Separacion de audios

```
%%capture
!git clone https://github.com/speechbrain/speechbrain/
!pip install speechbrain
!pip install transformers
```

```
import speechbrain as sb
from speechbrain.dataio.dataio import read_audio
from IPython.display import Audio
import soundfile as sf
```

```
from IPython.display import Audio
from IPython.core.display import display
```

```
from speechbrain.pretrained import SepformerSeparation as separator

model = separator.from_hparams(source="speechbrain/sepformer-wsj02mix", savedir='pretrained_models/sepformer-wsj02mix')
est_sources = model.separate_file(path='speechbrain/sepformer-wsj02mix/test_mixture.wav')
```

```
hyperparams.yaml: 100%     1.51k/1.51k [00:00<00:00, 19.0kB/s]
masknet.ckpt: 100%     113M/113M [00:01<00:00, 117MB/s]
encoder.ckpt: 100%     17.3k/17.3k [00:00<00:00, 625kB/s]
decoder.ckpt: 100%     17.2k/17.2k [00:00<00:00, 383kB/s]
test_mixture.wav: 100%     66.2k/66.2k [00:00<00:00, 1.48MB/s]
```

```
import math
import os
```

```
def escalar_audio(audio1_path, audio2_path, overlap_percentage = 0, volume_ratio = 0):
    # Cargar los archivos de audio
    audio1, sr = librosa.load(audio1_path, sr=None)
    audio2, sr = librosa.load(audio2_path, sr=None)

    # Obtener la duración de cada audio
    duration1 = len(audio1) / sr
    duration2 = len(audio2) / sr

    # Determinar el audio más largo y el más corto
    if duration1 >= duration2:
        long_audio = audio1
        short_audio = audio2
    else:
        long_audio = audio2
        short_audio = audio1

    # Asegurarse de que el audio más corto tenga la misma duración que el más largo
    long_audio = long_audio[:len(short_audio)]

    audio1 = short_audio
    audio2 = long_audio

    # Ajustar el volumen del segundo audio según la razón en dB
    rms_audio1 = librosa.feature.rms(y = audio1).mean()
    rms_audio2 = librosa.feature.rms(y = audio2).mean()

    # Calcular el valor RMS de cada audio
    dB_rms_audio1 = 10*math.log(rms_audio1)
    dB_rms_audio2 = 10*math.log(rms_audio2)

    # Calcular la relación de amplitud entre los audios
    relacion_amplitud = volume_ratio - (dB_rms_audio1 - dB_rms_audio2)
    relacion_amplitud += rms_audio1

    # Ajustar el volumen del audio2 utilizando el factor de ajuste
    factor_ajuste = math.pow(10, relacion_amplitud / 10)
    audio1_ajustado = audio1 * factor_ajuste


    # Calcular la duración de la superposición
    overlap_samples = int(len(audio1_ajustado) * overlap_percentage)

    audio1_padded = np.pad(audio1_ajustado, (overlap_samples, 0), mode='constant')


    # Obtener la duración de cada audio
    duration1 = len(audio1_padded) / sr
    duration2 = len(audio2) / sr

    # Determinar el audio más largo y el más corto
    if duration1 >= duration2:
        long_audio = audio1_padded
        short_audio = audio2
    else:
```

```
        long_audio = audio2
        short_audio = audio1_padded

    # Asegurarse de que el audio más corto tenga la misma duración que el más largo
    short_audio = np.pad(short_audio, (0, len(long_audio) - len(short_audio)))


    # Combinar los audios
    merged_audio = long_audio + short_audio

    # Normalizar el audio
    merged_audio = librosa.util.normalize(merged_audio)

    return long_audio, short_audio, merged_audio
```

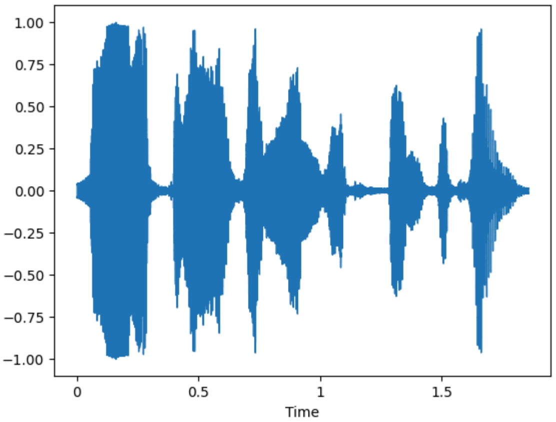
```
audio1, sr1 = librosa.load(est_sources, sr=None)
display(Audio(audio1, rate=sr1))
librosa.display.waveshow(audio1, sr=sr1)
```

```
!pip install pydub
```

```
Collecting pydub
  Downloading pydub-0.25.1-py2.py3-none-any.whl (32 kB)
Installing collected packages: pydub
Successfully installed pydub-0.25.1
```

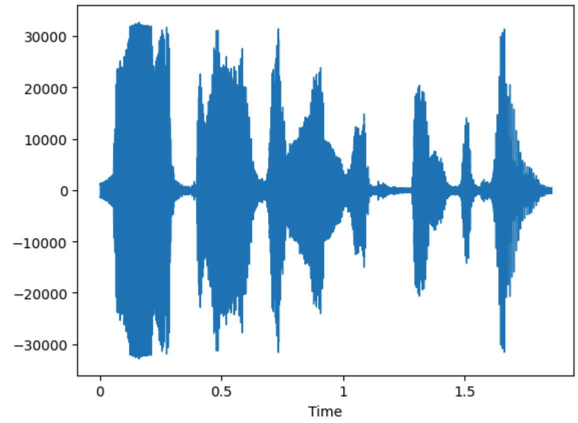
```
from pydub import AudioSegment

def calculate_volume(audio_path):
    audio = AudioSegment.from_file(audio_path)
    return audio.dBFS

def scale_audio(audio_path, target_dBFS):
    audio = AudioSegment.from_file(audio_path)
    current_dBFS = audio.dBFS
    diff_dBFS = target_dBFS - current_dBFS
    scaled_audio = audio + diff_dBFS
    return scaled_audio

# Ejemplo de uso
audio_path = SAVEE + '/' + savee_dir_list[5]

# Calcular el volumen
volume = calculate_volume(audio_path)
print("Volumen actual:", volume, "dBFS")

audio, _ = librosa.load(audio_path, sr=None)
librosa.display.waveshow(audio, sr=8000)
```

```
Volumen actual: -11.659523405651292 dBFS
<librosa.display.AdaptiveWaveplot at 0x7f198ccc3520>
```



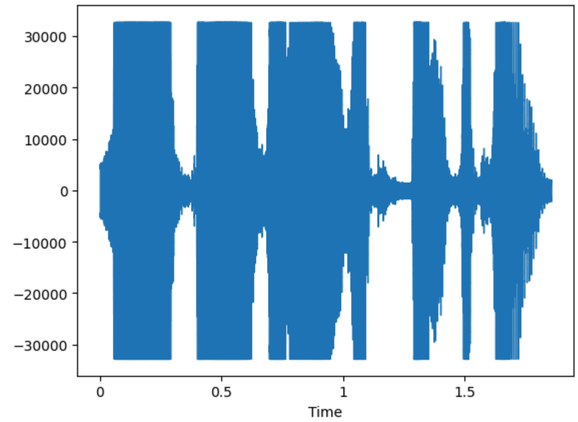

```
volume = calculate_volume(audio_path)
print("Volumen actual:", volume, "dBFS")

db = 0

audio = AudioSegment.from_file(audio_path)
sonido = np.array(audio.get_array_of_samples()).astype(np.float32)
librosa.display.waveshow(sonido, sr=8000)
```

```
Volumen actual: -11.659523405651292 dBFS
<librosa.display.AdaptiveWaveplot at 0x7f198f8be050>
```



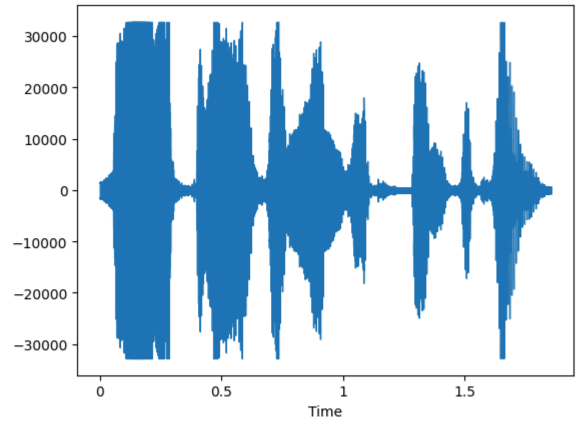

```
change_dBFS = db - volume
audio = audio.apply_gain(change_dBFS)
librosa.display.waveshow(np.array(audio.get_array_of_samples()).astype(np.float32), sr=8000)
```

```
<librosa.display.AdaptiveWaveplot at 0x7f199001d0f0>
```




```
# Escalar el volumen a -10 dBFS
target_dBFS = -10
scaled_audio = scale_audio(audio_path, target_dBFS)
print(scaled_audio)
scaled_audio = np.array(scaled_audio.get_array_of_samples()).astype(np.float32)
librosa.display.waveshow(scaled_audio, sr=8000)
print("Audio escalado guardado como audio_escalado.wav")
```

```
<pydub.audio_segment.AudioSegment object at 0x7f7b62ed4460>
Audio escalado guardado como audio_escalado.wav
```




```
sf.write('person_one.wav', est_sources[:, :, 0].detach().cpu().squeeze(), sr1)
sf.write('person_two.wav', est_sources[:, :, 1].detach().cpu().squeeze(), sr1)
```

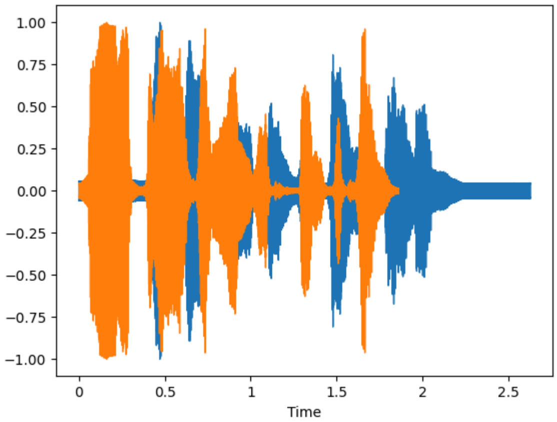
```
RAVDESS = "/content/drive/MyDrive/Tesis/RCST_8k/Ravdess"
CREMA = "/content/drive/MyDrive/Tesis/RCST_8k/CremaD"
TESS = "/content/drive/MyDrive/Tesis/RCST_8k/TESS"
SAVEE = "/content/drive/MyDrive/Tesis/RCST_8k/Surrey"
```

```
savee_dir_list = os.listdir(SAVEE)
savee_dir_list = [file for file in savee_dir_list if file.endswith('.wav')]
print(savee_dir_list[5])
```

```
DC_a04.wav
```

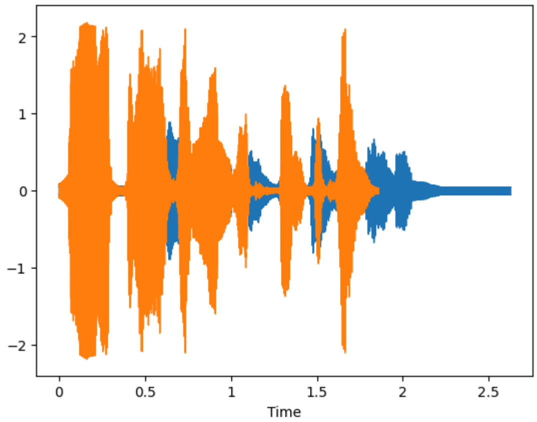
```
audio_path1 = SAVEE + '/' + savee_dir_list[5]
audio_path2 = SAVEE + '/' + savee_dir_list[10]

audio2, sr2 = librosa.load(audio_path2, sr=None)
librosa.display.waveshow(audio2, sr=8000)
volume2 = calculate_volume(audio_path2)
print("Volumen actual:", volume2, "dBFS")

audio3, sr3 = librosa.load(audio_path1, sr=None)
librosa.display.waveshow(audio3, sr=8000)
volume1 = calculate_volume(audio_path1)
print("Volumen actual:", volume1, "dBFS")
```

```
Volumen actual: -15.056927707858287 dBFS
Volumen actual: -11.659523405651292 dBFS
```




```
volume_ratio = 0

relacion_amplitud = volume_ratio - (volume2 - volume1)
# relacion_amplitud += rms_audio1

# Ajustar el volumen del audio2 utilizando el factor de ajuste
factor_ajuste = math.pow(10, relacion_amplitud / 10)
audio3 = audio3 * factor_ajuste

librosa.display.waveshow(audio2, sr=8000)
volume2 = calculate_volume(audio_path2)
print("Volumen actual:", volume2, "dBFS")

librosa.display.waveshow(audio3, sr=8000)
volume1 = calculate_volume(audio_path1)
print("Volumen actual:", volume1, "dBFS")
```

```
Volumen actual: -15.056927707858287 dBFS
Volumen actual: -11.659523405651292 dBFS
```



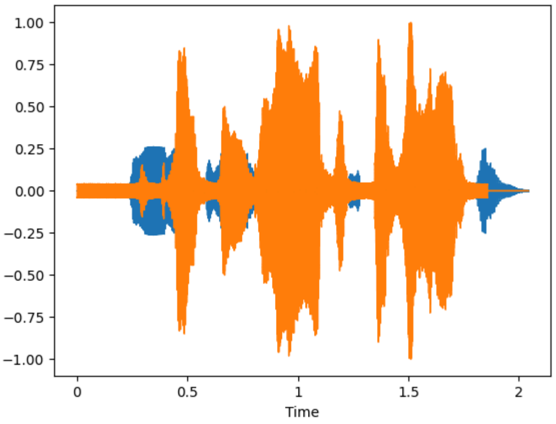

```
audio_path1 = SAVEE + '/' + savee_dir_list[5]
audio_path2 = SAVEE + '/' + savee_dir_list[54]
print(audio_path2)
print(audio_path1)

audio1, audio2, merge = escalar_audio(audio_path1, audio_path2,0.1, -5)

librosa.display.waveshow(audio1, sr=8000)
display(Audio(audio1, rate=8000))
librosa.display.waveshow(audio2, sr=8000)
# librosa.display.waveshow(merge, sr=8000)
display(Audio(audio2, rate=8000))
display(Audio(merge, rate=8000))
```

```
/content/drive/MyDrive/Tesis/RCST_8k/Surrey/DC_f13.wav
/content/drive/MyDrive/Tesis/RCST_8k/Surrey/DC_a04.wav
```

0:00 / 0:02

0:00 / 0:02

0:00 / 0:02



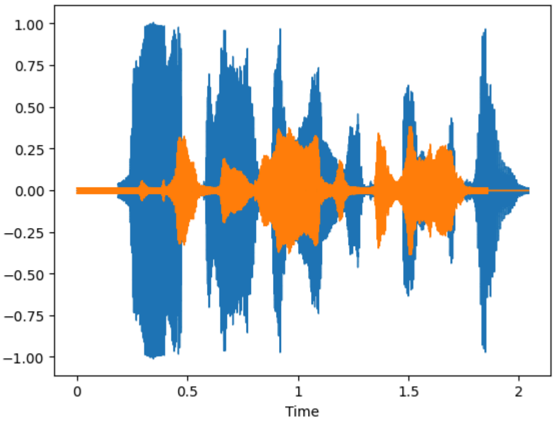

```
audio_path1 = SAVEE + '/' + savee_dir_list[5]
audio_path2 = SAVEE + '/' + savee_dir_list[54]
print(audio_path2)
print(audio_path1)

audio1, audio2, merge = escalar_audio(audio_path1, audio_path2,0.1, 5)

librosa.display.waveshow(audio1/2.6, sr=8000)
display(Audio(audio1, rate=8000))
librosa.display.waveshow(audio2/2.6, sr=8000)
# librosa.display.waveshow(merge, sr=8000)
display(Audio(audio2, rate=8000))
display(Audio(merge, rate=8000))
```

```
/content/drive/MyDrive/Tesis/RCST_8k/Surrey/DC_f13.wav
/content/drive/MyDrive/Tesis/RCST_8k/Surrey/DC_a04.wav
```

0:00 / 0:02

0:00 / 0:02

0:00 / 0:02



Pruebas de audio

Librerias

```
import os
import pandas as pd
import numpy as np
from numpy import asarray
from numpy import save
import shutil
import math
import librosa
import librosa.display
import soundfile as sf
import numpy as np
import random
```

```
from keras.models import model_from_json
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from IPython.display import Audio
from IPython.core.display import display
from scipy.io import wavfile
import matplotlib.pyplot as plt
import pprint
import statistics
import seaborn as sns
from tqdm.notebook import tqdm
```

```
import subprocess

try:
        import speechmetrics as sm
except ImportError:
        print('Instalando paquetes...')
        subprocess.check_call(["python", '-m', 'pip', 'install', 'git+https://github.com/aliutkus/speechmetrics#egg=speechmetrics[cpu]
        import speechmetrics as sm
```

```
Instalando paquetes...
```

```
%%capture
!git clone https://github.com/speechbrain/speechbrain/
!pip install speechbrain
!pip install transformers
import speechbrain as sb
from speechbrain.dataio.dataio import read_audio
from speechbrain.pretrained import SepformerSeparation as separator
```

```
DEBUG:speechbrain.utils.checkpoints:Registered checkpoint save hook for _speechbrain_save
DEBUG:speechbrain.utils.checkpoints:Registered checkpoint load hook for _speechbrain_load
DEBUG:speechbrain.utils.checkpoints:Registered checkpoint save hook for save
DEBUG:speechbrain.utils.checkpoints:Registered checkpoint load hook for load
DEBUG:speechbrain.utils.checkpoints:Registered checkpoint save hook for _save
DEBUG:speechbrain.utils.checkpoints:Registered checkpoint load hook for _recover
```

Pruebas

```
window = 8
metrics = sm.load(['relative.pesq', 'relative.stoi', 'relative.sisdr'], window)
```

```
Loaded  speechmetrics.relative.pesq
Loaded  speechmetrics.relative.sisdr
Loaded  speechmetrics.relative.stoi
```

```
model = separator.from_hparams(source="speechbrain/sepformer-libri2mix", savedir='pretrained_models/sepformer-libri2mix')
```

```
INFO:speechbrain.utils.fetching:Fetch hyperparams.yaml: Fetching from HuggingFace Hub 'speechbrain/sepformer-libri2mix' if not cached
hyperparams.yaml:       1.47k/? [00:00<00:00, 123kB/s]
DEBUG:speechbrain.utils.fetching:Fetch: Local file found, creating symlink '/root/.cache/huggingface/hub/models--speechbrain--sepformer-libri2mix/snapshots/eb43c5bfbb2aa6546
INFO:speechbrain.utils.fetching:Fetch custom.py: Fetching from HuggingFace Hub 'speechbrain/sepformer-libri2mix' if not cached
DEBUG:speechbrain.utils.parameter_transfer:Collecting files (or symlinks) for pretraining in pretrained_models/sepformer-libri2mix.
INFO:speechbrain.utils.fetching:Fetch encoder.ckpt: Fetching from HuggingFace Hub 'speechbrain/sepformer-libri2mix' if not cached
encoder.ckpt: 100%                                  17.3k/17.3k [00:00<00:00, 930kB/s]
DEBUG:speechbrain.utils.fetching:Fetch: Local file found, creating symlink '/root/.cache/huggingface/hub/models--speechbrain--sepformer-libri2mix/snapshots/eb43c5bfbb2aa6546
DEBUG:speechbrain.utils.parameter_transfer:Set local path in self.paths["encoder"] = /content/pretrained_models/sepformer-libri2mix/encoder.ckpt
INFO:speechbrain.utils.fetching:Fetch masknet.ckpt: Fetching from HuggingFace Hub 'speechbrain/sepformer-libri2mix' if not cached
masknet.ckpt: 100%                                  113M/113M [00:00<00:00, 226MB/s]
DEBUG:speechbrain.utils.fetching:Fetch: Local file found, creating symlink '/root/.cache/huggingface/hub/models--speechbrain--sepformer-libri2mix/snapshots/eb43c5bfbb2aa6546
DEBUG:speechbrain.utils.parameter_transfer:Set local path in self.paths["masknet"] = /content/pretrained_models/sepformer-libri2mix/masknet.ckpt
INFO:speechbrain.utils.fetching:Fetch decoder.ckpt: Fetching from HuggingFace Hub 'speechbrain/sepformer-libri2mix' if not cached
decoder.ckpt: 100%                                  17.3k/17.3k [00:00<00:00, 723kB/s]
DEBUG:speechbrain.utils.fetching:Fetch: Local file found, creating symlink '/root/.cache/huggingface/hub/models--speechbrain--sepformer-libri2mix/snapshots/eb43c5bfbb2aa6546
DEBUG:speechbrain.utils.parameter_transfer:Set local path in self.paths["decoder"] = /content/pretrained_models/sepformer-libri2mix/decoder.ckpt
INFO:speechbrain.utils.parameter_transfer:Loading pretrained files for: encoder, masknet, decoder
DEBUG:speechbrain.utils.parameter_transfer:Redirecting (loading from local path): encoder -> /content/pretrained_models/sepformer-libri2mix/encoder.ckpt
DEBUG:speechbrain.utils.parameter_transfer:Redirecting (loading from local path): masknet -> /content/pretrained_models/sepformer-libri2mix/masknet.ckpt
DEBUG:speechbrain.utils.parameter_transfer:Redirecting (loading from local path): decoder -> /content/pretrained_models/sepformer-libri2mix/decoder.ckpt
```

```
from pydub import AudioSegment

# 1. Carga tu archivo de audio (puede ser mp3, wav, ogg, etc.)
# Asegúrate de que el archivo 'mi_audio.mp3' esté en la misma carpeta que tu script.
try:
    audio_original = AudioSegment.from_file("mix_grab3.wav")
except FileNotFoundError:
    print("Error: El archivo 'mi_audio.mp3' no fue encontrado.")
    print("Asegúrate de que el nombre del archivo es correcto y está en la misma carpeta.")
else:
    # 2. Define la duración del corte en milisegundos
    # pydub trabaja con milisegundos, así que 3 segundos = 3000 ms
    duracion_corte_ms = 3 * 1000

    # 3. Realiza el corte (desde el inicio hasta los 3000 ms)
    # Se usa una sintaxis de rebanado, igual que con las listas de Python
    corte_de_audio = audio_original[:duracion_corte_ms]

    # 4. Exporta el audio cortado a un nuevo archivo
    corte_de_audio.export("mix_grab65.wav", format="wav")
```

```
    print("✅ ¡Audio cortado exitosamente! Se ha guardado como 'mix_grab45.wav'.")
```

```
✅ ¡Audio cortado exitosamente! Se ha guardado como 'mix_grab45.wav'.
```

```
rate = 8000

est_sources = model.separate_file(path='mix_grab65.wav')
person_two = est_sources[:, :, 1].detach().cpu().squeeze()
person_one = est_sources[:, :, 0].detach().cpu().squeeze()

sf.write('person_one.wav', person_one, rate)
sf.write('person_two.wav', person_two, rate)
```

```
Resampling the audio from 16000 Hz to 8000 Hz
```

```
from IPython.display import Audio

# Ruta al archivo de audio
ruta_archivo = 'person_one.wav'  # Reemplaza con la ruta de tu archivo

# Reproducir el audio
Audio(ruta_archivo)
```

0:00 / 0:03

```
# Ruta al archivo de audio
ruta_archivo = 'person_two.wav'  # Reemplaza con la ruta de tu archivo

# Reproducir el audio
Audio(ruta_archivo)
```

0:00 / 0:03

```
model = separator.from_hparams(source="speechbrain/sepformer-wsj02mix", savedir='pretrained_models/sepformer-wsj02mix')
```

```
INFO:speechbrain.utils.fetching:Fetch hyperparams.yaml: Fetching from HuggingFace Hub 'speechbrain/sepformer-wsj02mix' if not cached
hyperparams.yaml:   1.51k/? [00:00<00:00, 146kB/s]
DEBUG:speechbrain.utils.fetching:Fetch: Local file found, creating symlink '/root/.cache/huggingface/hub/models--speechbrain--sepformer-wsj02mix/snapshots/3a2826343a10e2d2e8
INFO:speechbrain.utils.fetching:Fetch custom.py: Fetching from HuggingFace Hub 'speechbrain/sepformer-wsj02mix' if not cached
DEBUG:speechbrain.utils.parameter_transfer:Collecting files (or symlinks) for pretraining in pretrained_models/sepformer-wsj02mix.
INFO:speechbrain.utils.fetching:Fetch masknet.ckpt: Fetching from HuggingFace Hub 'speechbrain/sepformer-wsj02mix' if not cached
masknet.ckpt: 100%   113M/113M [00:00<00:00, 343MB/s]
DEBUG:speechbrain.utils.fetching:Fetch: Local file found, creating symlink '/root/.cache/huggingface/hub/models--speechbrain--sepformer-wsj02mix/snapshots/3a2826343a10e2d2e8
DEBUG:speechbrain.utils.parameter_transfer:Set local path in self.paths["masknet"] = /content/pretrained_models/sepformer-wsj02mix/masknet.ckpt
INFO:speechbrain.utils.fetching:Fetch encoder.ckpt: Fetching from HuggingFace Hub 'speechbrain/sepformer-wsj02mix' if not cached
encoder.ckpt: 100%   17.3k/17.3k [00:00<00:00, 1.26MB/s]
DEBUG:speechbrain.utils.fetching:Fetch: Local file found, creating symlink '/root/.cache/huggingface/hub/models--speechbrain--sepformer-wsj02mix/snapshots/3a2826343a10e2d2e8
DEBUG:speechbrain.utils.parameter_transfer:Set local path in self.paths["encoder"] = /content/pretrained_models/sepformer-wsj02mix/encoder.ckpt
INFO:speechbrain.utils.fetching:Fetch decoder.ckpt: Fetching from HuggingFace Hub 'speechbrain/sepformer-wsj02mix' if not cached
decoder.ckpt: 100%   17.2k/17.2k [00:00<00:00, 1.34MB/s]
DEBUG:speechbrain.utils.fetching:Fetch: Local file found, creating symlink '/root/.cache/huggingface/hub/models--speechbrain--sepformer-wsj02mix/snapshots/3a2826343a10e2d2e8
DEBUG:speechbrain.utils.parameter_transfer:Set local path in self.paths["decoder"] = /content/pretrained_models/sepformer-wsj02mix/decoder.ckpt
INFO:speechbrain.utils.parameter_transfer:Loading pretrained files for: masknet, encoder, decoder
DEBUG:speechbrain.utils.parameter_transfer:Redirecting (loading from local path): masknet -> /content/pretrained_models/sepformer-wsj02mix/masknet.ckpt
DEBUG:speechbrain.utils.parameter_transfer:Redirecting (loading from local path): encoder -> /content/pretrained_models/sepformer-wsj02mix/encoder.ckpt
DEBUG:speechbrain.utils.parameter_transfer:Redirecting (loading from local path): decoder -> /content/pretrained_models/sepformer-wsj02mix/decoder.ckpt
```

```
rate = 8000

est_sources = model.separate_file(path='mix_grab65.wav')
person_two = est_sources[:, :, 1].detach().cpu().squeeze()
person_one = est_sources[:, :, 0].detach().cpu().squeeze()

sf.write('person_one2.wav', person_one, rate)
sf.write('person_two2.wav', person_two, rate)
```

```
Resampling the audio from 16000 Hz to 8000 Hz
```

```
from IPython.display import Audio

# Ruta al archivo de audio
ruta_archivo = 'person_one2.wav'  # Reemplaza con la ruta de tu archivo

# Reproducir el audio
Audio(ruta_archivo)
```

0:00 / 0:03

```
# Ruta al archivo de audio
ruta_archivo = 'person_two2.wav'  # Reemplaza con la ruta de tu archivo

# Reproducir el audio
Audio(ruta_archivo)
```

0:00 / 0:03

1. Instalar bibliotecas necesarias

```
# @title 1. Instalar bibliotecas necesarias
!pip install jiwer
!pip install git+https://github.com/openai/whisper.git
```

```
Requirement already satisfied: jiwer in /usr/local/lib/python3.11/dist-packages (4.0.0)
Requirement already satisfied: click>=8.1.8 in /usr/local/lib/python3.11/dist-packages (from jiwer) (8.2.1)
Requirement already satisfied: rapidfuzz>=3.9.7 in /usr/local/lib/python3.11/dist-packages (from jiwer) (3.13.0)
Collecting git+https://github.com/openai/whisper.git
  Cloning https://github.com/openai/whisper.git to /tmp/pip-req-build-wu3yjeu7
  Running command git clone --filter=blob:none --quiet https://github.com/openai/whisper.git /tmp/pip-req-build-wu3yjeu7
  Resolved https://github.com/openai/whisper.git to commit c0d2f624c09dc18e709e37c2ad90c039a4eb72a2
  Installing build dependencies ... done
  Getting requirements to build wheel ... done
  Preparing metadata (pyproject.toml) ... done
Requirement already satisfied: more-itertools in /usr/local/lib/python3.11/dist-packages (from openai-whisper==20250625) (10.7.0)
Requirement already satisfied: numba in /usr/local/lib/python3.11/dist-packages (from openai-whisper==20250625) (0.60.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from openai-whisper==20250625) (1.23.5)
Requirement already satisfied: tiktoken in /usr/local/lib/python3.11/dist-packages (from openai-whisper==20250625) (0.9.0)
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packages (from openai-whisper==20250625) (2.6.0+cu124)
Requirement already satisfied: tqdm in /usr/local/lib/python3.11/dist-packages (from openai-whisper==20250625) (4.67.1)
Requirement already satisfied: triton>=2 in /usr/local/lib/python3.11/dist-packages (from openai-whisper==20250625) (3.2.0)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.11/dist-packages (from numba->openai-whisper==20250625) (0.43.0)
Requirement already satisfied: regex>=2022.1.18 in /usr/local/lib/python3.11/dist-packages (from tiktoken->openai-whisper==20250625) (2024.11.6)
Requirement already satisfied: requests>=2.26.0 in /usr/local/lib/python3.11/dist-packages (from tiktoken->openai-whisper==20250625) (2.32.3)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (3.18.0)
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (4.14.0)
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (3.5)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (3.1.6)
Requirement already satisfied: fsspec in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (2025.3.2)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (12.4.127)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (12.4.127)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (12.4.127)
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (9.1.0.70)
Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (12.4.5.8)
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (11.2.1.3)
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (10.3.5.147)
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (11.6.1.9)
Requirement already satisfied: nvidia-cusparse-cu12==12.3.1.170 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (12.3.1.170)
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (0.6.2)
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (2.21.5)
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (12.4.127)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (12.4.127)
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/dist-packages (from torch->openai-whisper==20250625) (1.13.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.11/dist-packages (from sympy==1.13.1->torch->openai-whisper==20250625) (1.3.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26.0->tiktoken->openai-whisper==20250625) (3.4.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26.0->tiktoken->openai-whisper==20250625) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26.0->tiktoken->openai-whisper==20250625) (2.4.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (from requests>=2.26.0->tiktoken->openai-whisper==20250625) (2025.6.15)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/dist-packages (from jinja2->torch->openai-whisper==20250625) (3.0.2)
```

2. Definir texto de referencia y subir audio

```
# @title 2. Definir texto de referencia y subir audio
from google.colab import files

# ⬇️ IMPORTANTE: Reemplaza este texto con la transcripción correcta de tu audio.
ground_truth1 = "el oso es un animal que impone respeto"
ground_truth2 = "el tiburon es uno de los"

# Obtener la ruta del archivo subido
audio_file_path1 = "/content/person_one.wav"
audio_file_path2 = "/content/person_two.wav"
```

3. Transcribir el audio y calcular el WER

```
# @title 3. Transcribir el audio y calcular el WER
import whisper
import jiwer

# Cargar el modelo de Whisper (puedes usar "tiny", "base", "small", "medium", "large")
model = whisper.load_model("base")

print("\n🔄 Transcribiendo el audio... (esto puede tardar unos minutos)")

# Realizar la transcripción
result = model.transcribe(audio_file_path1, language="es")
hypothesis = result["text"]

print(f"📜 Transcripción generada (hipótesis): '{hypothesis}'")

# Limpiar y normalizar ambos textos para una comparación justa
transformation = jiwer.Compose([
    jiwer.ToLowerCase(),
    jiwer.RemoveMultipleSpaces(),
    jiwer.RemovePunctuation(),
    jiwer.Strip()
])

# Calcular el WER
error = jiwer.wer(
    ground_truth1,
    hypothesis
)
```

```
print("\n--- Resultados del WER ---")
print(f"📊 Tasa de Error de Palabra (WER): {error:.2%}")
print("\nUn WER más bajo es mejor. Un 0% significa una transcripción perfecta.")
```

```
🔄 Transcribiendo el audio... (esto puede tardar unos minutos)
📜 Transcripción generada (hipótesis): ' Con los esos son el mal timpone de respeto solo'

--- Resultados del WER ---
📊 Tasa de Error de Palabra (WER): 112.50%

Un WER más bajo es mejor. Un 0% significa una transcripción perfecta.
```

```
# Realizar la transcripción
result = model.transcribe(audio_file_path2, language="es")
hypothesis = result["text"]

print(f"📜 Transcripción generada (hipótesis): '{hypothesis}'")

# Limpiar y normalizar ambos textos para una comparación justa
transformation = jiwer.Compose([
    jiwer.ToLowerCase(),
    jiwer.RemoveMultipleSpaces(),
    jiwer.RemovePunctuation(),
    jiwer.Strip()
])

# Calcular el WER
error = jiwer.wer(
    ground_truth2,
    hypothesis
)

print("\n--- Resultados del WER ---")
print(f"📊 Tasa de Error de Palabra (WER): {error:.2%}")
print("\nUn WER más bajo es mejor. Un 0% significa una transcripción perfecta.")
```

```
📜 Transcripción generada (hipótesis): ' El pivurón es uno de los anillos.'

--- Resultados del WER ---
📊 Tasa de Error de Palabra (WER): 50.00%

Un WER más bajo es mejor. Un 0% significa una transcripción perfecta.
```