read()
            if not ret:
                break
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            frame = cv2.resize(frame, (64,64))
```

```
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
            frame = cv2.resize(frame, (64,64))
            frames.append(frame)
        cap.release()

        # Pad if fewer frames
        while len(frames) < self.num_frames:
            frames.append(np.zeros((64,64)))

        frames = np.array(frames, dtype=np.float32) / 255.0
        frames = np.expand_dims(frames, axis=1)  # add channel dim

        return torch.tensor(frames), torch.tensor(label, dtype=torch.long)
```

```
from torch.utils.data import DataLoader
from sklearn.model_selection import train_test_split

# List all video paths and labels (assuming you already have them in 'label_mapping')
video_files = list(label_mapping.keys())
labels = [label_mapping[k] for k in video_files]

# Split into train and test (80/20)
train_videos, test_videos, train_labels, test_labels = train_test_split(
    video_files, labels, test_size=0.2, random_state=42, stratify=labels
)
```

```
train_dataset = VideoFramesDataset(train_videos, train_labels, num_frames=75)
test_dataset  = VideoFramesDataset(test_videos, test_labels, num_frames=75)

# DataLoaders
batch_size = 16  # reduce if GPU memory is limited
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader  = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

print(f"Train batches: {len(train_loader)}, Test batches: {len(test_loader)}")
```

```
Train batches: 5719, Test batches: 1430
```

```
# --------------------------------
# 1️⃣ SyncNet-style baseline model
# --------------------------------
import torch
import torch.nn as nn

class LipSyncNet(nn.Module):
    def __init__(self, n_mfcc=20, n_frames=16, n_channels=3, hidden_dim=512):
        super().__init__()
        # Audio branch
        self.audio_net = nn.Sequential(
            nn.Conv1d(n_mfcc, 128, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.Conv1d(128, 256, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.AdaptiveAvgPool1d(1)
        )
        # Video branch (simple CNN on frames)
        self.video_net = nn.Sequential(
            nn.Conv2d(n_channels, 32, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d((1,1))
        )
        # Fully connected classifier
        self.classifier = nn.Sequential(
            nn.Linear(256 + 64, hidden_dim),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(hidden_dim, 2)  # real/fake
        )

    def forward(self, audio, video):
        # audio: [B, n_mfcc, n_frames], video: [B, C, H, W]
        a = self.audio_net(audio).squeeze(-1)    # [B, 256]
        v = self.video_net(video).view(video.size(0), -1)  # [B, 64]
        x = torch.cat([a,v], dim=1)
        out = self.classifier(x)
        return out

# Initialize and move to device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = LipSyncNet(n_mfcc=20, n_frames=16).to(device)
```

```
model = LipSyncNet(n_mfcc=20, n_frames=16).to(device)
print(model)
```

```
LipSyncNet(
  (audio_net): Sequential(
    (0): Conv1d(20, 128, kernel_size=(5,), stride=(1,), padding=(2,))
    (1): ReLU()
    (2): Conv1d(128, 256, kernel_size=(5,), stride=(1,), padding=(2,))
    (3): ReLU()
    (4): AdaptiveAvgPool1d(output_size=1)
  )
  (video_net): Sequential(
    (0): Conv2d(3, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (1): ReLU()
    (2): Conv2d(32, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1))
    (3): ReLU()
    (4): AdaptiveAvgPool2d(output_size=(1, 1))
  )
  (classifier): Sequential(
    (0): Linear(in_features=320, out_features=512, bias=True)
    (1): ReLU()
    (2): Dropout(p=0.3, inplace=False)
    (3): Linear(in_features=512, out_features=2, bias=True)
  )
)
```

```
from torch.utils.data import Dataset, DataLoader
import cv2  # for video frame reading
import numpy as np
import os

class LipSyncDataset(Dataset):
    def __init__(self, mfcc_dir, video_dir, metadata, n_frames=16, n_mfcc=20, transform=None):
        self.mfcc_dir = mfcc_dir
        self.video_dir = video_dir
        self.metadata = metadata  # DataFrame with 'file' and 'label'
        self.n_frames = n_frames
        self.n_mfcc = n_mfcc
        self.transform = transform

    def __len__(self):
        return len(self.metadata)

    def __getitem__(self, idx):
        row = self.metadata.iloc[idx]
        file_name = row['file']
        label = row['label']

        # Load MFCC
        mfcc_path = os.path.join(self.mfcc_dir, file_name.replace('.mp4','.npy'))
        audio = np.load(mfcc_path)  # shape [n_mfcc, frames]
        if audio.shape[1] < self.n_frames:
            pad_width = self.n_frames - audio.shape[1]
            audio = np.pad(audio, ((0,0),(0,pad_width)), mode='constant')
        audio = audio[:, :self.n_frames]

        # Load video frames (uniform sampling)
        video_path = os.path.join(self.video_dir, file_name)
        cap = cv2.VideoCapture(video_path)
        frames = []
        total_frames = int(cap.get(cv2.CAP_PROP_FRAME_COUNT))
        indices = np.linspace(0, total_frames-1, self.n_frames, dtype=int)
        for i in range(total_frames):
            ret, frame = cap.read()
            if not ret:
                break
            if i in indices:
                frame = cv2.resize(frame, (64,64))
                frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
                frames.append(frame)
        cap.release()
        frames = np.array(frames).transpose(3,0,1,2)  # [C, T, H, W] -> rearranged later

        # convert to tensors
        audio = torch.tensor(audio, dtype=torch.float32)
        video = torch.tensor(frames, dtype=torch.float32) / 255.0
        return audio, video, torch.tensor(label, dtype=torch.long)

# Example usage
train_dataset = LipSyncDataset(
    mfcc_dir=r"E:\Publications\Paper 6\Datasets\Preprocessed\train\audio",
    video_dir=r"E:\Publications\Paper 6\Datasets\Preprocessed\train\video",
```

```python
# Example usage
train_dataset = LipSyncDataset(
    mfcc_dir=r"E:\Publications\Paper 6\Datasets\Preprocessed\train\audio",
    video_dir=r"E:\Publications\Paper 6\Datasets\Preprocessed\train\video",
    metadata=metadata[metadata['file'].str.contains('train')]
)
train_loader = DataLoader(train_dataset, batch_size=32, shuffle=True)
print(f"Train batches: {len(train_loader)}")
```

```
Train batches: 2460
```

```python
import os

train_audio_dir = r"E:\Publications\Paper 6\Datasets\Preprocessed\train\audio"
print("First 10 files in train audio folder:")
print(os.listdir(train_audio_dir)[:10])
```

```
First 10 files in train audio folder:
['000469.npy', '000470.npy', '000471.npy', '000472.npy', '000473.npy', '000474.npy', '000475.npy', '000476.npy', '000477.npy', '000478.npy']
```

```python
class LipSyncDataset(torch.utils.data.Dataset):
    def __init__(self, mfcc_dir, video_dir, label_mapping, n_frames=80):
        self.mfcc_dir = mfcc_dir
        self.video_dir = video_dir
        self.label_mapping = label_mapping
        self.files = sorted([f for f in os.listdir(mfcc_dir) if f.endswith('.npy')])
        self.n_frames = n_frames

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        file_name = self.files[idx]

        # Load MFCC
        mfcc_path = os.path.join(self.mfcc_dir, file_name)
        audio = np.load(mfcc_path)  # shape [n_mfcc, frames]
        if audio.shape[1] < self.n_frames:
            pad_width = self.n_frames - audio.shape[1]
            audio = np.pad(audio, ((0,0),(0,pad_width)), mode='constant')
        else:
            audio = audio[:, :self.n_frames]
        audio = torch.tensor(audio, dtype=torch.float32)

        # Load video (assuming frames are preprocessed similarly)
        video_path = os.path.join(self.video_dir, file_name.replace('.npy', '.npy'))  # adjust if using video frames
        video = np.load(video_path)
        video = torch.tensor(video, dtype=torch.float32)

        # Get label
        label = self.label_mapping[file_name]  # use file_name as key
        label = torch.tensor(label, dtype=torch.long)

        return audio, video, label
```

```python
import cv2
import os
import numpy as np
from tqdm import tqdm

# Base folder where train/dev/test are located
dataset_root = r"E:\Publications\Paper 6\Datasets\Preprocessed"

splits = ["train", "dev", "test"]
frame_size = (128, 128)  # Resize frames to save memory

for split in splits:
    video_folder = os.path.join(dataset_root, split)
    save_folder = os.path.join(video_folder, "video")
    os.makedirs(save_folder, exist_ok=True)

    videos = [f for f in os.listdir(video_folder) if f.endswith(".mp4")]
    print(f"\nProcessing {split} videos: {len(videos)} files")

    for video_file in tqdm(videos, desc=f"{split} videos"):
        video_path = os.path.join(video_folder, video_file)
        cap = cv2.VideoCapture(video_path)
        frames = []

        while True:
            ret, frame = cap.read()
            if not ret:
                break
```

```python
                break
            frame = cv2.resize(frame, frame_size)
            frames.append(frame)

        cap.release()
        # Save frames as .npy
        np.save(os.path.join(save_folder, video_file.replace(".mp4", ".npy")), np.array(frames))
```

```python
import os
import cv2
import numpy as np
from tqdm import tqdm
import torch
from torchvision import transforms

# ----------------------------
# Paths
# ----------------------------
dataset_root = r"E:\Publications\Paper 6\Datasets\New Folder\LAV-DF\LAV-DF"
preprocessed_root = r"D:\Paper6_Preprocessed"  # Save to another drive with enough space

splits = ["train", "dev", "test"]

for split in splits:
    os.makedirs(os.path.join(preprocessed_root, split), exist_ok=True)

# ----------------------------
# Parameters
# ----------------------------
frame_size = (128, 128)
max_frames_per_video = 50
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

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

```python
transform = transforms.Compose([
    transforms.ToTensor(),  # Converts to [C,H,W] float tensor
    transforms.Resize(frame_size),  # Resize on GPU
    transforms.Grayscale(num_output_channels=1)  # Single channel
])

# ----------------------------
# Function to process videos
# ----------------------------
def process_videos(split):
    video_dir = os.path.join(dataset_root, split)
    save_dir = os.path.join(preprocessed_root, split)

    video_files = [f for f in os.listdir(video_dir) if f.endswith(".mp4")]
    print(f"Processing {split} videos: {len(video_files)} files")

    for vf in tqdm(video_files, desc=f"{split} videos"):
        video_path = os.path.join(video_dir, vf)
        cap = cv2.VideoCapture(video_path)
        frames = []
        frame_count = 0

        while True:
            ret, frame = cap.read()
            if not ret or frame_count >= max_frames_per_video:
                break
```

```python
            # Convert BGR to RGB for torchvision
            frame = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
            # Apply GPU transform
            frame_tensor = transform(frame).to(device)
            frames.append(frame_tensor.cpu().numpy())  # Move back to CPU for saving
            frame_count += 1

        cap.release()

        if frames:
            frames = np.stack(frames)
            save_path = os.path.join(save_dir, vf.replace(".mp4", ".npz"))
            np.savez_compressed(save_path, frames)

# ----------------------------
# Run for all splits
# ----------------------------
for split in splits:
    process_videos(split)

print("All preprocessing done! Compressed files saved.")
```

```
Using device: cpu
Processing train videos: 56776 files
train videos: 100%|████████████████████████████████████████████████████████████████████████████████████████████████████████████| 56776/5
Processing dev videos: 31501 files
dev videos: 100%|██████████████████████████████████████████████████████████████████████████████████████████████████████████████| 31501/3
```

```python
# ✅ Fresh start: process TEST videos only, saving compressed arrays to E drive
import os, cv2, numpy as np
from tqdm import tqdm
import shutil

# === CONFIGURATION ===
input_dir = r"E:\Publications\Paper 6\Datasets\New Folder\LAV-DF\LAV-DF\test"
output_dir = r"E:\Publications\Paper 6\Datasets\Preprocessed\test\video"

os.makedirs(output_dir, exist_ok=True)

TARGET_FRAMES = 20       # keep this many frames per video
FRAME_SKIP = 2           # skip every 2nd frame for speed
RESIZE = (96, 96)        # resize frames to this size
MIN_FREE_GB = 5          # minimum free space (in GB) to continue saving

# === UTILS ===
def get_free_gb(path):
    total, used, free = shutil.disk_usage(os.path.dirname(path))
    return free / (1024**3)

def extract_frames(video_path, target_frames=20, frame_skip=2, resize=(96, 96)):
    cap = cv2.VideoCapture(video_path)
    frames = []
    idx = 0
    while True:
        ret, frame = cap.read()
        if not ret:
            if idx % frame_skip == 0:
                frame = cv2.resize(frame, resize, interpolation=cv2.INTER_AREA)
                frames.append(frame)
            idx += 1
            if len(frames) >= target_frames:
                break
    cap.release()
    return np.array(frames, dtype=np.uint8)

# === MAIN EXTRACTION LOOP ===
video_files = sorted([f for f in os.listdir(input_dir) if f.lower().endswith(".mp4")])
print(f"🎬 Found {len(video_files)} videos in {input_dir}")

for vf in tqdm(video_files, desc="Extracting test videos", unit="video"):
    in_path = os.path.join(input_dir, vf)
    out_path = os.path.join(output_dir, vf.replace(".mp4", ".npz"))

    try:
        # check free space before saving
        if get_free_gb(output_dir) < MIN_FREE_GB:
            print("⚠️ Low disk space (<5GB). Stopping to prevent corruption.")
            break

        frames = extract_frames(in_path, TARGET_FRAMES, FRAME_SKIP, RESIZE)
        if frames.size == 0:
            tqdm.write(f"⚠️ No frames extracted from {vf}, skipping.")
            continue

        np.savez_compressed(out_path, frames=frames)

    except Exception as e:
        tqdm.write(f"⚠️ Error processing {vf}: {e}")

print("\n✅ Test video preprocessing complete!")
print(f"Saved processed files to: {output_dir}")
```

```
🎬 Found 26100 videos in E:\Publications\Paper 6\Datasets\New Folder\LAV-DF\LAV-DF\test
Extracting test videos: 100%|██████████| 26100/2610
✅ Test video preprocessing complete!
Saved processed files to: E:\Publications\Paper 6\Datasets\Preprocessed\test\video
```

```python
import numpy as np
import os

test_video_dir = r"E:\Publications\Paper 6\Datasets\Preprocessed\test\video"
sample_file = os.listdir(test_video_dir)[0]  # first file
npz = np.load(os.path.join(test_video_dir, sample_file))
```

```
test_video_dir = r"E:\Publications\Paper 6\Datasets\Preprocessed\test\video"
sample_file = os.listdir(test_video_dir)[0]  # first file
npz = np.load(os.path.join(test_video_dir, sample_file))
print("Keys inside the .npz:", npz.files)
```

```
Keys inside the .npz: ['frames']
```

```
def pad_or_trim_video(video_array, max_frames=50):
    """Pad or trim video frames to have exactly max_frames."""
    n = video_array.shape[0]
    if n < max_frames:
        pad_width = max_frames - n
        pad_shape = ((0, pad_width),) + tuple((0,0) for _ in range(video_array.ndim-1))
        video_array = np.pad(video_array, pad_shape, mode='constant')
    elif n > max_frames:
        video_array = video_array[:max_frames]
    return video_array

class LipSyncDataset(torch.utils.data.Dataset):
    def __init__(self, split, mfcc_root, video_root, max_frames=50):
        self.split = split
        self.mfcc_root = mfcc_root
        self.video_root = video_root
        self.max_frames = max_frames
        self.files = [f for f in os.listdir(video_root) if f.endswith(".npz")]
```

```
    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        fname = self.files[idx]

        # Load MFCC
        mfcc_path = os.path.join(self.mfcc_root, fname.replace(".npz", ".npy"))
        audio = np.load(mfcc_path)
        audio = torch.tensor(audio, dtype=torch.float32)

        # Load Video
        video_npz = np.load(os.path.join(self.video_root, fname))
        key = list(video_npz.keys())[0]  # dynamically get the first key
        video = pad_or_trim_video(video_npz[key], self.max_frames)
        video = torch.tensor(video, dtype=torch.float32)

        return audio, video, fname
```

```
import os
import numpy as np
import torch
from torch.utils.data import Dataset, DataLoader
```

```
mfcc_paths = {
    "train": r"E:\Publications\Paper 6\Datasets\Preprocessed\train\audio",
    "dev":   r"E:\Publications\Paper 6\Datasets\Preprocessed\dev\audio",
    "test":  r"E:\Publications\Paper 6\Datasets\Preprocessed\test\audio"
}

video_paths = {
    "train": r"D:\Paper6_Preprocessed\train",
    "dev":   r"D:\Paper6_Preprocessed\dev",
    "test":  r"E:\Publications\Paper 6\Datasets\Preprocessed\test\video"
}

# ----------------------------
# Parameters
# ----------------------------
MAX_MFCC_FRAMES = 400      # Adjust based on your preprocessing
MAX_VIDEO_FRAMES = 50
DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", DEVICE)

# ----------------------------
# Helper functions
# ----------------------------
def pad_or_trim_mfcc(mfcc):
    n_mfcc, n_frames = mfcc.shape
    if n_frames < MAX_MFCC_FRAMES:
        pad_width = MAX_MFCC_FRAMES - n_frames
        mfcc = np.pad(mfcc, ((0,0),(0,pad_width)), mode='constant')
```

```
        else:
            mfcc = mfcc[:, :MAX_MFCC_FRAMES]
        return mfcc

    def pad_or_trim_video(frames):
        n_frames = frames.shape[0]
        if n_frames < MAX_VIDEO_FRAMES:
            pad_width = MAX_VIDEO_FRAMES - n_frames
            pad_shape = (pad_width, *frames.shape[1:])
            frames = np.concatenate([frames, np.zeros(pad_shape, dtype=frames.dtype)], axis=0)
        else:
            frames = frames[:MAX_VIDEO_FRAMES]
        return frames

    # ------------------------------
    # Dataset class
    # ------------------------------
    class LipSyncDataset(Dataset):
        def __init__(self, split):
            self.split = split
            self.mfcc_dir = mfcc_paths[split]
            self.video_dir = video_paths[split]

            # Collect files present in both audio & video
            self.files = []
            for f in os.listdir(self.mfcc_dir):
                if f.endswith(".npy") and os.path.exists(os.path.join(self.video_dir, f.replace(".npy",".npz"))):
                    self.files.append(f)
```

```
            print(f"[{split}] Matched: {len(self.files)}")

        def __len__(self):
            return len(self.files)

        def __getitem__(self, idx):
            fname = self.files[idx]

            # Load MFCC
            audio = np.load(os.path.join(self.mfcc_dir, fname))
            audio = pad_or_trim_mfcc(audio)
            audio = torch.tensor(audio, dtype=torch.float32)

            # Load video frames
            video_npz = np.load(os.path.join(self.video_dir, fname.replace(".npy",".npz")))
            if "frames" in video_npz:
                key = "frames"
            elif "arr_0" in video_npz:
                key = "arr_0"
            else:
                raise KeyError(f"No valid key in {fname}")

            video_frames = pad_or_trim_video(video_npz[key])
            video = torch.tensor(video_frames, dtype=torch.float32)

            return audio, video, fname
```

```
    # ------------------------------
    batch_size = 2
    data_loaders = {}
    for split in ["train","dev","test"]:
        ds = LipSyncDataset(split)
        loader = DataLoader(ds, batch_size=batch_size, shuffle=True)
        data_loaders[split] = loader

    # ------------------------------
    # Test a batch from each split
    # ------------------------------
    for split, loader in data_loaders.items():
        audio, video, names = next(iter(loader))
        print(f"\n✅ {split.upper()} batch")
        print("Audio:", audio.shape, "| Video:", video.shape, "| Samples:", len(names))
```

```
Using device: cpu
[train] Matched: 56775
[dev] Matched: 31501
[test] Matched: 26100

✅ TRAIN batch
Audio: torch.Size([2, 13, 400]) | Video: torch.Size([2, 50, 1, 128, 128]) | Samples: 2

✅ DEV batch
Audio: torch.Size([2, 13, 400]) | Video: torch.Size([2, 50, 1, 128, 128]) | Samples: 2

✅ TEST batch
Audio: torch.Size([2, 13, 400]) | Video: torch.Size([2, 50, 96, 96, 3]) | Samples: 2
```

```python
import torch
import torch.nn as nn
import torch.optim as optim

# ------------------------------
# Device
# ------------------------------
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# ------------------------------
# Updated LipSyncNet for n_mfcc=13
# ------------------------------
class LipSyncNet(nn.Module):
    def __init__(self, n_mfcc=13):
        super().__init__()
        self.audio_net = nn.Sequential(
            nn.Conv1d(n_mfcc, 128, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.Conv1d(128, 256, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.AdaptiveAvgPool1d(1)
        )
        self.video_net = nn.Sequential(
            nn.Conv2d(3, 32, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool2d((1, 1))
        )
        self.classifier = nn.Sequential(
            nn.Linear(256+64, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 2)
        )

    def forward(self, audio, video):
        a = self.audio_net(audio).squeeze(-1)  # [B, 256]
        v = self.video_net(video).view(video.size(0), -1)  # [B, 64]
        x = torch.cat([a, v], dim=1)
        return self.classifier(x)

# ------------------------------
# Initialize model, criterion, optimizer
# ------------------------------
model = LipSyncNet(n_mfcc=13).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-4)

print("Model, criterion, and optimizer ready!")
```

```
Using device: cpu
Model, criterion, and optimizer ready!
```

```python
class LipSyncDataset(Dataset):
    def __init__(self, audio_dir, video_dir, max_frames=50):
        self.audio_dir = audio_dir
        self.video_dir = video_dir
        self.files = [f for f in os.listdir(audio_dir) if f.endswith('.npy')]
        self.max_frames = max_frames

    def __len__(self):
        return len(self.files)

    def __getitem__(self, idx):
        import numpy as np
        import torch
        import os

        fname = self.files[idx]

        # Load audio
        audio = np.load(os.path.join(self.audio_dir, fname))
        audio = torch.tensor(audio, dtype=torch.float32)

        # Load video
        video_npz = np.load(os.path.join(self.video_dir, fname.replace('.npy','.npz')))

        # Automatically get the first key in npz (works for both train/dev/test)
        video_key = list(video_npz.keys())[0]
        video = video_npz[video_key]
```

```
        if video.shape[0] < self.max_frames:
            pad = self.max_frames - video.shape[0]
            video = np.pad(video, ((0,pad),(0,0),(0,0),(0,0)), mode='constant')
        else:
            video = video[:self.max_frames]
        video = torch.tensor(video, dtype=torch.float32)

        # Dummy label (all real=1)
        label = torch.tensor(1, dtype=torch.long)
        return audio, video, label
```

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
import numpy as np
import os
from tqdm import tqdm

# ----------------------------
# Paths
# ----------------------------
train_audio_dir = r"E:\Publications\Paper 6\Datasets\Preprocessed\train\audio"
dev_audio_dir = r"E:\Publications\Paper 6\Datasets\Preprocessed\dev\audio"
test_audio_dir = r"E:\Publications\Paper 6\Datasets\Preprocessed\test\audio"

train_video_dir = r"D:\Paper6_Preprocessed\train"
```

```
train_video_dir = r"D:\Paper6_Preprocessed\train"
dev_video_dir = r"D:\Paper6_Preprocessed\dev"
test_video_dir = r"E:\Publications\Paper 6\Datasets\Preprocessed\test\video"

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# ----------------------------
# Dataset
# ----------------------------
class LipSyncDataset(Dataset):
    def __init__(self, split, max_audio_frames=400, max_video_frames=50):
        if split=="train":
            self.audio_dir = train_audio_dir
            self.video_dir = train_video_dir
        elif split=="dev":
            self.audio_dir = dev_audio_dir
            self.video_dir = dev_video_dir
        else:
            self.audio_dir = test_audio_dir
            self.video_dir = test_video_dir

        self.files = sorted([f for f in os.listdir(self.audio_dir) if f.endswith('.npy')])
        self.max_audio_frames = max_audio_frames
        self.max_video_frames = max_video_frames

    def __len__(self):
        return len(self.files)
```

```
    def pad_or_trim_audio(self, audio):
        n_mfcc, n_frames = audio.shape
        if n_frames < self.max_audio_frames:
            audio = np.pad(audio, ((0,0),(0,self.max_audio_frames - n_frames)), mode='constant')
        else:
            audio = audio[:, :self.max_audio_frames]
        return audio

    def pad_or_trim_video(self, video):
        if video.shape[0] < self.max_video_frames:
            pad_len = self.max_video_frames - video.shape[0]
            video = np.pad(video, ((0,pad_len),(0,0),(0,0),(0,0)), mode='constant')
        else:
            video = video[:self.max_video_frames]
        return video

    def __getitem__(self, idx):
        fname = self.files[idx]

        # Load audio
        audio = np.load(os.path.join(self.audio_dir, fname))
        audio = self.pad_or_trim_audio(audio)
        audio = torch.tensor(audio, dtype=torch.float32)

        # Load video
        video_npz = np.load(os.path.join(self.video_dir, fname.replace('.npy','.npz')))
        if 'frames' in video_npz:
```

```
            video = video_npz['frames']
        else:
            video = video_npz[list(video_npz.keys())[0]]

        # Ensure single channel and correct shape [T,C,H,W]
        if video.ndim == 4:  # [T,H,W,C] -> [T,1,H,W]
            if video.shape[3]==1:
                video = video.transpose(0,3,1,2)
            else:
                video = video.mean(axis=3, keepdims=True).transpose(0,3,1,2)
        video = self.pad_or_trim_video(video)
        video = torch.tensor(video, dtype=torch.float32)

        label = torch.ones(1, dtype=torch.long)
        return audio, video, label

# ----------------------------
# DataLoaders
# ----------------------------
train_loader = DataLoader(LipSyncDataset('train'), batch_size=2, shuffle=True)
dev_loader = DataLoader(LipSyncDataset('dev'), batch_size=2, shuffle=False)
test_loader = DataLoader(LipSyncDataset('test'), batch_size=2, shuffle=False)

# ----------------------------
# Model
# ----------------------------
class LipSyncNet(nn.Module):
    def __init__(self):
        super().__init__()
        # Audio network
        self.audio_net = nn.Sequential(
            nn.Conv1d(13, 128, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.Conv1d(128, 256, kernel_size=5, stride=1, padding=2),
            nn.ReLU(),
            nn.AdaptiveAvgPool1d(1)
        )
        # Video network (1-channel)
        self.video_net = nn.Sequential(
            nn.Conv3d(1, 32, kernel_size=(3,3,3), stride=1, padding=1),
            nn.ReLU(),
            nn.Conv3d(32, 64, kernel_size=(3,3,3), stride=1, padding=1),
            nn.ReLU(),
            nn.AdaptiveAvgPool3d((1,1,1))
        )
        # Classifier
        self.classifier = nn.Sequential(
            nn.Linear(256+64, 512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 2)
        )

    def forward(self, audio, video):
        a = self.audio_net(audio).squeeze(-1)  # [B,256]
        # Correct permute: [B,T,C,H,W] -> [B,C,T,H,W]
        video = video.permute(0,2,1,3,4)
        v = self.video_net(video).view(video.size(0), -1)  # [B,64]
        x = torch.cat([a,v], dim=1)
        return self.classifier(x)

# ----------------------------
# Training setup
# ----------------------------
model = LipSyncNet().to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=1e-3)

# ----------------------------
# Training loop - 1 epoch
# ----------------------------
model.train()
for batch_idx, (audio, video, labels) in enumerate(tqdm(train_loader, desc="Training Batches")):
    audio, video, labels = audio.to(device), video.to(device), labels.squeeze().to(device)
    optimizer.zero_grad()
    outputs = model(audio, video)
    loss = criterion(outputs, labels)
    loss.backward()
    optimizer.step()

# ----------------------------
# Test 1 batch
# ----------------------------
model.eval()
# ----------------------------
model.eval()
with torch.no_grad():
    for split, loader in zip(['train','dev','test'], [train_loader, dev_loader, test_loader]):
        audio, video, labels = next(iter(loader))
        audio, video = audio.to(device), video.to(device)
        outputs = model(audio, video)
        print(f"\n✅ {split.upper()} batch")
        print("Audio:", audio.shape, "| Video:", video.shape, "| Outputs:", outputs.shape)
```

```
Using device: cpu
Training Batches: 100%|████████████████████████████████████████████████████████████████████████████████████████████████████████████| 28
```

```python
import os

# Check current folder
print("Current working directory:", os.getcwd())

# List all files in current folder
print("Files here:", os.listdir())

# If you have a models folder
if os.path.exists("models"):
    print("Models folder contents:", os.listdir("models"))
```

```
egtrans-ms', 'ntuser.ini', 'OneDrive', 'Pictures', 'PrintHood', 'Recent', 'Saved Games', 'Searches', 'SendTo', 'Start Menu', 'Templates', 'Untitled.ipynb', 'untitled.txt', 'Videos
```

```python
# Check if dev_loader exists
try:
    print(dev_loader)
except NameError:
    print("dev_loader not found")

# Check if test_loader exists
try:
    print(test_loader)
except NameError:
    print("test_loader not found")
```

```
<torch.utils.data.dataloader.DataLoader object at 0x000002DCC84D5CF0>
<torch.utils.data.dataloader.DataLoader object at 0x000002DCC84D5B70>
```

```python
import torch
from sklearn.metrics import accuracy_score, classification_report
from tqdm import tqdm

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Using device:", device)

# Ensure model is in eval mode
model.eval()

def evaluate(loader, split_name="DEV"):
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for audio, video, labels in tqdm(loader, desc=f"Evaluating {split_name}"):
            audio, video, labels = audio.to(device), video.to(device), labels.squeeze().to(device)

            outputs = model(audio, video)
            preds = torch.argmax(outputs, dim=1)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    acc = accuracy_score(all_labels, all_preds)
    print(f"\n✅ {split_name} Accuracy: {acc*100:.2f}%")
    print(f"\n{split_name} Classification Report:\n", classification_report(all_labels, all_preds, digits=4))

# Evaluate on DEV set
evaluate(dev_loader, "DEV")

# Evaluate on TEST set
evaluate(test_loader, "TEST")
```

```
Using device: cpu
Evaluating DEV: 100%|████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████████| 15750/
--------------------------------------------------------------------------------
```

```
!pip install --upgrade openai-whisper
```

```
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
   ---------------------------------------- 0/6 [sympy]
```

```
# Install required packages
!pip install torch==2.1.0+cu118 torchvision==0.15.2+cu118 torchaudio==2.1.0 --index-url https://download.pytorch.org/whl/cu118
!pip install transformers librosa tqdm opencv-python whisper aeneas
!pip install git+https://github.com/joonson/syncnet_python.git
```

```
copying aeneas\cew\__init__.py -> build\lib.win-amd64-cpython-310\aeneas\cew
creating build\lib.win-amd64-cpython-310\aeneas\cfw
copying aeneas\cfw\cfw_setup.py -> build\lib.win-amd64-cpython-310\aeneas\cfw
copying aeneas\cfw\__init__.py -> build\lib.win-amd64-cpython-310\aeneas\cfw
creating build\lib.win-amd64-cpython-310\aeneas\cmfcc
copying aeneas\cmfcc\cmfcc_setup.py -> build\lib.win-amd64-cpython-310\aeneas\cmfcc
copying aeneas\cmfcc\__init__.py -> build\lib.win-amd64-cpython-310\aeneas\cmfcc
creating build\lib.win-amd64-cpython-310\aeneas\cwave
copying aeneas\cwave\cwave_setup.py -> build\lib.win-amd64-cpython-310\aeneas\cwave
copying aeneas\cwave\__init__.py -> build\lib.win-amd64-cpython-310\aeneas\cwave
creating build\lib.win-amd64-cpython-310\aeneas\extra
copying aeneas\extra\ctw_espeak.py -> build\lib.win-amd64-cpython-310\aeneas\extra
copying aeneas\extra\ctw_speect.py -> build\lib.win-amd64-cpython-310\aeneas\extra
copying aeneas\extra\__init__.py -> build\lib.win-amd64-cpython-310\aeneas\extra
creating build\lib.win-amd64-cpython-310\aeneas\syncmap
copying aeneas\syncmap\format.py -> build\lib.win-amd64-cpython-310\aeneas\syncmap
copying aeneas\syncmap\fragment.py -> build\lib.win-amd64-cpython-310\aeneas\syncmap
copying aeneas\syncmap\fragmentlist.py -> build\lib.win-amd64-cpython-310\aeneas\syncmap
copying aeneas\syncmap\headtailformat.py -> build\lib.win-amd64-cpython-310\aeneas\syncmap
copying aeneas\syncmap\missingparametererror.py -> build\lib.win-amd64-cpython-310\aeneas\syncmap
copying aeneas\syncmap\smfaudacity.py -> build\lib.win-amd64-cpython-310\aeneas\syncmap
copying aeneas\syncmap\smfbase.py -> build\lib.win-amd64-cpython-310\aeneas\syncmap
copying aeneas\syncmap\smfcsv.py -> build\lib.win-amd64-cpython-310\aeneas\syncmap
copying aeneas\syncmap\smfeaf.py -> build\lib.win-amd64-cpython-310\aeneas\syncmap
```