Handling large video files (e.g., 4-hour-long videos) for transcription or other processing tasks requires similar strategies to those used for audio files, with added considerations for video streams. Here's a step-by-step guide:

---

1. Extract Audio from the Video
First, extract the audio from the video, as most transcription tools (like Whisper) process audio files directly.

Using FFmpeg:
```bash
ffmpeg -i input.mp4 -vn -acodec pcm_s16le -ar 16000 -ac 1 output.wav
```
- `-vn`: Ignore video.
- `-acodec pcm_s16le`: Use uncompressed WAV format.
- `-ar 16000`: Set sample rate to 16 kHz (optimal for Whisper).
- `-ac 1`: Convert to mono audio.

---

2. Split the Video into Smaller Segments
For large files, splitting the video into smaller chunks ensures manageable file sizes and memory usage.

Split Video Using FFmpeg:
```bash
ffmpeg -i input.mp4 -c copy -map 0 -segment_time 1800 -f segment output_%03d.mp4
```
This splits the video into 30-minute segments (`1800` seconds) without re-encoding.

Split Audio Using FFmpeg:
If only the audio is needed, split the extracted audio:
```bash
ffmpeg -i output.wav -f segment -segment_time 1800 -c copy audio_%03d.wav
```

---

3. Transcribe Each Segment
Once the video or audio is split, process each segment using Whisper or another transcription tool.

Example with Whisper:
```python
import whisper

model = whisper.load_model("base")
```

```
for i in range(num_segments):  # Adjust for the number of segments
    result = model.transcribe(f"audio_{i}.wav")
    with open(f"audio_{i}.srt", "w") as srt_file:
        srt_file.write(result["text"])
```

---

4. Handle Memory Issues
If processing large segments is memory-intensive:
- Use smaller Whisper models (e.g., `base` or `tiny`).
- Process files sequentially instead of parallelly.

---

5. Combine Transcriptions
After processing all segments, merge the transcriptions into one file.

Combine SRT Files:
```python
with open("final_transcription.srt", "w") as final_file:
    for i in range(num_segments):
        with open(f"audio_{i}.srt", "r") as segment_file:
            final_file.write(segment_file.read())
            final_file.write("\n")
```

---

6. Optional: Recombine Video with Subtitles
If you want to overlay subtitles on the video, recombine the processed SRT file with the video.

Using FFmpeg:
```bash
ffmpeg -i input.mp4 -vf subtitles=final_transcription.srt output_with_subtitles.mp4
```

---

7. Optimize Video Processing
Before splitting:
- Downsample or compress the video to reduce file size:
  ```bash
  ffmpeg -i input.mp4 -vf scale=1280:720 -crf 28 compressed.mp4
  ```
  - `scale=1280:720`: Downscale to 720p.
  - `crf 28`: Set compression quality (lower is better; 23 is default).

Example Workflow
1. Extract audio from the video.
2. Split the audio or video into smaller chunks.
3. Process each chunk with Whisper for transcription.
4. Combine the transcriptions.
5. Optionally, recombine subtitles with the original video.

This method ensures efficient handling of large video files, both in terms of performance and memory.

==========

Large files of Audio File

==========

Handling large audio files like 4-hour-long WAV or MP3 files requires special techniques to ensure efficient processing, especially when using tools like Whisper. Here's how to handle such files:

---

1. Split the Audio File into Smaller Segments
Processing smaller segments reduces memory usage and allows for parallel processing.

Tools for Splitting:
- **FFmpeg (Command Line):**
  ```bash
  ffmpeg -i input.wav -f segment -segment_time 1800 -c copy output_%03d.wav
  ```
  This splits the file into 30-minute (1800 seconds) segments.

- **Python (Using Pydub Library):**
  ```python
  from pydub import AudioSegment
  from pydub.playback import split_on_silence

  # Load the audio file
  audio = AudioSegment.from_file("input.wav")
  ```

```
# Split into 30-minute chunks (in milliseconds)
chunk_length_ms = 30 * 60 * 1000
chunks = [audio[i:i+chunk_length_ms] for i in range(0, len(audio), chunk_length_ms)]

# Export each chunk
for i, chunk in enumerate(chunks):
    chunk.export(f"output_{i}.wav", format="wav")
```

---

2. Process Each Segment Separately
Once the file is split, you can process each segment individually using Whisper or another transcription tool.

Example with Whisper:
```python
import whisper

model = whisper.load_model("base")
for i in range(num_segments):  # Adjust based on the number of segments
    result = model.transcribe(f"output_{i}.wav")
    with open(f"output_{i}.srt", "w") as srt_file:
        srt_file.write(result["text"])
```

---

3. Handle Memory Issues
Large files can still pose memory issues, even when split:
- Use a smaller Whisper model (`base` or `tiny`) for lower memory usage.
- Process files sequentially instead of parallelly if memory is constrained.

---

4. Merge Results
After processing all segments, combine the transcriptions to form a continuous output.

Example for Combining SRT Files:
```python
with open("final_transcription.srt", "w") as final_file:
    for i in range(num_segments):
        with open(f"output_{i}.srt", "r") as segment_file:
            final_file.write(segment_file.read())
            final_file.write("\n")
```

---

5. Optimize Audio File Size
Before splitting or processing:
- Convert large WAV files to MP3 to reduce size (lossy compression):
  ```bash
  ffmpeg -i input.wav -vn -ar 44100 -ac 2 -b:a 192k output.mp3
  ```
- Downsample audio for faster processing:
  ```bash
  ffmpeg -i input.wav -ar 16000 output_16k.wav
  ```

By splitting the file, optimizing its size, and processing it incrementally, you can handle long audio files efficiently without running into memory or performance bottlenecks.