```
import torch
import torchaudio

print(torch.__version__)
print(torchaudio.__version__)

import matplotlib.pyplot as plt
```

```
2.6.0+cu124
2.6.0+cu124
```

```
!pip install mir_eval transformers openai-whisper --quiet
```

```
    ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 800.5/800.5 kB 12.4 MB/s eta 0:00:00
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 102.8/102.8 kB 10.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 363.4/363.4 MB 4.2 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 13.8/13.8 MB 121.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 24.6/24.6 MB 98.0 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 883.7/883.7 kB 53.3 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 664.8/664.8 MB 2.1 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 211.5/211.5 MB 5.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 56.3/56.3 MB 13.4 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 127.9/127.9 MB 9.6 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.5/207.5 MB 5.7 MB/s eta 0:00:00
   ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.1/21.1 MB 105.6 MB/s eta 0:00:00
  Building wheel for openai-whisper (pyproject.toml) ... done
```

```
from IPython.display import Audio
from mir_eval import separation
from torchaudio.pipelines import HDEMUCS_HIGH_MUSDB_PLUS
from torchaudio.utils import download_asset
import soundfile as sf
from transformers import Wav2Vec2ForCTC, Wav2Vec2Processor
```

```
bundle = HDEMUCS_HIGH_MUSDB_PLUS
model = bundle.get_model()
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)
sample_rate = bundle.sample_rate
print(f"Sample rate: {sample_rate}")
```

```
Sample rate: 44100
```

```
from torchaudio.transforms import Fade

def separate_sources(
    model,
    mix,
    segment=10.0,
    overlap=0.1,
    device=None,
):
    """
    Apply model to a given mixture. Use fade, and add segments together in order to add model segment by segment.
    """
    if device is None:
        device = mix.device
    else:
        device = torch.device(device)

    batch, channels, length = mix.shape

    chunk_len = int(sample_rate * segment * (1 + overlap))
    start = 0
    end = chunk_len
    overlap_frames = overlap * sample_rate
    fade = Fade(fade_in_len=0, fade_out_len=int(overlap_frames), fade_shape="linear")

    final = torch.zeros(batch, len(model.sources), channels, length, device=device)

    while start < length - overlap_frames:
        chunk = mix[:, :, start:end]
        with torch.no_grad():
            out = model.forward(chunk)
        out = fade(out)
        final[:, :, :, start:end] += out
        if start == 0:
            fade.fade_in_len = int(overlap_frames)
            start += int(chunk_len - overlap_frames)
        else:
            start += chunk_len
        end += chunk_len
        if end >= length:
            fade.fade_out_len = 0
    return final


def plot_spectrogram(stft, title="Spectrogram"):
    magnitude = stft.abs()
    spectrogram = 20 * torch.log10(magnitude + 1e-8).numpy()
    _, axis = plt.subplots(1, 1)
    axis.imshow(spectrogram, cmap="viridis", vmin=-60, vmax=0, origin="lower", aspect="auto")
    axis.set_title(title)
    plt.tight_layout()
```

run the model

```
import torchaudio
test_song = download_asset("/content/test.wav")
waveform, sample_rate = torchaudio.load(test_song)
waveform = waveform.to(device)
mixture = waveform

# parameters
segment: int = 10
overlap = 0.1

print("Separating track")

ref = waveform.mean(0)
waveform = (waveform - ref.mean()) / ref.std()  # normalization

sources = separate_sources(
    model,
    waveform[None],
    device=device,
    segment=segment,
    overlap=overlap,
)[0]
sources = sources * ref.std() + ref.mean()

sources_list = model.sources
sources = list(sources)

audios = dict(zip(sources_list, sources))
```

```
Separating track
```

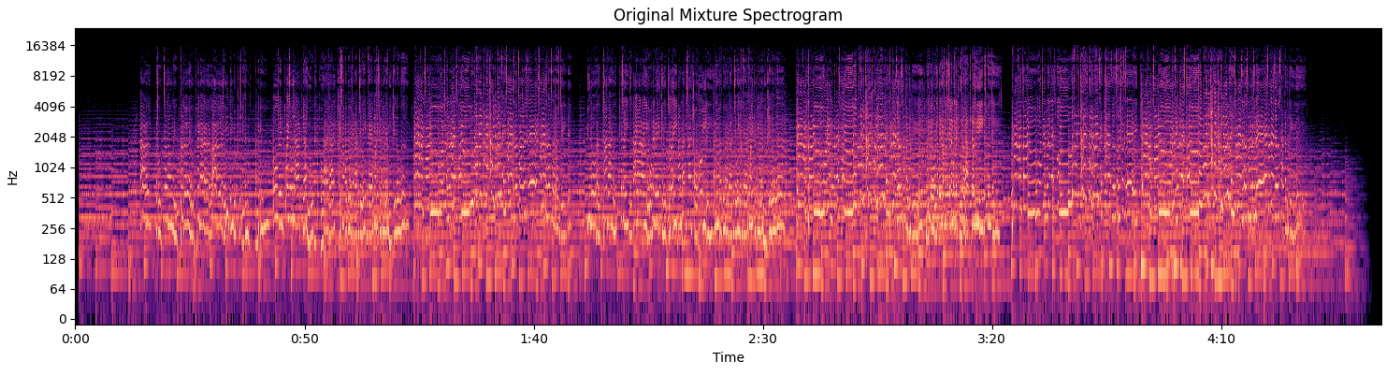
```
N_FFT = 4096
N_HOP = 4
stft = torchaudio.transforms.Spectrogram(
    n_fft=N_FFT,
    hop_length=N_HOP,
    power=None,
)
```

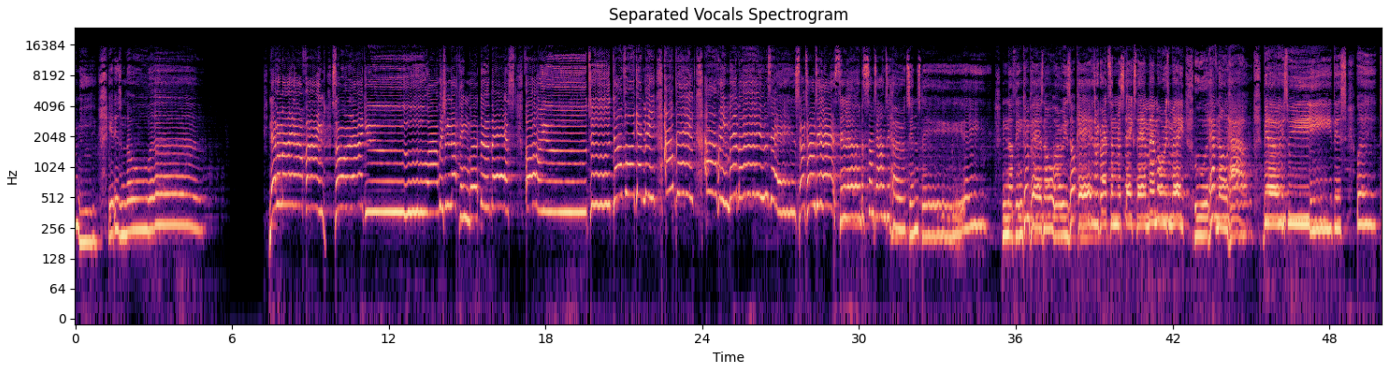
```
segment_start = 150
segment_end = 200

frame_start = segment_start * sample_rate
frame_end = segment_end * sample_rate

drums_spec = audios["drums"][:, frame_start:frame_end].cpu()
bass_spec = audios["bass"][:, frame_start:frame_end].cpu()
vocals_spec = audios["vocals"][:, frame_start:frame_end].cpu()
```

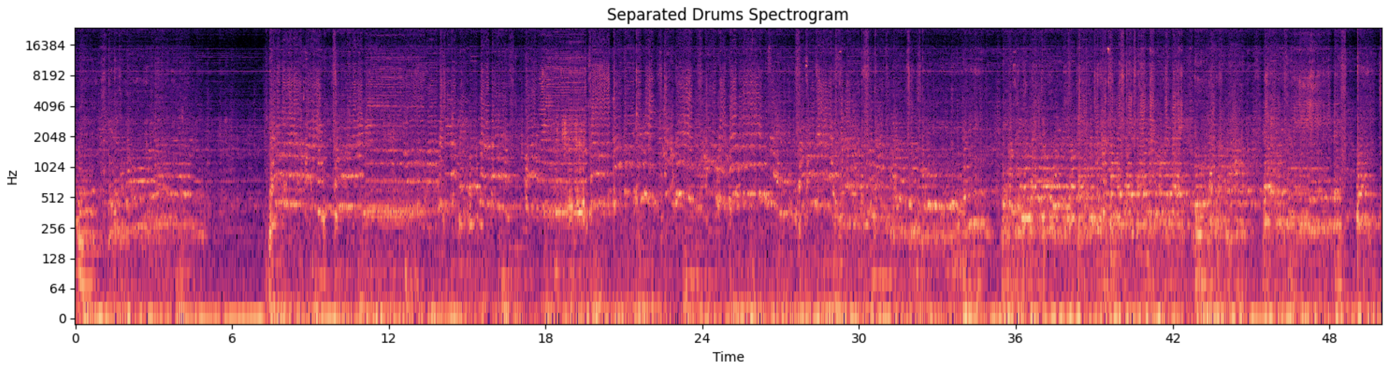
```
other_spec = audios["other"][:, frame_start:frame_end].cpu()

#  plot or listen to these separated stems
from IPython.display import Audio


Audio(vocals_spec.squeeze(0).numpy(), rate=sample_rate)
```

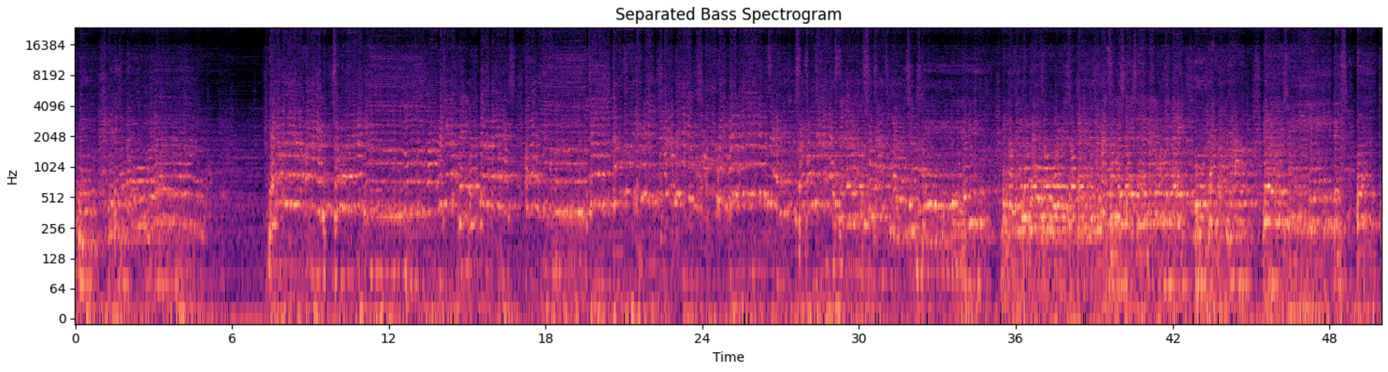
0:00 / 0:50

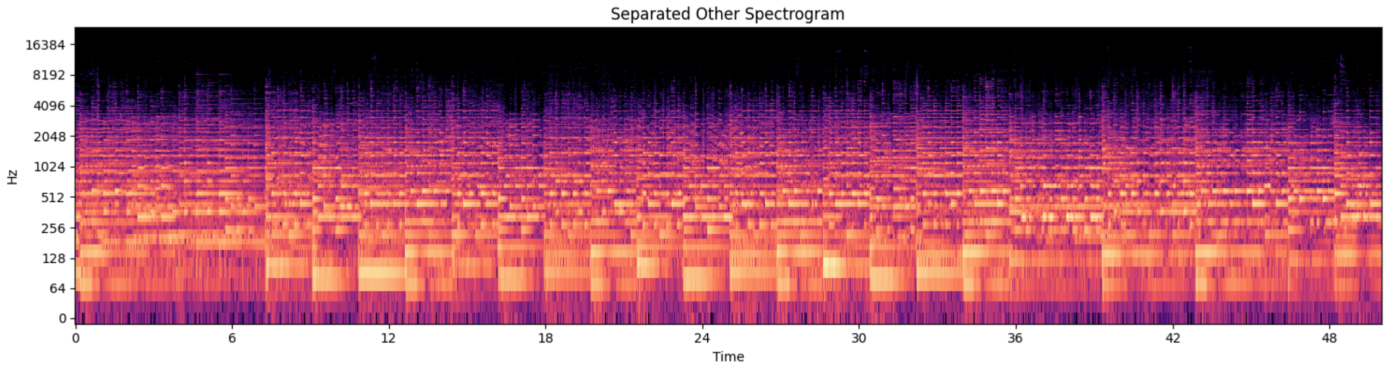
```
import librosa
import librosa.display
import matplotlib.pyplot as plt
import numpy as np

#load test wav file
mix, sr = librosa.load("/content/test.wav", sr=sample_rate, mono=True)

#compute the spectrogram waves
n_fft = 2048
hop_length = 512

#get mixture wave
mix_stft = librosa.stft(mix, n_fft=n_fft, hop_length=hop_length)
mix_db = librosa.amplitude_to_db(np.abs(mix_stft), ref=np.max)

#average separated stems
vocals_stft = librosa.stft(vocals_spec.squeeze(0).numpy(), n_fft=n_fft, hop_length=hop_length)
drums_stft = librosa.stft(drums_spec.squeeze(0).numpy(), n_fft=n_fft, hop_length=hop_length)
bass_stft = librosa.stft(bass_spec.squeeze(0).numpy(), n_fft=n_fft, hop_length=hop_length)
other_stft = librosa.stft(other_spec.squeeze(0).numpy(), n_fft=n_fft, hop_length=hop_length)


vocals_stft = np.mean(vocals_stft, axis=0)
drums_stft = np.mean(drums_stft, axis=0)
bass_stft = np.mean(bass_stft, axis=0)
other_stft = np.mean(other_stft, axis=0)


vocals_db = librosa.amplitude_to_db(np.abs(vocals_stft), ref=np.max)
drums_db = librosa.amplitude_to_db(np.abs(drums_stft), ref=np.max)
bass_db = librosa.amplitude_to_db(np.abs(bass_stft), ref=np.max)
other_db = librosa.amplitude_to_db(np.abs(other_stft), ref=np.max)

#plot spectrograms
fig, axs = plt.subplots(5, 1, figsize=(15, 20))

librosa.display.specshow(mix_db, sr=sr, hop_length=hop_length, x_axis='time', y_axis='log', ax=axs[0])
axs[0].set_title('Original Mixture Spectrogram')

librosa.display.specshow(vocals_db, sr=sr, hop_length=hop_length, x_axis='time', y_axis='log', ax=axs[1])
axs[1].set_title('Separated Vocals Spectrogram')

librosa.display.specshow(drums_db, sr=sr, hop_length=hop_length, x_axis='time', y_axis='log', ax=axs[2])
axs[2].set_title('Separated Drums Spectrogram')

librosa.display.specshow(bass_db, sr=sr, hop_length=hop_length, x_axis='time', y_axis='log', ax=axs[3])
axs[3].set_title('Separated Bass Spectrogram')

librosa.display.specshow(other_db, sr=sr, hop_length=hop_length, x_axis='time', y_axis='log', ax=axs[4])
axs[4].set_title('Separated Other Spectrogram')

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


Original Mixture Spectrogram
16384
8192
4096
2048
1024
512
256
128
64
0
Hz
0:00
0:50
1:40
2:30
3:20
4:10
Time


Separated Vocals Spectrogram
16384
8192
4096
2048
1024
512
256
128
64
0
Hz
0
6
12
18
24
30
36
42
48
Time


Separated Drums Spectrogram
16384
8192
4096
2048
1024
512
256
128
64
0
Hz
0
6
12
18
24
30
36
42
48
Time


Separated Bass Spectrogram
16384
8192
4096
2048
1024
512
256
128
64
0
Hz
0
6
12
18
24
30
36
42
48
Time


Separated Other Spectrogram
16384
8192
4096
2048
1024
512
256
128
64
0
Hz
0
6
12
18
24
30
36
42
48
Time

```
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base-960h")
asr_model = Wav2Vec2ForCTC.from_pretrained("facebook/wav2vec2-base-960h")
asr_model.eval()

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
asr_model.to(device)
```

```
preprocessor_config.json: 100%                                    159/159 [00:00<00:00, 8.29kB/s]

tokenizer_config.json: 100%                                    163/163 [00:00<00:00, 7.19kB/s]

config.json: 100%                                    1.60k/1.60k [00:00<00:00, 40.7kB/s]

vocab.json: 100%                                    291/291 [00:00<00:00, 24.0kB/s]

special_tokens_map.json: 100%                                    85.0/85.0 [00:00<00:00, 9.38kB/s]

model.safetensors: 100%                                    378M/378M [00:04<00:00, 58.1MB/s]
Some weights of Wav2Vec2ForCTC were not initialized from the model checkpoint at facebook/wav2vec2-base-960h and are newly i
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Wav2Vec2ForCTC(
  (wav2vec2): Wav2Vec2Model(
    (feature_extractor): Wav2Vec2FeatureEncoder(
      (conv_layers): ModuleList(
        (0): Wav2Vec2GroupNormConvLayer(
          (conv): Conv1d(1, 512, kernel_size=(10,), stride=(5,), bias=False)
          (activation): GELUActivation()
          (layer_norm): GroupNorm(512, 512, eps=1e-05, affine=True)
        )
        (1-4): 4 x Wav2Vec2NoLayerNormConvLayer(
          (conv): Conv1d(512, 512, kernel_size=(3,), stride=(2,), bias=False)
          (activation): GELUActivation()
        )
        (5-6): 2 x Wav2Vec2NoLayerNormConvLayer(
          (conv): Conv1d(512, 512, kernel_size=(2,), stride=(2,), bias=False)
          (activation): GELUActivation()
        )
      )
    )
    (feature_projection): Wav2Vec2FeatureProjection(
      (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
      (projection): Linear(in_features=512, out_features=768, bias=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): Wav2Vec2Encoder(
      (pos_conv_embed): Wav2Vec2PositionalConvEmbedding(
        (conv): ParametrizedConv1d(
          768, 768, kernel_size=(128,), stride=(1,), padding=(64,), groups=16
          (parametrizations): ModuleDict(
            (weight): ParametrizationList(
              (0): _WeightNorm()
            )
          )
        )
        (padding): Wav2Vec2SamePadLayer()
        (activation): GELUActivation()
      )
      (layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
      (layers): ModuleList(
        (0-11): 12 x Wav2Vec2EncoderLayer(
          (attention): Wav2Vec2SdpaAttention(
            (k_proj): Linear(in_features=768, out_features=768, bias=True)
            (v_proj): Linear(in_features=768, out_features=768, bias=True)
            (q_proj): Linear(in_features=768, out_features=768, bias=True)
            (out_proj): Linear(in_features=768, out_features=768, bias=True)
          )
          (dropout): Dropout(p=0.1, inplace=False)
          (layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
          (feed_forward): Wav2Vec2FeedForward(
            (intermediate_dropout): Dropout(p=0.1, inplace=False)
            (intermediate_dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
            (output_dense): Linear(in_features=3072, out_features=768, bias=True)
            (output_dropout): Dropout(p=0.1, inplace=False)
          )
          (final_layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        )
      )
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (lm_head): Linear(in_features=768, out_features=32, bias=True)
)
```

```
vocals_audio = audios["vocals"].cpu()
```

```
    # If stereo → make mono
    if vocals_audio.shape[0] > 1:
        vocals_audio = vocals_audio.mean(dim=0, keepdim=True)

    # Resample to 16kHz (Wav2Vec2 requirement)
    target_sr = 16000
    sample_rate = 44100  # your original sample rate

    resampler = torchaudio.transforms.Resample(orig_freq=sample_rate, new_freq=target_sr)
    vocals_audio = resampler(vocals_audio)

    # Remove batch dimension
    vocals_audio = vocals_audio.squeeze(0)

    # Set chunk size
    chunk_duration = 15  # seconds
    chunk_size = chunk_duration * target_sr  # samples per chunk

    # Chunk and transcribe
    full_transcript = []

    print("🎤 Starting chunked transcription...\n")

    for start in range(0, vocals_audio.shape[0], chunk_size):
        end = min(start + chunk_size, vocals_audio.shape[0])
        chunk = vocals_audio[start:end]

        # Skip very short chunks
        if chunk.shape[0] < target_sr * 2:  # less than 2 seconds
            continue

        # Prepare input
        input_values = processor(chunk, sampling_rate=target_sr, return_tensors="pt").input_values
        input_values = input_values.to(device)

        # Predict
        with torch.no_grad():
            logits = asr_model(input_values).logits

        predicted_ids = torch.argmax(logits, dim=-1)
        transcription = processor.decode(predicted_ids[0])

        # Save this chunk's text
        full_transcript.append(transcription)

        print(f"Chunk {start/target_sr:.1f}-{end/target_sr:.1f} sec:\n{transcription}\n")

    # Join all chunks together
    final_text = "\n".join(full_transcript)

    # Output final full text
    print("\n🎶 Final Full Predicted Lyrics:\n")
    print(final_text)
```

```
🎤 Starting chunked transcription...

Chunk 0.0-15.0 sec:
O

Chunk 15.0-30.0 sec:
H H THAT YOOER SETTLE DON THAT YE FOUN A G ANY O MARRY A

Chunk 30.0-45.0 sec:
AA HA DAT'S YE OLD DREAMS CAME TOO GUESS SHE GAVE THINGS I DN GIVE TO YOU

Chunk 45.0-60.0 sec:
OLD FREN WHYN SOLD SHI AIN'T LIKE YO D HOLD BAG  HI

Chunk 60.0-75.0 sec:
T FROM E LID I HATE TO TURN OUT BATTLE OF THE BL UN INVIT IT BUT I COULDN'T STAY AWAAYY I COULDN'T FIGHT IT I I HOPED YOU'D

Chunk 75.0-90.0 sec:
IT IS NOOW NEVER MI O FINE SO IKE YOU A WHIS NOTHING BU HE BE

Chunk 90.0-105.0 sec:
ES AY D DOFF  GEMBAT AAN A REMEMBOU SAI SOMETIMES ES AND LO

Chunk 105.0-120.0 sec:
BUT SOMETIMES IT URS INC TEAD SOMETIMES IT LESS AN LUS BUT SOMETIMES IT OSIN STA A
```