



Faculty of Engineering
Signal Processing Laboratory

Monaural Audio Speaker Separation Using Source-Contrastive Estimation

Guy Varsano
Yuval Varsano

Fourth-year project towards a bachelor's degree in engineering

Advisor: Mordehay Moradi
Academic supervisor: Prof. Sharon Gannot

October 2024

Contents

1. Introduction	5
2. Theoretical background	6
3. Building the dataset	8
3.1 LibriSpeech dataset	8
3.2 RIR	9
3.3 Overlap	11
3.4 Pipeline for building the data	13
4. Mask	19
4.1 Wiener Filter-like Mask (WFM) Implementation.	19
4.2 Ideal Binary Mask (IBM) Implementation.	20
4.3 Weight Mask Implementation.	21
5. Loss function	23
5.1 Objective of the Loss Function.	23
5.2 Reconstruction Error.	23
5.3 Mask Optimization.	24
5.4 Permutation Invariance.	24
5.5 Dynamic Speaker Handling.	24
6. Deep Attractor Network	25
6.1 Components of the DANet Model.	25
6.2 Model Workflow.	27
6.3 Key Characteristics.	28
7. The Training Process	30
8. The Testing Process	32
9. Results	35
9.1 model 1- 2/3 speakers with noise.	35
9.2 SI-SDR Calculation.	36
9.3 model 2- 2 speakers without noise.	37
9.4 model 3- 2 speakers with noise.	42
9.5 model 4- 2 speakers with noise.	47
10. Summary and conclusions	52

11. Audio files	53
11. Bibliography	54

Figures

Figure 1- example of original waveform.	6
Figure 2- example of STFT magnitude spectrogram.	6
Figure 3- k-means.	7
Figure 4- original waveform before and after rir.	8
Figure 5- STFT magnitude spectrogram before and after rir.	9
Figure 6- example of rir.	10
Figure 7- example of CSV file.	16
Figure 8- example for three speakers.	17
Figure 9- log scaled mixture spectrogram.	22
Figure 10- real and estimated WF mask for speaker 1+2.	22
Figure 11- DANet model.	29
Figure 12- k-means clustering.	33
Figure 13- validation loss graph.	35
Figure 14- training loss graph.	37
Figure 15- validation and train SI-SDR graphs.	38
Figure 16- signal and spectrogram of the mix.	38
Figure 17- signal and spectrogram of speaker 1+2.	39
Figure 18- comparison between the separated and original.	40
Figure 19- average SI-SDR by category.	41
Figure 20- validation, train and SI-SDR graphs.	42
Figure 21- signal and spectrogram of the mix.	43
Figure 22- original signal for noise.	43
Figure 23- real and estimated WF mask for speaker 1+2.	44
Figure 24- comparison between the separated and original.	45
Figure 25- average SI-SDR by category.	46
Figure 26- validation, train and SI-SDR graphs.	47

Figure 27- signal and spectrogram of the mix.	48
Figure 28- original and reconstructed signal for noise.	48
Figure 29- real and estimated WF mask for speaker 1+2 and noise.	49
Figure 30- comparison between the separated and original.	50
Figure 31- average SI-SDR by category.	51

1 Introduction

Speaker separation, a fundamental problem in audio processing, addresses the challenge of isolating multiple simultaneous speakers in a recording, commonly known as the "cocktail party problem." Traditionally, this problem has been approached using multi microphone arrays or specialized signal processing techniques. However, with the prevalence of single microphone recordings, particularly with portable devices like smartphones, monaural audio source separation has gained increasing attention.

In the paper, "Monaural Audio Speaker Separation Using Source-Contrastive Estimation", researchers propose an algorithm, termed source-contrastive estimation, which leverages deep neural networks to separate speakers from monaural audio recordings.

During our work on this project, we read and understood the article thoroughly. Our goal is first to realize the article, and then to propose alternative models to improve the results of the existing model.

2 Theoretical Background

STFT

The short-time Fourier transform (STFT), is a Fourier-related transform used to determine the sinusoidal frequency and phase content of local sections of a signal as it changes over time. In practice, the procedure for computing STFTs is to divide a longer time signal into shorter segments of equal length and then compute the Fourier transform separately on each shorter segment. This reveals the Fourier spectrum on each shorter segment.

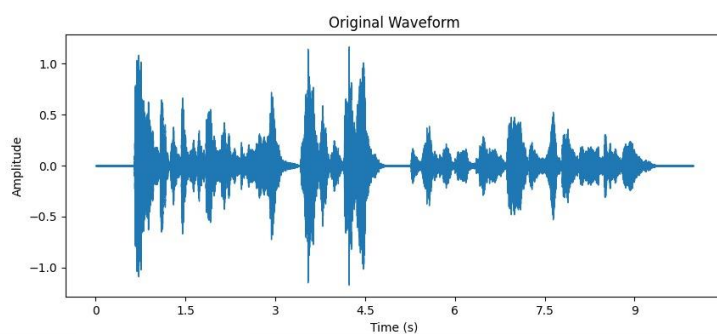


Figure 1

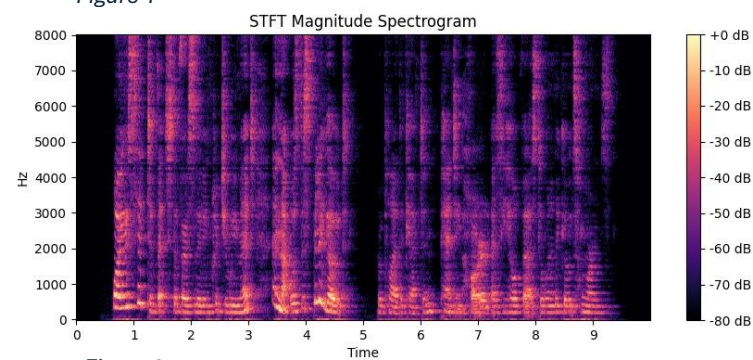


Figure 2

SIR

SIR is a measure used in telecommunications and signal processing to quantify the quality of a received signal relative to the interference present. It's calculated by dividing the power of the desired signal by the power of the interference and noise. A higher SIR indicates better signal quality and less interference.

SNR

SNR is also a measure used in telecommunications and signal processing, but it focuses on the ratio of the desired signal power to the power of the background noise. It's calculated by dividing the power of the signal by the power of the noise. A higher SNR indicates a stronger, clearer signal relative to the noise present, resulting in better communication or data transmission quality.

K-means

K-means clustering is a widely used algorithm for unsupervised learning. It groups data points into clusters based on their similarities. The algorithm begins by initializing cluster centroids randomly. Each data point is then assigned to the nearest centroid, forming clusters. The centroids are updated by calculating the mean of the points in each cluster. This process is repeated until the centroids stabilize and no longer move significantly, indicating that the clusters have been successfully formed.

The number of clusters, K , must be specified beforehand, and the algorithm aims to minimize the variance within each cluster.

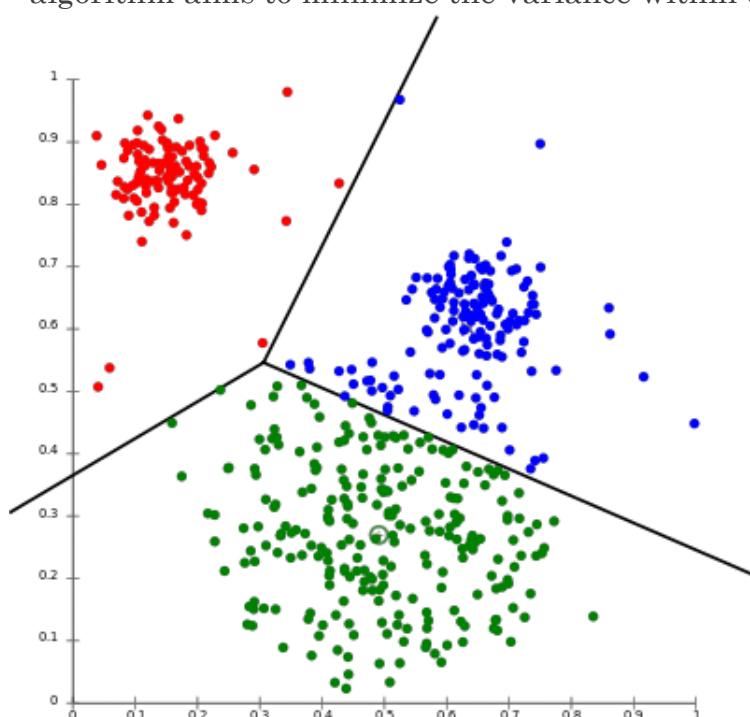


Figure 3

3 Building the dataset

3.1 LibriSpeech dataset

The LibriSpeech corpus is derived from audiobooks that are part of the LibriVox project and contains 1000 hours of speech sampled at 16 kHz. Audiobooks from Project Gutenberg make up the majority of the collection. To produce a corpus of English read speech suitable for training speech recognition systems, LibriSpeech aligns, and segments audiobook read speech with the corresponding book text automatically, filters out segments with noisy transcripts, and produces the corpus.

The LibriSpeech dataset training set is separated into train-clean and train-clean - 360 sets, we used both.

The LibriSpeech dataset testing set is separated into test-clean and test-other categories. ‘Clean’ and ‘other’ categories are assigned depending on how well or challenging Automatic Speech Recognition systems would perform. We used the clean one.

the lower-WER speakers designated as “clean” and the higher-WER speakers designated as “other”.

test-clean - test set, "clean" speech. From the “clean” pool, 20 male and 20 female speakers were drawn at random and assigned to a test set. For each test set speaker, approximately eight minutes of speech are used, for total of approximately 5 hours and 20 minutes each.

train-clean – 100/360 - training set of 100/360 hours "clean" speech. The rest of the audio in the “clean” pool that were not used for the test set was randomly split into two training sets with approximate size 100 and 360 hours respectively. For each speaker in these training sets the amount of speech was limited to 25 minutes, in order to avoid major imbalances in per-speaker audio duration.

3.2 RIR

The Room Impulse Response (RIR) is a fundamental concept in audio signal processing, particularly in the field of room acoustics. It describes the characteristics of how sound propagates within an enclosed space, such as a room, concert hall, or any environment where sound waves interact with surfaces. The RIR captures the complete acoustic behavior of the room, including reflections, reverberation, echoes, and absorption of sound energy by surfaces like walls, floors, and ceilings.

To generate RIR signal we used `pyroomacoustics` – a Python library specifically designed for simulating and analyzing room acoustics.

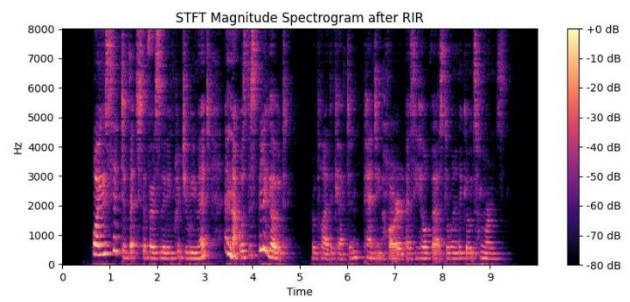
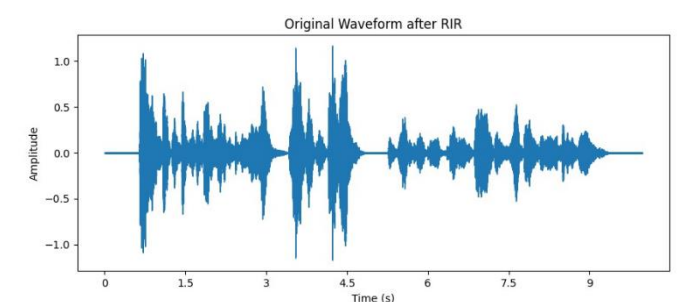
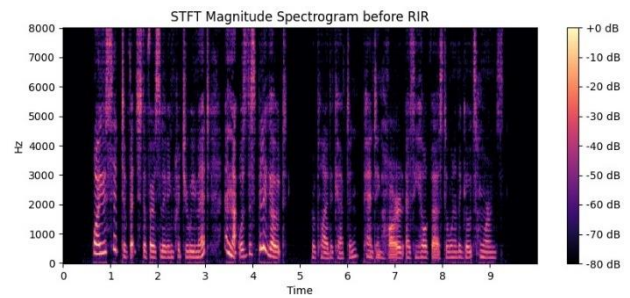
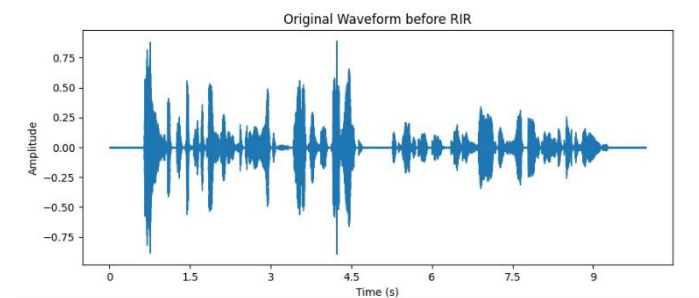


Figure 4

Figure 5

Smearing is present across the time domain, suggesting the presence of the rir.

The process begins by defining a virtual room with specific dimensions, layout, and material properties, where sound sources (such as speakers) and a microphone array are placed. Each sound source is associated with raw audio samples representing the emitted sound. The simulation then generates artificial Room Impulse Responses (RIRs) between the sources and microphones. These RIRs encapsulate the acoustic characteristics of the room, including reflections and reverberation.

RIR:

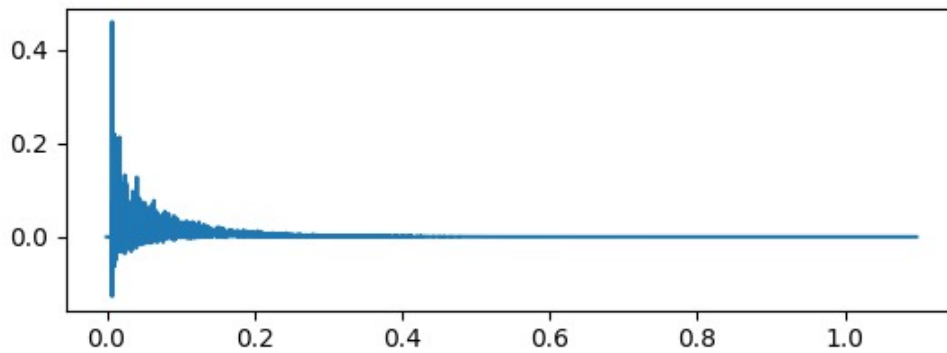


Figure 6

By convolving the audio samples from the sources with the corresponding RIRs, microphone signals are generated, simulating how sound is captured within the virtual environment. Finally, we transfer the resulting signal into a high pass filter.

The HPF (High-Pass Filter) ensures that only relevant higher-frequency components, typically associated with the audio sources and room reflections, are preserved, and further processed, contributing to improved audio quality and intelligibility in the simulated environment.

3.3 Overlap

Dataset creation process:

We defined the fixed length of the samples to be 10 seconds. percentage of overlap is defined by the overlap length divided by the sample length. We choose randomly whether the signal will be from samples of 2 or 3 speakers.

For 2 speakers: we select randomly the overlap [50%, 75%, 100%] and pick two recordings such that their length is greater or equal to this overlap length, if we pick a recording longer than 10 seconds, we will cut it to 10 seconds as the size of the sample. Then we randomly choose the location where the recordings will overlap [sub, beg, end] and place the signals accordingly.

For beg we will overlap the signals randomly at the beginning of the first signal. For end at the end. For sub we will cut the second signal according to the overlap (For example, the overlap in 75% so we will make sure that the second signal will be exactly 7.5 second) and then place the second signal randomly in the range of the first signal.

For 3 speakers: Start as "For 2 speakers", then we select randomly the second overlap [40%, 50%, 60%] where the upper limit is the overlap from the previous step. After that we select randomly a third signals by the rules and with whom to overlap the third signal, whether with the first one or with the second one. Finally, as we did for the previous step, then we randomly choose the location where the recordings will overlap [sub, beg, end] and place the signals accordingly.

For 2 speakers we will define the last channel as zero signal.

For example:

We sampled 2 speakers with an overlap of 50% with lengths of 7.5 and 6 at the end.

First signal with length 6



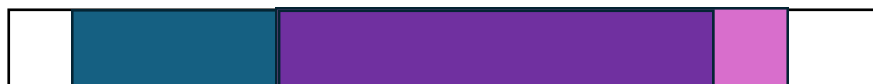
Second signal with length 7.5



The overlap signal:

After drawing an overlap position, we will also draw the starting position of the first signal on the 10 second sample and build the second signal accordingly

The purple part is the overlapping part of 5 seconds

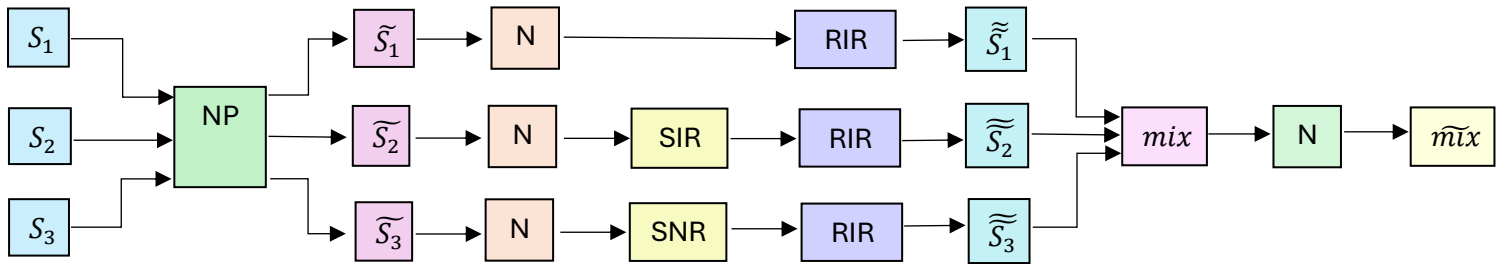


3.4 Pipeline for building the data

For the case of 2 signals S_3 will be np.zeros

N = normalization

NP = New Positions, Depending on the overlaps returns the new positions of each of the signals.



Alpha defined as following $\alpha = \frac{1}{\max(\text{abs}(\text{mix}))}$

We will save each of the signals $\hat{S}_1, \hat{S}_2, \hat{S}_3$ separately and the $\tilde{\text{mix}}$.

$$\hat{S}_1 = \alpha \cdot \tilde{\tilde{S}}_1$$

$$\hat{S}_2 = \alpha \cdot \tilde{\tilde{S}}_2$$

$$\hat{S}_3 = \alpha \cdot \tilde{\tilde{S}}_3$$

The proses of creating the data in our code

1. Reading YAML Configuration:

- The script reads configuration settings from a YAML file `data_config.yaml`. In this file we save pre-set parameters.

2. Looping for Simulation:

- Iterate through a loop to create multiple dataset samples. In each iteration we create one mixed signal that has gone through the pipeline $\tilde{\text{mix}}$.

3. Choose speakers:

- Choosing two or three signals from the LibriSpeech dataset according to the overlap as explained in 3.3.
- For two signals, the third signal will be `np.zeros`.
- Make sure to choose 2 or 3 signals from different speakers.

4. Place the signals:

- Choose randomly the type of overlap (sub, end or beg).
- Place each of the signal (according to the limitations explained in 3.3) in a 10 second array.

5. Normalize the signals.

6. Noise:

- Randomly choose 10 second noise from the LibriSpeech dataset.
- Normalize the noise.

7. Save the signals and noise before room simulation for training.

8. Signal Processing for Separation:

- Calculate and adjust Signal-to-Interference Ratio (SIR) between the primary signal (the first signal) and the second signal.
- If there are three speakers, calculate SIR for the third signal.

9. Handling noise:

- Evaluate the standard deviation (std) of each signal and determine the signal with the lowest standard deviation.
- Adjust and calculate Signal-to-Noise Ratio (SNR) for the selected signal (signal with the lowest std) against background noise.

10. Room Impulse Response (RIR) Simulation:

- Randomizes the room dimensions, speaker placement and microphone placement (according to limitation on the distance between the speakers, between the speakers and the microphone and the distance between the microphone and the speakers from the walls and ceiling).
- Generate signals after RIR as explained in 3.2.
- Transferring the signals in a HPF.

11. Mix the signals.
12. Normalize the mix signal.
13. Calculate $\widehat{S}_1, \widehat{S}_2, \widehat{S}_3$ as explained above.
14. Save the mix, $\widehat{S}_1, \widehat{S}_2, \widehat{S}_3$ and the noise after SNR.
15. Logging Parameters:
 - Compile simulation parameters into a dictionary and append them to a CSV file for record-keeping and analysis.

We created 3 different CSV files for training, validation and testing. In each of the files we saved the following data:

Field	Description
dataset_number	The specific scenario or dataset number used in the current test or simulation.
num_of_speakers	The total number of speakers involved in the simulation.
paths[0], paths[1], paths[2]	File paths for the audio data of each speaker. O+ ne path for each speaker.
h	The height of the room, affecting acoustics.
l	The length of the room, impacting sound propagation and reverberation.
w	The width of the room, influencing room acoustics.
mic_loc	The microphone's location in the room, determining how sound is captured.
speakers_loc[0], speakers_loc[1], speakers_loc[2]	The locations of each speaker in the room, indicating sound source placement.
first_overlap	The percentage of overlap between the first and second speaker's speech.
first_overlap_type	The type of overlap (sub, beg, end) between the first and second speaker.
second_overlap	The percentage of overlap between the second and third speaker's speech.
second_overlap_type	The type of overlap (sub, beg, end) between the second and third speaker.

s1_start, s1_end	Start and end times for the first speaker's speech.
s2_start, s2_end	Start and end times for the second speaker's speech.
s3_start, s3_end	Start and end times for the third speaker's speech.
rt60	The reverberation time (RT60), indicating sound decay time in the room.
sir1, sir2	Signal-to-interference ratios for the first and second speakers, measuring clarity amid interference.
snr	The signal-to-noise ratio, representing the clarity of speech amidst noise.
noise_path	The file path where noise data is stored, if noise is added to the simulation.
pickle_path	The file path where the simulation's configuration or results are saved in pickle format.

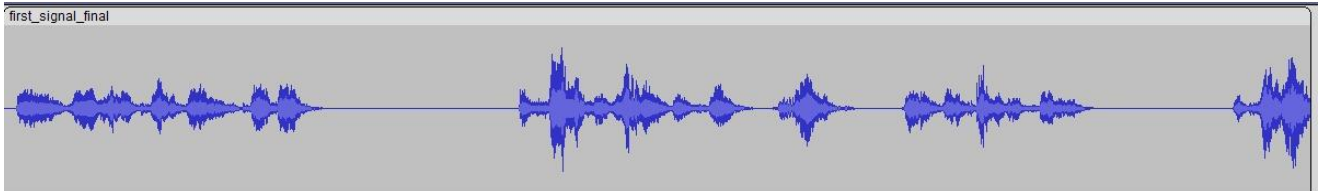
Example for some of the columns in the CSV

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U
1	scenario	num of sp	first_path	second_p	third_pat	room heig	room leng	room widt	mic loc	first loc	second lo	third loc	overlap1	overlap1 t	overlap2	overlap2 t	first start	first end	second st	second er	third
2	1	2	/dsi/gann	/dsi/gann	0	3	4.82	4.83	[3.4822667	[2.1081126	[2.5002485	[2.1081126	0.75	sub	0	beg	0	159744	1160	120968	
3	2	2	/dsi/gann	/dsi/gann	0	2.91	6.29	5.12	[4.0091326	[3.3038507	[3.7100091	[3.3038507	0.75	beg	0	sub	0	159744	0	119808	8
4	3	2	/dsi/gann	/dsi/gann	0	2.53	4.59	4.86	[3.3376513	[3.6880856	[3.8269813	[3.6880856	0.5	beg	0	end	59195	157115	0	139067	13
5	4	2	/dsi/gann	/dsi/gann	0	2.83	5.01	5.78	[1.0919765	[1.9007686	[1.9960805	[1.9007686	0.75	end	0	sub	0	159744	39936	159744	11
6	5	2	/dsi/gann	/dsi/gann	0	2.57	6.16	5.6	[4.0453555	[4.5380522	[5.5567572	[4.5380522	1	beg	0	sub	0	159744	0	159744	11
7	6	2	/dsi/gann	/dsi/gann	0	2.73	6.19	4.86	[3.6335213	[3.8493247	[2.9188911	[3.8493247	0.5	beg	0	sub	84	158324	0	79956	7
8	7	2	/dsi/gann	/dsi/gann	0	2.9	5.24	6.26	[2.2838942	[1.7634745	[2.2142633	[1.7634745	1	beg	0	beg	0	159744	0	159744	
9	8	2	/dsi/gann	/dsi/gann	0	2.57	6.3	5.55	[1.2142813	[1.8606931	[1.1785266	[1.8606931	0.75	beg	0	end	216	147496	0	120024	14
10	9	2	/dsi/gann	/dsi/gann	0	2.6	6.31	4.69	[5.6648445	[5.4233455	[5.5184065	[5.4233455	1	sub	0	sub	0	159744	0	159744	10
11	10	2	/dsi/gann	/dsi/gann	0	2.93	5.85	4.99	[2.6287564	[1.9947782	[1.3587263	[1.9947782	0.5	sub	0	end	0	159744	69247	149119	15
12	11	2	/dsi/gann	/dsi/gann	0	2.72	5.43	4.65	[2.6800085	[4.2009163	[3.5273778	[4.2009163	0.5	sub	0	beg	0	159744	78016	157888	
13	12	2	/dsi/gann	/dsi/gann	0	2.72	5.6	5.5	[1.5740043	[0.9994595	[1.5785112	[0.9994595	0.75	sub	0	sub	0	159744	19897	139705	8
14	13	2	/dsi/gann	/dsi/gann	0	2.83	5.96	5.9	[3.3225734	[2.1448464	[2.2216055	[2.1448464	0.75	end	0	end	0	159744	39936	159744	15
15	14	2	/dsi/gann	/dsi/gann	0	2.54	4.65	6.23	[3.1178514	[3.6704343	[2.1925481	[3.6704343	0.75	end	0	sub	0	159744	39936	159744	4
16	15	2	/dsi/gann	/dsi/gann	0	2.67	5.05	5.4	[2.0021036	[2.3699955	[2.9444684	[2.3699955	1	sub	0	beg	0	159744	0	159744	
17	16	2	/dsi/gann	/dsi/gann	0	2.85	5.55	4.81	[4.0824321	[4.3835533	[4.5982521	[4.3835533	0.5	beg	0	end	24042	151482	0	103914	10
18	17	2	/dsi/gann	/dsi/gann	0	2.98	4.97	5.79	[4.1896166	[3.6762666	[4.0345065	[3.6762666	0.75	sub	0	sub	0	159744	9312	129120	7
19	18	2	/dsi/gann	/dsi/gann	0	2.51	4.7	4.74	[2.9834223	[3.5364107	[2.3883623	[3.5364107	0.5	sub	0	beg	22005	120725	40006	119878	
20	19	2	/dsi/gann	/dsi/gann	0	2.65	5.98	5.41	[5.0583117	[3.6705074	[4.9006095	[3.6705074	0.5	end	0	beg	45396	148356	68484	159744	
21	20	2	/dsi/gann	/dsi/gann	0	2.55	6.11	6.38	[4.3543793	[4.0107803	[4.8602143	[4.0107803	0.75	end	0	sub	0	159744	39936	159744	10
22	21	2	/dsi/gann	/dsi/gann	0	2.78	5.47	4.59	[2.4105523	[3.2300415	[1.9459525	[3.2300415	0.5	sub	0	sub	12844	117884	24846	104718	1
23	22	2	/dsi/gann	/dsi/gann	0	2.76	5.35	4.8	[0.5124248	[1.3075446	[2.0172182	[1.3075446	0.75	sub	0	beg	0	159744	20209	140017	
24	23	2	/dsi/gann	/dsi/gann	0	2.73	5.22	5.48	[3.2316034	[2.7590734	[3.3751633	[2.7590734	1	end	0	beg	0	159744	0	159744	
25	24	2	/dsi/gann	/dsi/gann	0	2.78	4.97	5.93	[1.9470093	[2.3967916	[2.2916894	[2.3967916	0.5	beg	0	sub	60443	150763	0	140315	8

Figure 7

Example for three speakers:

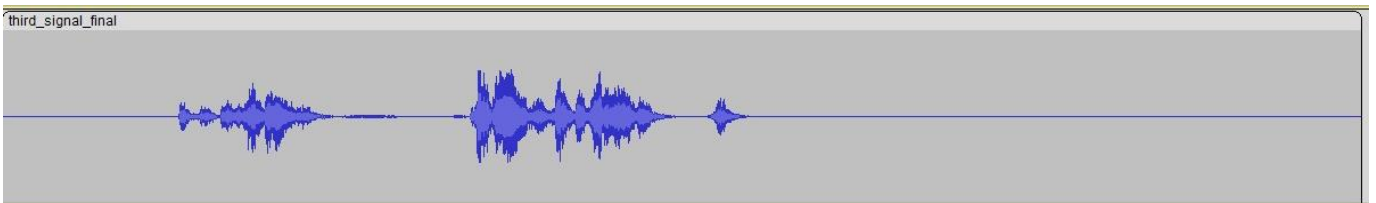
First signal:



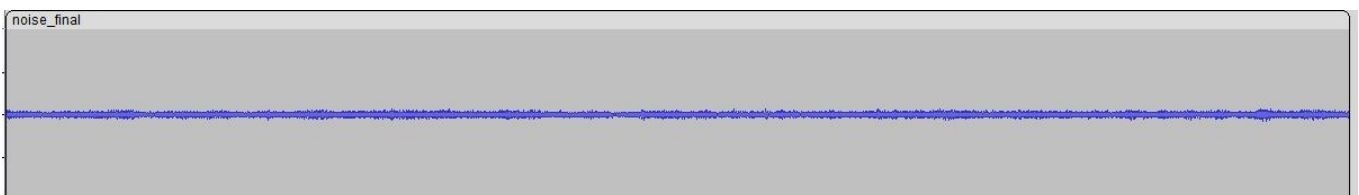
Second signal:



Third signal:



Noise:



Mixed signal:

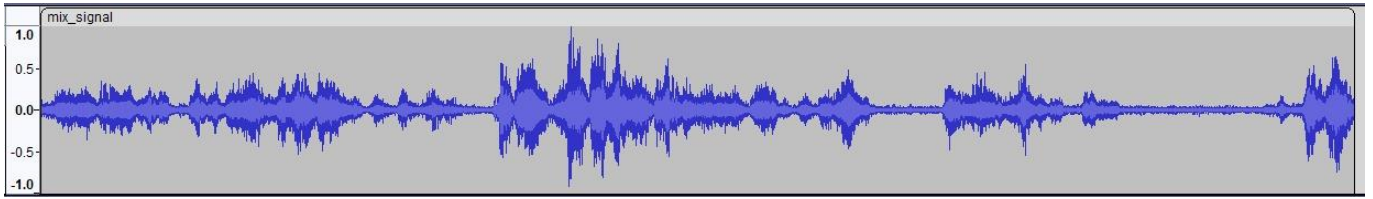


Figure 8

4 Mask

The implementation of masks is crucial for the speech separation process. We use several types of masks, including the Wiener Filter-like Mask (WFM) and the Ideal Binary Mask (IBM), to separate individual speaker signals from a mixed audio input. Below is a detailed explanation of these masks and their role in our project.

4.1. Wiener Filter-like Mask (WFM) Implementation

The Wiener Filter-like Mask (WFM) estimates the contribution of each speaker to the mixed signal. Here's how the WFM is implemented in our project:

- **Magnitude Spectrogram Calculation:** First, the time-domain signals (the mixture and individual sources) are converted into the frequency domain using the Short-Time Fourier Transform (STFT). The magnitude spectrograms of the mixture and the individual sources are then obtained by taking the absolute value of their STFT representations:

$$\begin{aligned}
 S_1(f, t) &= \alpha \cdot \tilde{S}_1 & S_2(f, t) &= \alpha \cdot \tilde{S}_2 & S_3(f, t) &= \alpha \cdot \tilde{S}_3 \\
 mix_mag &= |X(f, t)| \\
 first_mag &= |S_1(f, t)|, & second_mag &= |S_2(f, t)|, & third_mag &= |S_3(f, t)|
 \end{aligned}$$

- **Mask Calculation:** The WFM is computed by squaring the magnitude spectrograms of the individual sources and dividing each by the sum of all squared magnitudes:

$$sum_mag_squared = first_mag^2 + second_mag^2 + third_mag^2$$

$$WFM_{i,f,t} = \frac{|S_i(f, t)|^2}{sum_mag_squared}$$

Here, $WFM_{i,f,t}$ represents the mask for the i -th speaker at frequency f and time t . The mask values range between 0 and 1, indicating the proportion of energy in each T-F bin that belongs to a particular speaker.

We will use the anechoic signal to create the masks so that the model will be able to handle both the separation of signals and dereverberation.

Reshaping for Output: After computing the WFM for each speaker, the mask is reshaped to match the expected input format for the next stages of your processing pipeline. The masks are reshaped from a 3D array with dimensions corresponding to frequency, time, and the number of speakers (T, F, nspk) into a 2D array (T*F, nspk), where each column represents a different speaker. T = 625, F = 257, nspk = 2/3.

4.2. Ideal Binary Mask (IBM) Implementation

The Ideal Binary Mask (IBM) assigns each T-F bin to the speaker with the highest energy. Here's how the IBM is implemented:

- **Magnitude Comparison:** For each T-F bin, the magnitudes of the sources are compared to determine which speaker has the highest energy. The IBM for each speaker is then generated based on these comparisons:

$$IBM_{1,f,t} = \begin{cases} 1 & \text{if } |S_1(f,t)| > |S_2(f,t)| \text{ and } |S_1(f,t)| > |S_3(f,t)| \\ 0 & \text{else} \end{cases}$$

$$IBM_{2,f,t} = \begin{cases} 1 & \text{if } |S_2(f,t)| > |S_1(f,t)| \text{ and } |S_2(f,t)| > |S_3(f,t)| \\ 0 & \text{else} \end{cases}$$

$$IBM_{3,f,t} = \begin{cases} 1 & \text{if } |S_3(f,t)| > |S_1(f,t)| \text{ and } |S_3(f,t)| > |S_2(f,t)| \\ 0 & \text{else} \end{cases}$$

- **Reshaping for Output:** Similar to the WFM, the IBM is reshaped from a 3D array to a 2D format suitable for further processing. The

resulting binary masks are then used to extract the spectrograms of the individual speakers by multiplying them element-wise with the mixture spectrogram.

4.3. Weight Mask Implementation

In addition to WFM and IBM, our project includes a weight mask designed to focus on the most significant T-F bins, those with the highest energy.

This mask is computed as follows:

- **Threshold Calculation:** A percentile-based threshold is calculated to determine the top percentage of T-F bins based on their energy.

This threshold is used to create a binary weight mask:

$$weight_mask_{f,t} = \begin{cases} 1 & \text{if } mix_mag > threshold \\ 0 & \text{else} \end{cases}$$

* *threshold = 10* – This will give us the top 90% of bins

- **Reshaping for Output:** The weight mask is reshaped similarly to the WFM and IBM, providing a focus on the T-F bins that contribute most significantly to the mixed signal.

Application

The masks serve multiple purposes:

- **Source Separation:** The masks are applied to the magnitude spectrogram of the mixture to isolate the individual sources. For example, the WFM allows for a more nuanced separation by estimating the contribution of each speaker to each T-F bin, while the IBM provides a more binary separation.
- **Training the Model:** The masks are also used during the training of our neural network model. For instance, the WFM and IBM are employed as targets for the model to learn how to separate the

mixed signal into its component sources. By minimizing the difference between the estimated masks and the true masks, our model gradually improves its ability to separate speakers.

- **Improving Robustness:** The weight mask, in particular, helps improve the robustness of the separation process by focusing on the most significant T-F bins, reducing the impact of noise and less relevant components of the mixture.

An example of real and estimated WF mask for speaker 1+2:

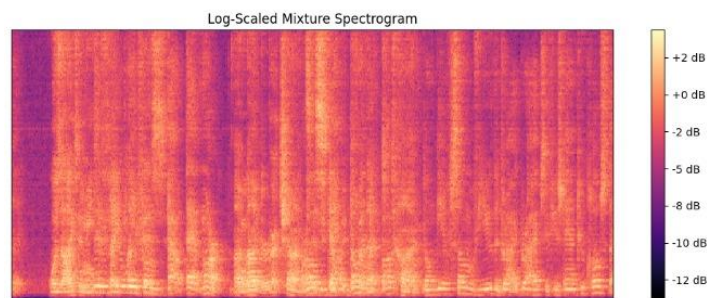


Figure 9

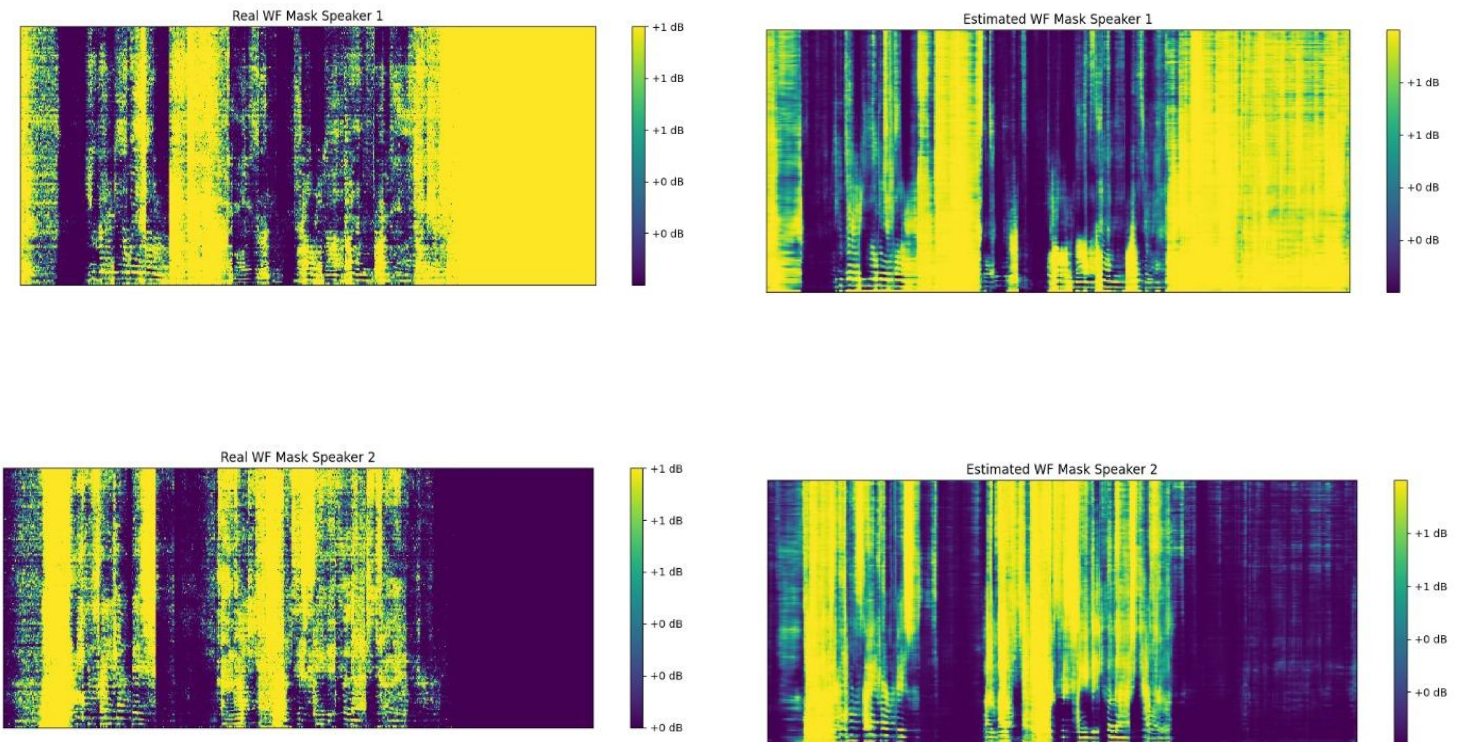


Figure 10

5 Loss function

The loss function is designed to optimize the separation of sources in a mixed audio signal by minimizing the error between the estimated masks and the ideal masks. The neural network's performance hinges on how well these masks represent the sources in the mixture. Let's break down the implementation:

5.1. Objective of the Loss Function

The primary objective of the loss function is to minimize the difference between the estimated magnitude spectrograms of the sources and their actual (ideal) magnitude spectrograms. The loss function achieves this by comparing the masks generated by the neural network with the ideal masks for each time-frequency bin.

5.2. Reconstruction Error

The loss function is based on the standard L_2 reconstruction error. This is a common approach in source separation tasks, where the loss quantifies how close the estimated magnitude spectrogram \hat{s}_i for each source is to the actual magnitude spectrogram s_i .

Mathematically, the L_2 reconstruction error is given by:

$$l = \frac{1}{C} \sum_{i=1}^C \|x \odot (m_i - \hat{m}_i)\|^2$$

where:

- C is the number of sources (speakers).
- x is the magnitude spectrogram of the mixed signal.
- m_i is the ideal mask for source i .
- \hat{m}_i is the mask estimated by the neural network for source i .
- \odot denotes element-wise multiplication.

5.3. Mask Optimization

The loss function drives the optimization of the estimated masks \widehat{m}_i to closely resemble the ideal masks m_i . The neural network learns to adjust its parameters so that the difference $m_i - \widehat{m}_i$ is minimized across all time-frequency bins, thereby improving the accuracy of the source separation.

5.4. Permutation Invariance

One of the challenges in source separation is the permutation problem, where the order of separated sources in the output can vary. In our case, however, we don't need to handle this issue because our network is explicitly provided with the masks in a predefined order. This ensures that the outputs are consistently aligned with the selected order of the masks, avoiding the need for permutation-invariant training.

5.5. Dynamic Speaker Handling

The number of speakers (or sources) in the mixture is not fixed, and the neural network needs to adapt to different numbers of sources. The loss function, combined with the network architecture, allows for dynamic handling of varying numbers of sources by adjusting the number of masks (or attractors) during training and inference.

6 Deep Attractor Network

The Deep Attractor Network (DANet) is a neural network architecture primarily designed for speech separation tasks, where the goal is to separate multiple speakers from a mixed audio signal. This model leverages the concept of attractor points in the embedding space, which helps in clustering similar features together. The model assigns each time-frequency (TF) bin of the mixed spectrogram to one of the sources, typically by applying masks.

6.1 Components of the DANet Model

1. Recurrent Neural Network (RNN) Layer:

- **Purpose:** The RNN layer is designed to capture temporal dependencies within the input features. This is particularly important for speech processing, as speech signals have inherent sequential patterns.
- **LSTM (Long Short-Term Memory):** The RNN implemented using LSTM units, which are capable of learning long-term dependencies while avoiding issues like vanishing gradients.
- **Input and Output Dimensions:** The input dimension corresponds to the number of frequency bins (or features) in the spectrogram, and the hidden dimension is set to a value of 300, which represents the size of the hidden state in the LSTM. If bidirectionality is enabled, the output dimension is doubled.

2. Fully Connected Layer (FC Layer):

- **Purpose:** After the RNN processes the input features, the output is passed through a fully connected layer. This layer is responsible for transforming the output of the RNN into embeddings, which represent the input data in a higher-dimensional space.
- **Non-Linearity:** The fully connected layer often uses a non-linear activation function, such as the hyperbolic tangent (tanh), to introduce non-linearity into the model, allowing it to learn complex patterns.

3. Embedding Space and Clustering:

- **Embedding Generation:** The embeddings produced by the FC layer represent the input features in a higher-dimensional space. Each time-frequency bin in the input spectrogram is mapped to a point in this embedding space.
- **Clustering:** The idea is to group or cluster similar embeddings together, which corresponds to separating different speakers. Attractor points are virtual points in the embedding space that attract embeddings corresponding to the same speaker. These points help in forming distinct clusters for each speaker.
- The **dashed arrow** between the embeddings V and the source assignments Y represents the clustering relationship. The embeddings are mapped to clusters (sources) based on their proximity to attractor points A . This clustering process determines the source assignments and subsequently guides mask generation.
- During testing, the attractor points A are calculated based on clusters identified in the embedding space using

techniques like K-means. These attractor points are then used to assign each time-frequency (T-F) bin to the correct source, enabling mask generation. In contrast, during training, the attractor points are computed directly from the true or estimated source assignments Y , providing a more straightforward and supervised approach to guide the clustering process and optimize the model's performance.

4. Masking Process:

- **Mask Generation:** The output embeddings are used to generate masks that are applied to the mixed spectrogram. Each mask corresponds to a different speaker, and when applied, it isolates the speech of that speaker from the mixture.
- **Application:** The masks are element-wise multiplied with the input spectrogram to reconstruct the separated sources.

6.2 Model Workflow

1. **Input Transformation:** The input spectrogram (with shape $B \times F \times T$) is first transposed to match the expected input shape of the RNN ($B \times T \times F$), where B is the batch size, T is the sequence length (time frames), and F is the number of frequency bins.
2. **Sequential Processing:** The transformed input is processed through the RNN layers, capturing the temporal dependencies across time frames.
3. **Embedding Creation:** The output from the RNN is flattened and passed through the FC layer to create embeddings. These embeddings are then reshaped to the original spectrogram dimensions with an additional dimension for the attractor points.

4. **Mask Estimation:** The embeddings are used to estimate the masks, which are then applied to the input spectrogram to separate the individual sources.

Pipeline for Training: During training, the true source assignments Y (or estimated ones) are known. These assignments are used to calculate the attractor points A . The loss function minimizes the difference between the generated masks M and the ground truth masks to optimize the model.

Pipeline for Testing: In testing, the true source assignments are unavailable. The model uses clustering techniques like K-means or fixed attractor points to infer the attractors A . These attractors are then used to compute source assignments Y , leading to the generation of masks for source separation.

5. **Output:** The model produces embeddings that can be used to cluster similar features together, ultimately allowing for the separation of different sources from a mixed audio signal.

6.3 Key Characteristics

- **Attractor Points:** A key innovation in DANet is the use of attractor points in the embedding space. These points guide the clustering process, making it more effective for source separation.

During training, true or estimated source assignments guide the attractor point calculation, ensuring the embeddings and masks align with the correct sources. During testing, source assignments are inferred through clustering methods, as no ground truth is available. The clustering step introduces an additional layer of computation in testing that is not present during training.

- **Bidirectional LSTM:** Bidirectional LSTMs allow the model to consider both past and future context when processing each time

frame, improving the model's ability to understand the temporal structure of the input.

- **Flexibility:** The architecture can be adapted to different input and output dimensions, depending on the specific requirements of the task.

Overall, the DANet model is well-suited for complex speech separation tasks, where distinguishing multiple overlapping sources is challenging. By leveraging RNNs, fully connected layers, and attractor-based clustering in the embedding space, the model provides a powerful approach to isolating individual sources from a mixed signal.

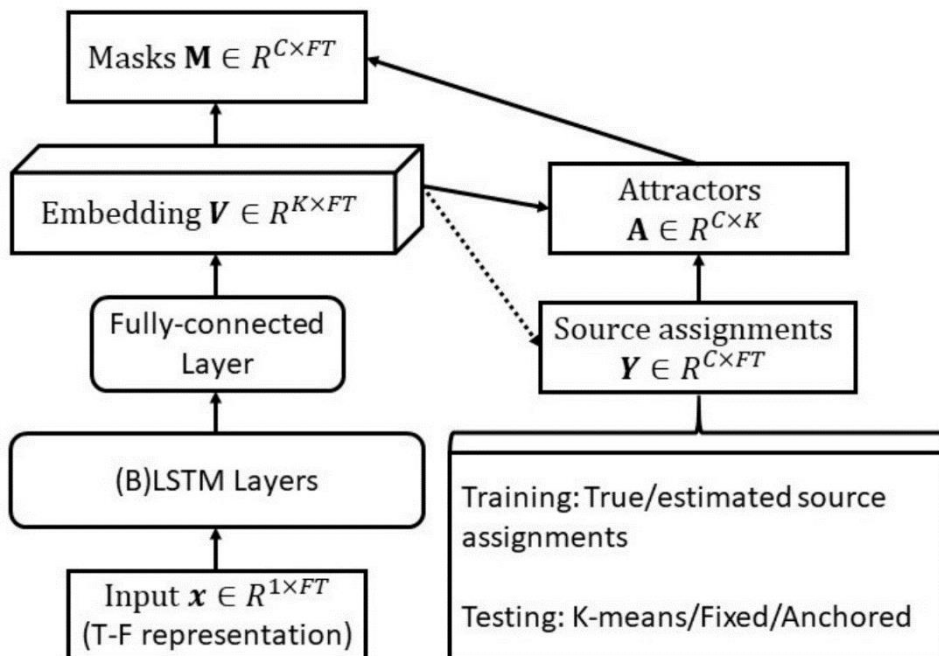


Figure 11

7 The Training Process

The training process for the DANet involves iteratively refining the model to accurately separate mixed audio signals into their constituent sources. This is achieved through a series of epochs where the model learns from examples using a systematic approach.

Initialization

Before training begins, essential hyperparameters are defined, including batch size, learning rate, and the number of training epochs. This setup allows for customization of the training process based on the dataset and the desired outcomes. Additionally, mechanisms are in place to leverage

Figure 11

GPU

acceleration if available, enhancing computational efficiency.

Data Loading

The training and validation datasets are prepared for processing. The training dataset is shuffled to ensure that the model encounters a diverse range of samples in each epoch, which helps prevent overfitting. In contrast, the validation dataset is processed in a fixed order, enabling consistent evaluation of the model's performance.

Model Architecture

The model consists of a recurrent neural network, using LSTM layers, which enables the capturing of temporal dependencies in the audio signals. After the RNN processing, the outputs are transformed through fully connected layers to estimate the separation masks needed for audio source separation.

Training Loop

During each epoch, the model undergoes a series of training iterations. For each iteration, a batch of audio features is fed into the model, which computes the estimated masks for the mixed audio signals. The difference between the estimated masks and the target masks is quantified using a loss function, guiding the model on how well it is performing.

The model's parameters are updated based on the computed gradients, allowing it to learn from its mistakes. Progress is monitored throughout the epoch, providing feedback on the model's performance in terms of loss values.

Validation Process

After completing the training iterations for an epoch, the model's performance is assessed on the validation dataset. This evaluation helps gauge how well the model generalizes to unseen data, which is crucial for ensuring its practical applicability. The validation loss is calculated similarly to the training loss, but without updating the model's parameters.

Learning Rate Management

Following each epoch, the training process checks for improvements in both training and validation losses. If the model demonstrates improved performance on the validation set, the current state of the model is saved. If no improvement is observed over several epochs, the learning rate may be adjusted to facilitate better convergence and potentially enhance the model's learning.

8 The Testing Process

The testing process is a crucial phase in any machine learning project, particularly in audio source separation tasks. It systematically evaluates the performance of the trained model, ensuring that it can generalize well to unseen data. By meticulously processing the mixed signals, extracting meaningful features, and reconstructing separated outputs, this approach provides a comprehensive evaluation of the model's capabilities. The use of K-Means clustering and masking further enhances the separation accuracy, making the approach robust and effective for practical applications.

Setup and Configuration:

The environment is prepared by importing essential libraries and modules, such as PyTorch for model operations and libraries for audio processing. Parameters like batch size, input feature dimensions, and model hyperparameters are defined to configure the testing setup.

Data Preparation:

A dataset of mixed audio signals is created for testing. This involves using a specific data loader to efficiently handle the loading of batches of audio data. By shuffling the data, the model's robustness is tested against various input scenarios.

Model Initialization:

A pre-trained model is loaded from a specified file. This model, trained on similar audio tasks, is set to evaluation mode, which disables certain features like dropout and batch normalization, ensuring consistent behavior during inference.

Audio Signal Processing:

The mixed audio signals are processed to convert them into a suitable format for analysis. This step includes applying Short-Time Fourier Transform (STFT) to transform the time-domain signals into the frequency domain, capturing their spectral features.

Feature Extraction:

The magnitude and phase of the frequency-domain representation are extracted. The magnitude is converted to a decibel scale for better representation, while phase information is retained for reconstructing the audio later. This step ensures that the model receives data in a format that maximizes its performance.

Inference:

The input features are fed into the model to generate embeddings. This process is done in a no-gradient context, meaning the model doesn't learn during testing, which conserves resources. The embeddings represent high-level features that capture the essential characteristics of the audio signals.

Clustering and Masking:

K-Means clustering is applied to the embeddings to identify distinct audio sources within the mixture. Each cluster center corresponds to a potential source. Based on these clusters, masks are created, which are mathematical representations used to filter out specific audio sources from the mixed input.

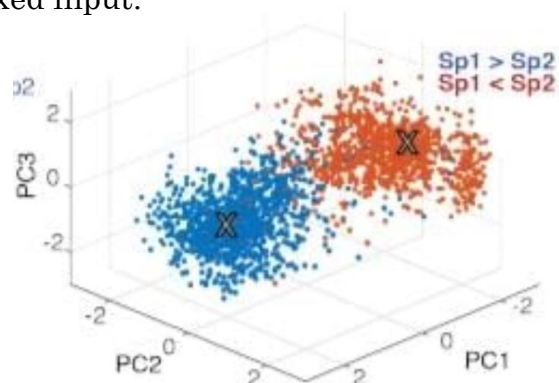


Figure 12

Signal Reconstruction:

After generating the masks, the original mixed spectrogram is modified by applying the masks to the respective components. This involves multiplying the magnitude spectrogram by the mask while retaining the original phase information. The inverse STFT is then applied to convert the modified spectrograms back into time-domain waveforms, resulting in separated audio signals.

Saving Results:

The separated audio signals are organized and saved into a specified directory structure. Each separated signal is written to a separate audio file, which facilitates further analysis or listening tests. This step ensures that outputs are systematically cataloged for easy retrieval.

9 Results

9.1 model 1- 2/3 speakers with noise

For the training and testing process we created 110,000 samples. 64% for training and 18% for validation and testing each. In total:

70,400- for training

19,800- for validation

19,800- for testing

$C = 3$ – for the masks and loss

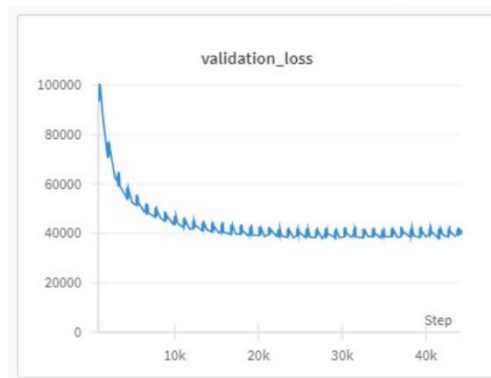


Figure 13

It is possible that one of the reasons for the bad results is that for the masks we used the non-resonated and non-normalized signal instead of the non-reverberated and normalized signal according to the mix. Also, we normalized the signals between a fixed range of values instead of according to the mix. Also, it appears the model encountered challenges in handling variable speaker counts, when switching between two and three speakers. This complexity likely impacted the separation quality, as the model needed to adapt to different speaker scenarios within the same processing framework.

Now we will make changes in the process of training and creating the signals to see how it will affect the results. Also, for all the following models we normalized the signals according to the mix.

9.2 SI-SDR Calculation

The **Scale-Invariant Signal-to-Distortion Ratio (SI-SDR)** is a key metric for evaluating the quality of estimated signals in tasks like speech separation, focusing on how well an estimated signal matches the target while ignoring differences in scale.

SI-SDR Calculation:

- **Zero-mean signals:** Both the estimated (\hat{s}) and original (s) signals are normalized to have zero mean to ensure that the evaluation is scale-invariant. This is important because we want to compare the signals based on their structure, not their amplitude.
- **Signal projection:** The estimated signal is projected onto the target signal. This process extracts the part of the estimated signal that directly aligns with the target, called the target signal component calculated as:

$$s_{target} = \frac{\langle \hat{s}, s \rangle}{\|s\|^2}$$

where $\langle \hat{s}, s \rangle$ represents the inner product between the estimated and target signals, and $\|s\|^2$ is the power of the target signal.

- **Noise estimation:** The difference between the estimated signal and the projected target signal is treated as noise, defined as:

$$e_{noise} = \hat{s} - s_{target}$$

This represents the part of the estimated signal that doesn't align with the target.

- **Energy ratio:** The core of the SI-SDR calculation is the ratio of the energy of the target signal s_{target} to the energy of the noise signal e_{noise} . This ratio is expressed in decibels (dB) using a logarithmic scale:

$$SI - SDR = 10 \log_{10} \left(\frac{\|s_{target}\|^2}{\|e_{noise}\|^2} \right)$$

Higher SI-SDR values indicate better separation, meaning that the estimated signal closely matches the target signal.

9.3 model 2- 2 speakers without noise

For the training and testing process we created 57,000 samples. 88% for training, 8% for validation and 4% for testing. In total:

50,160- for training

4,560- for validation

2,280- for testing

$C = 2$ – for the masks and loss

We will make the following changes in creating the masks:

$$\begin{aligned} \widetilde{S}_1(f, t) &= \alpha \cdot \widetilde{\widetilde{S}}_1 & \widetilde{S}_2(f, t) &= \alpha \cdot \widetilde{\widetilde{S}}_2 & \widetilde{S}_3(f, t) &= \alpha \cdot \widetilde{\widetilde{S}}_3 \\ \text{first_mag_r} &= |\widetilde{S}_1(f, t)|, & \text{second_mag_r} &= |\widetilde{S}_2(f, t)|, & \text{third_mag_r} &= |\widetilde{S}_3(f, t)| \end{aligned}$$

Mask Calculation: The WFM is computed by squaring the magnitude spectrograms of the individual sources and dividing each by the sum of all squared magnitudes:

$$\text{sum_mag_squared} = \text{first_mag_r}^2 + \text{second_mag_r}^2 + \text{third_mag_r}^2$$

$$WFM_{i,f,t} = \frac{|\widetilde{S}_i(f, t)|^2}{\text{sum_mag_squared}}$$

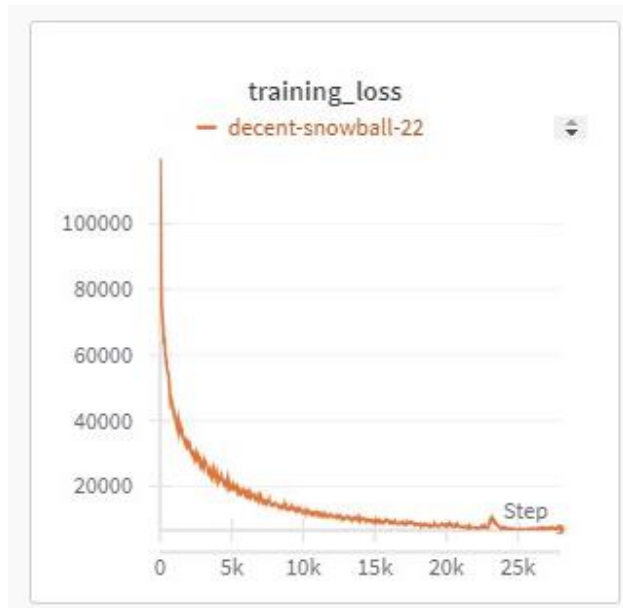


Figure 14

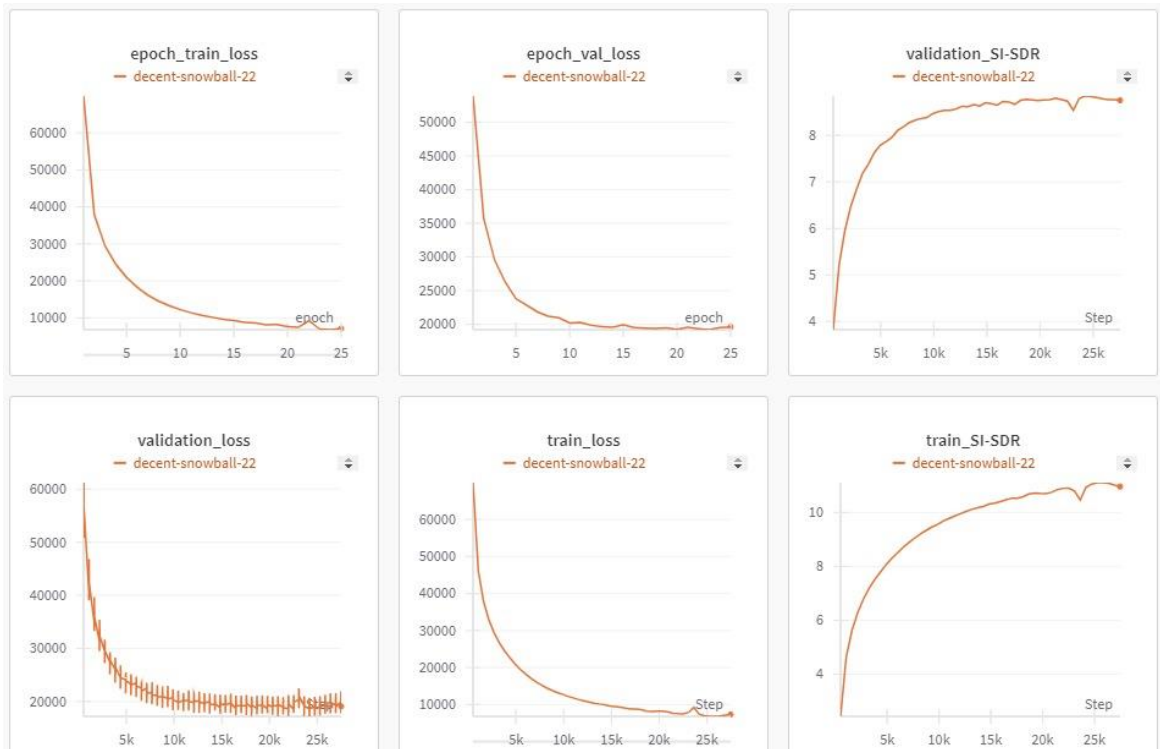
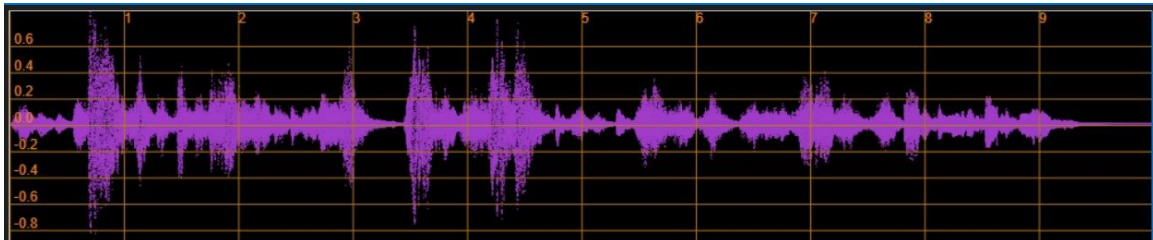


Figure 15

Mix signal:



Mix signal spectrogram:

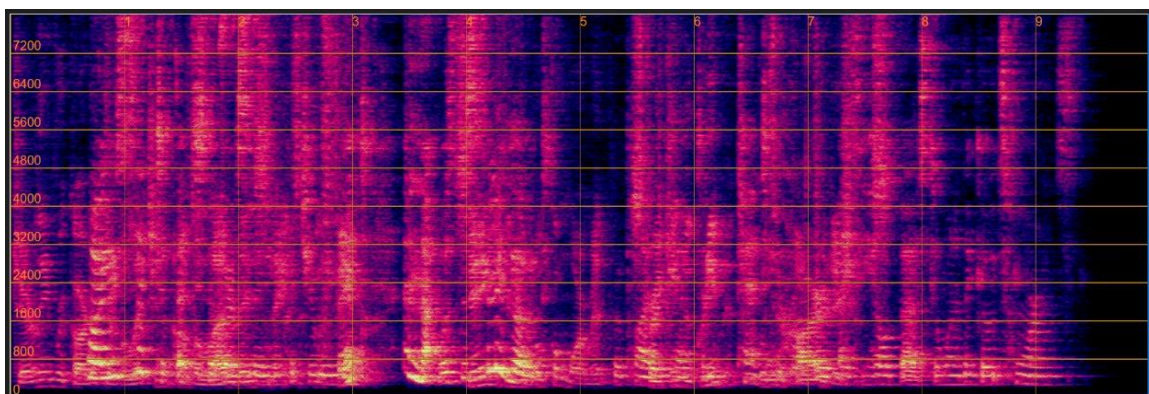
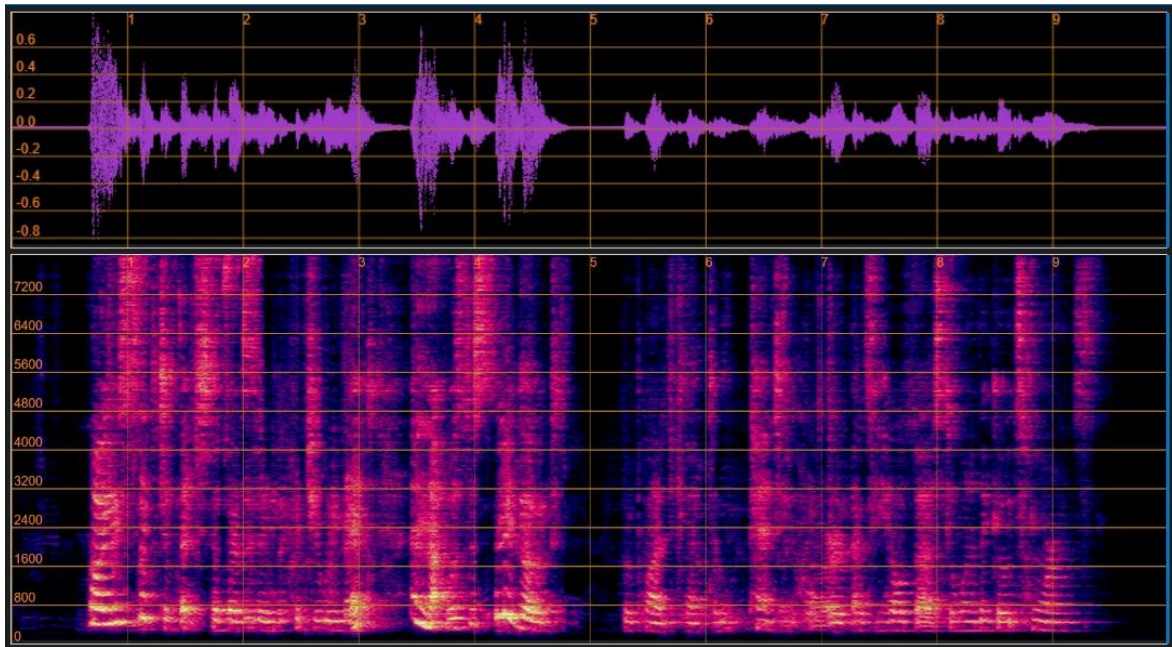


Figure 16

Signal and spectrogram of speaker 1 after the separation:



Signal and spectrogram of speaker 2 after the separation:

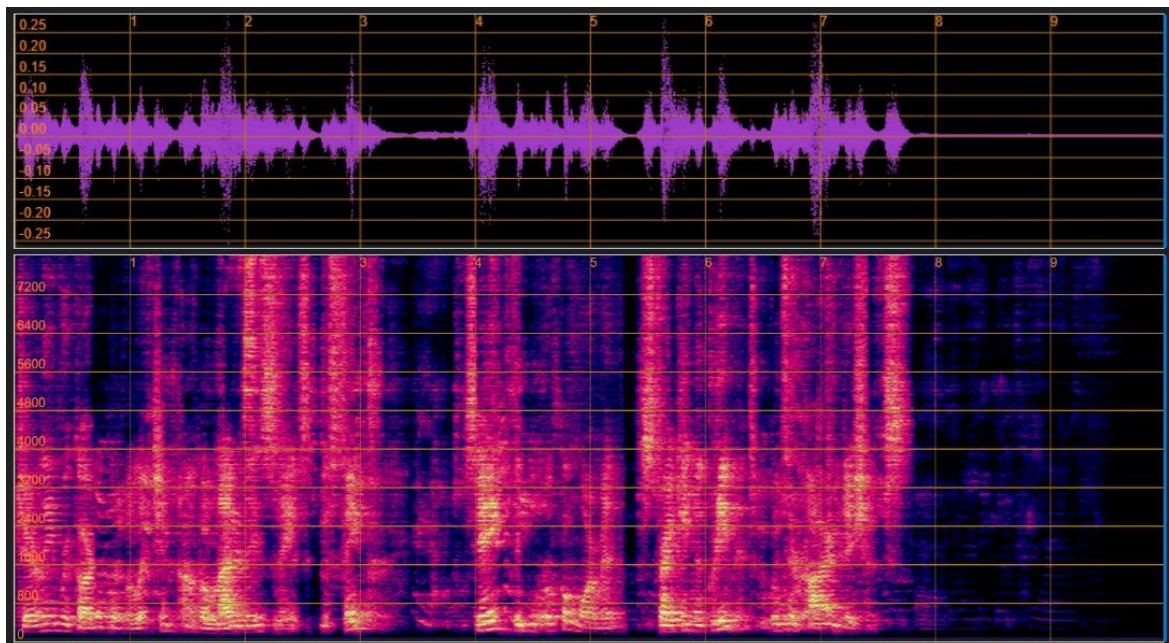


Figure 17

Comparison between the separated signal and the original:

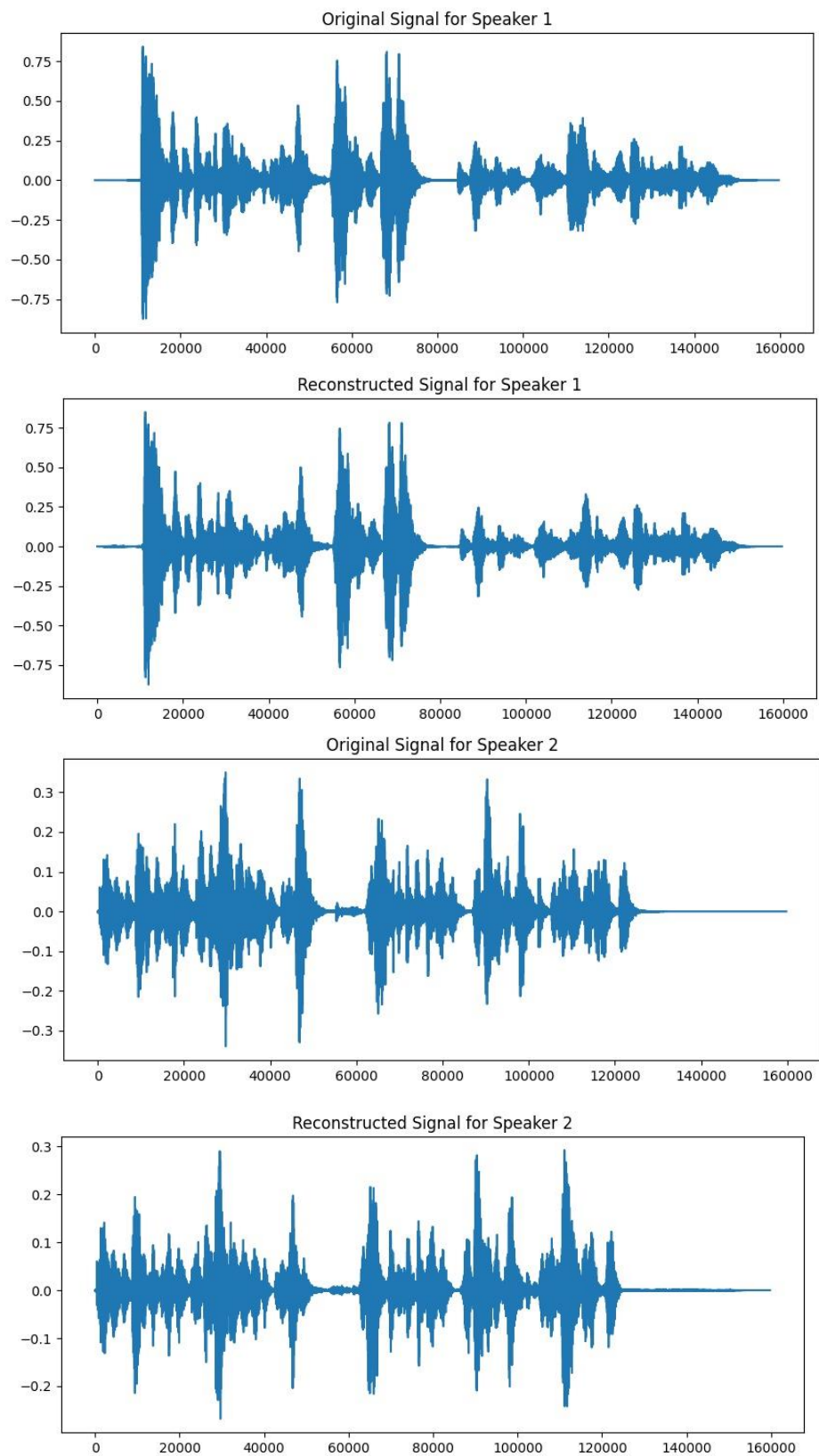


Figure 18

It can be seen that a good reconstruction of the signals is obtained from the mix.

We divided some of the criteria in the csv file into arrays by index. We created a new csv file for each index in the test and calculated the si-sdr. Then for each set of categories we calculated the average si-sdr.

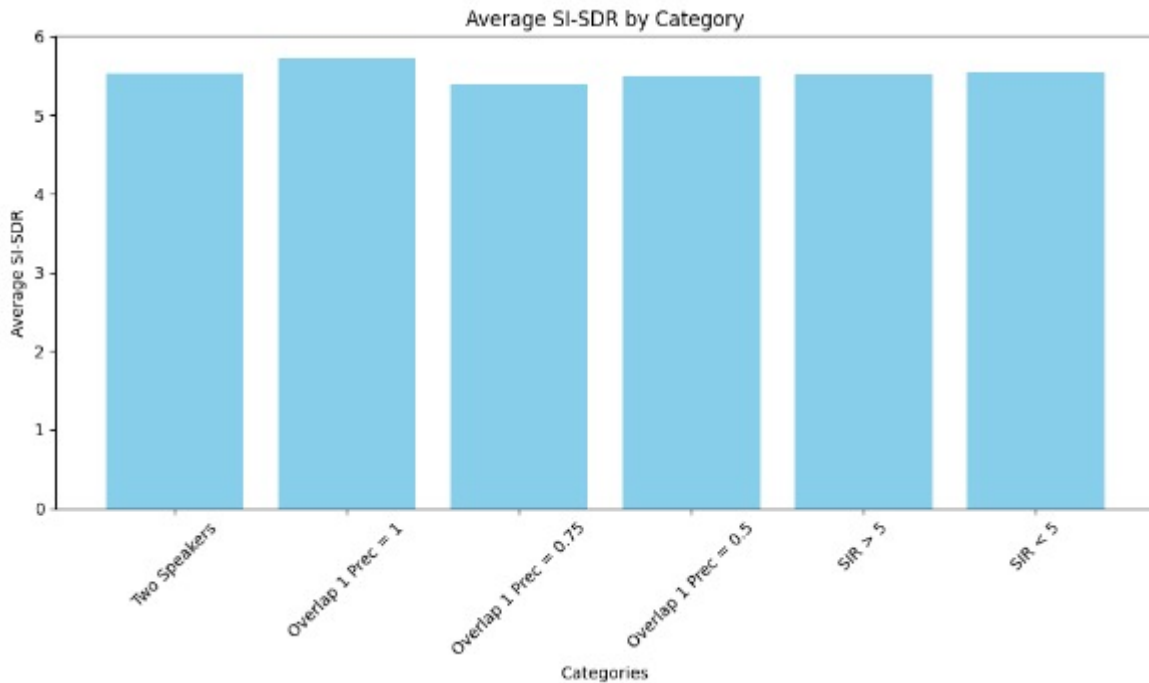


Figure 19

Insights

The model produces excellent results, with the reproduced signals closely resembling the originals and clear separation quality.

Additionally, it appears that the categories have minimal impact on the SI-SDR. This may be due to the random nature of data creation, for instance, a mix with a high overlap percentage but also a high SNR could involve two conflicting categories—one that complicates separation and another that facilitates it. As a result, the SI-SDR values are fairly evenly distributed, aligning with expectations given the data generation method. The differences might become clearer if we take the same signal and add elements such as reverberation, noise, and overlapping segments throughout. We also achieve a high SI-SDR, indicating strong separation performance.

9.4 model 3- 2 speakers with noise

For the training and testing process we created 57,000 samples. 88% for training, 8% for validation and 4% for testing. In total:

50,160- for training

4,560- for validation

2,280- for testing

$C = 2$ – for the masks and loss

Same as in model 2 but with noise (without a mask for the noise)

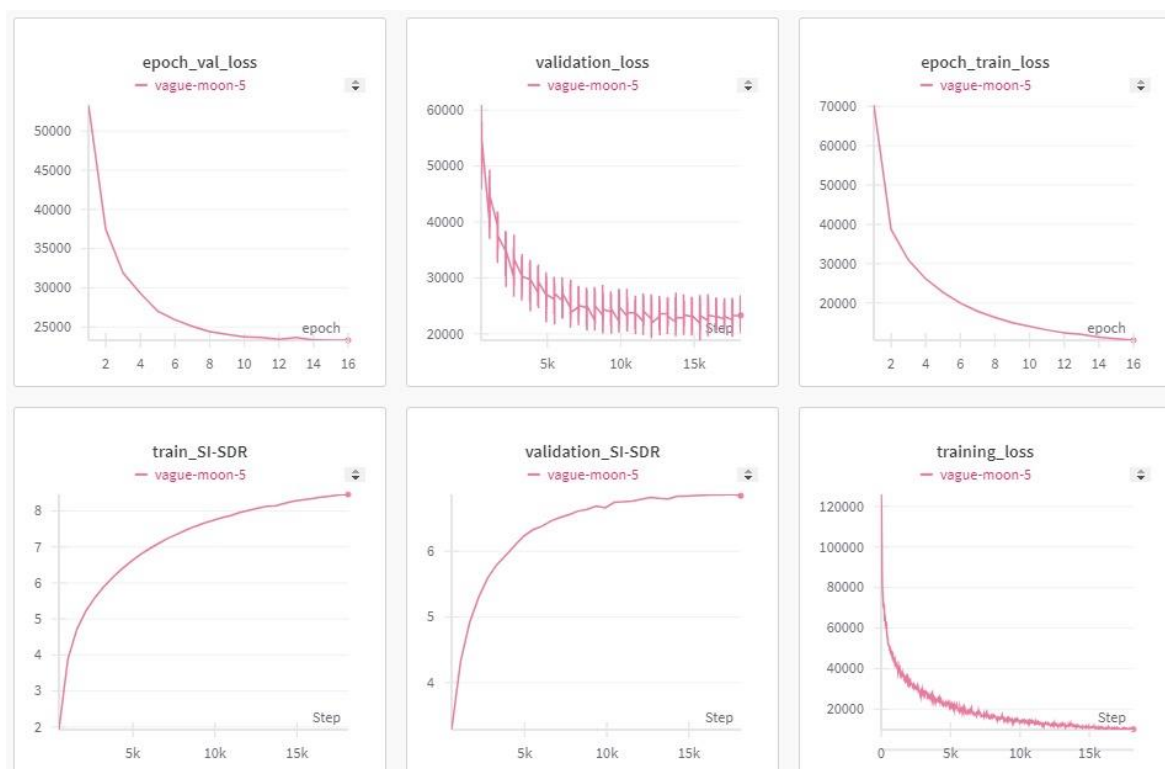
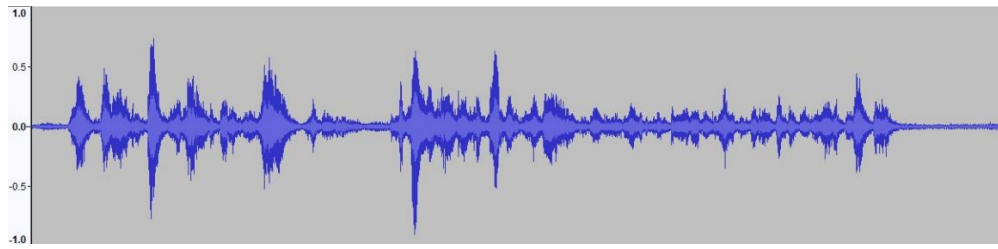


Figure 20

Mix signal:



Mix signal spectrogram:

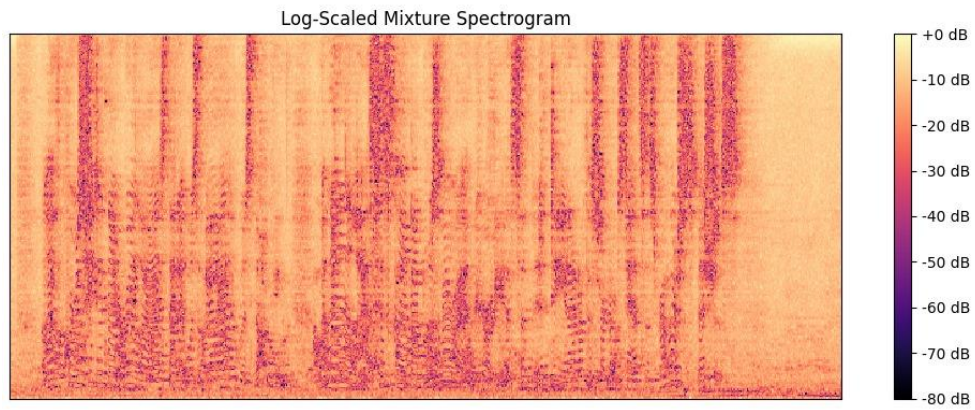


Figure 21

Noise:

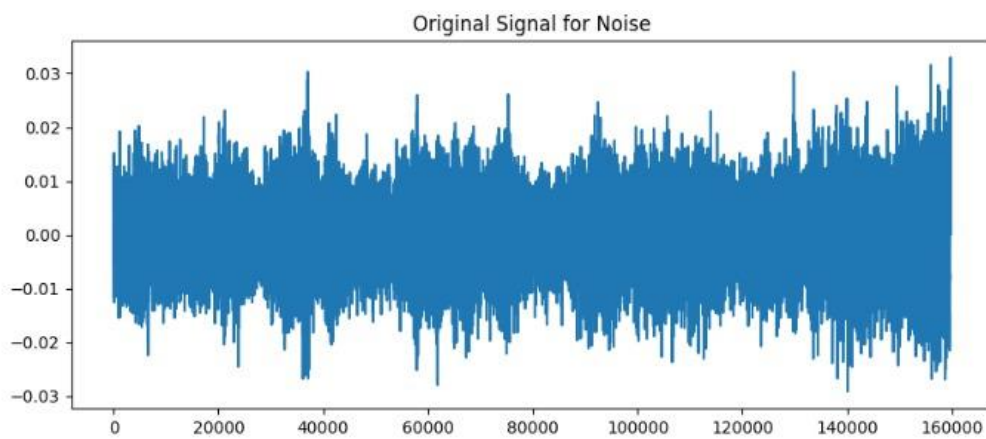
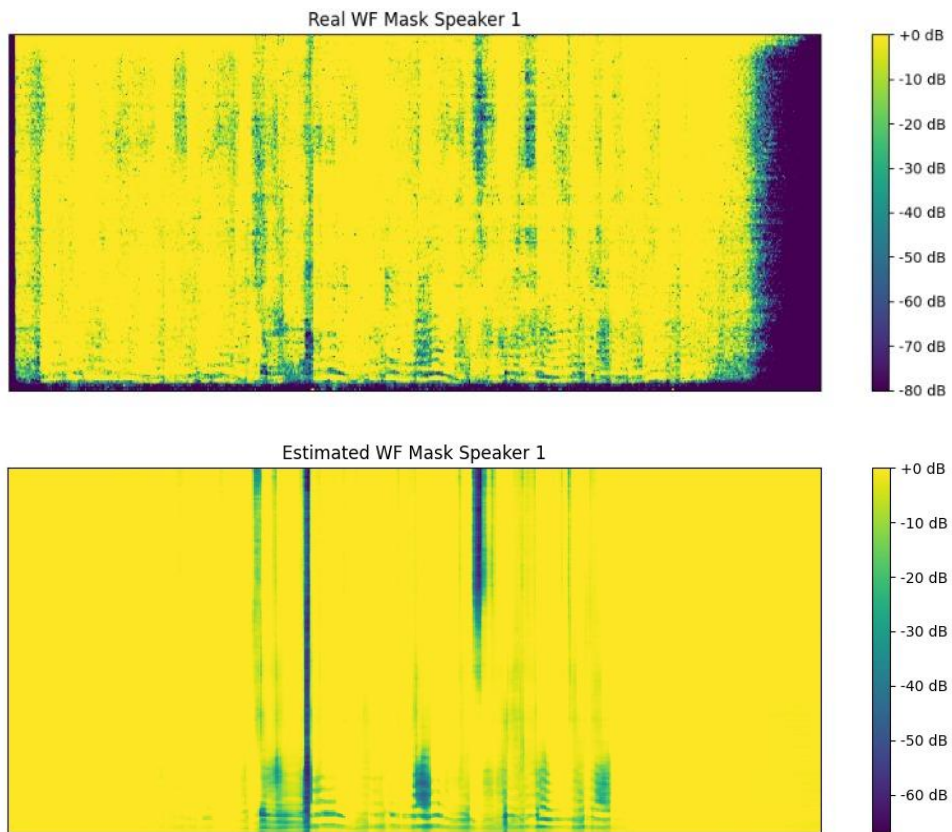


Figure 22

Real and estimated WF mask of speaker 1:



Real and estimated WF mask of speaker 2:

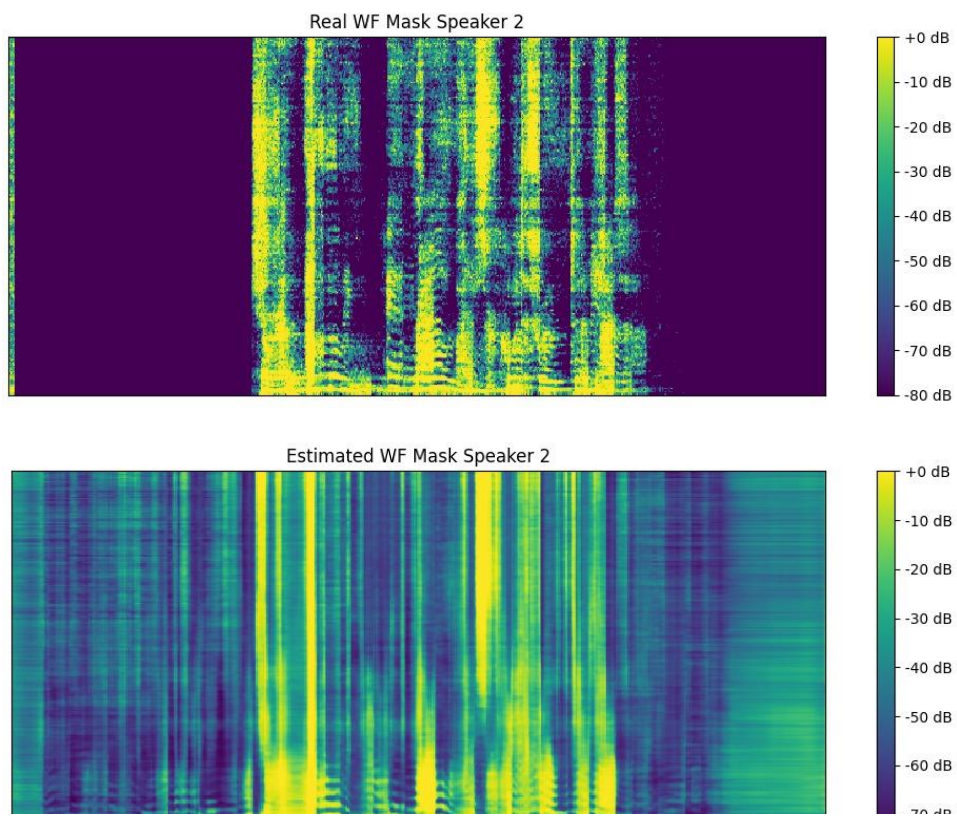


Figure 23

Comparison between the separated signal and the original:

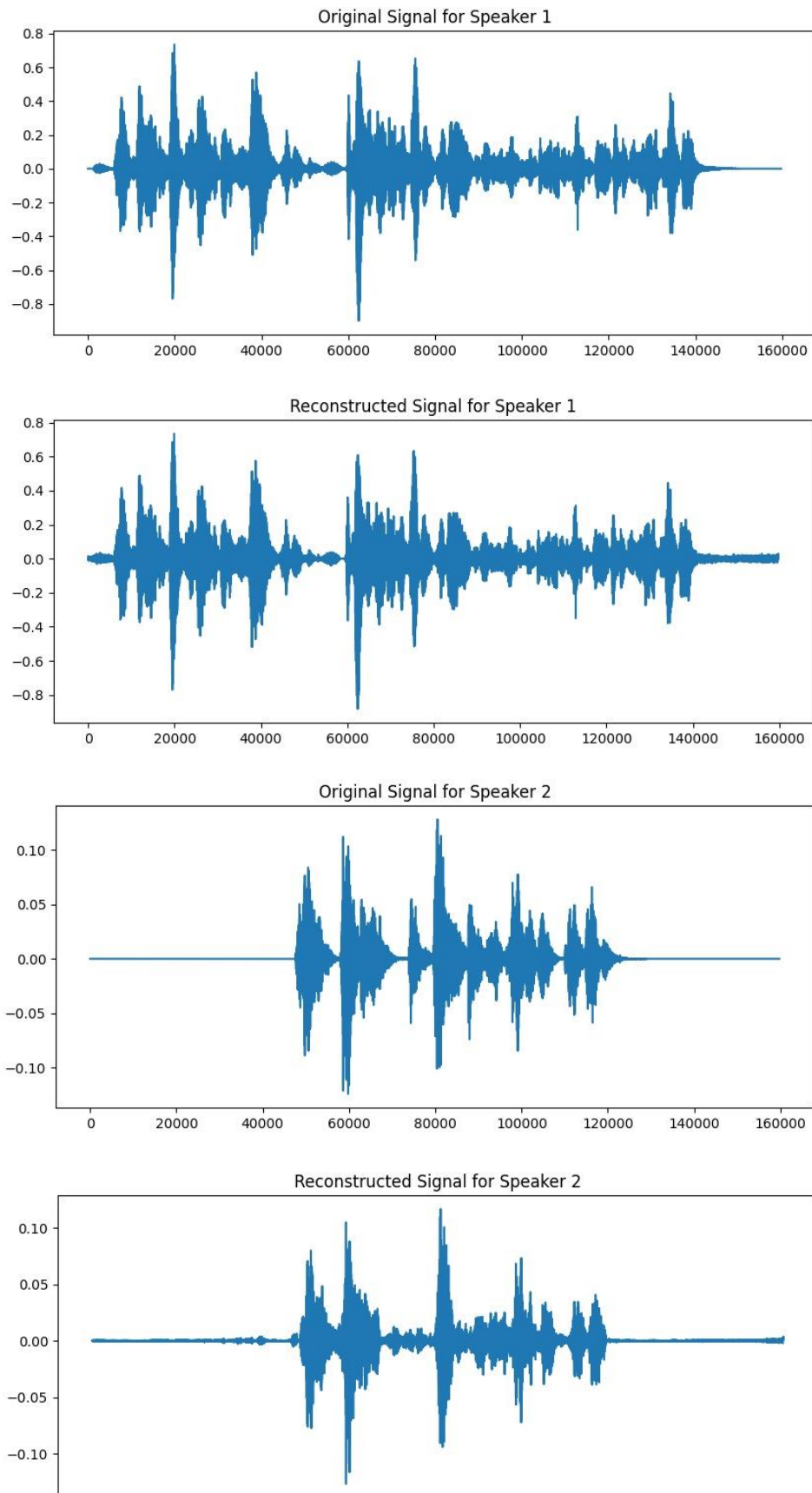


Figure 24

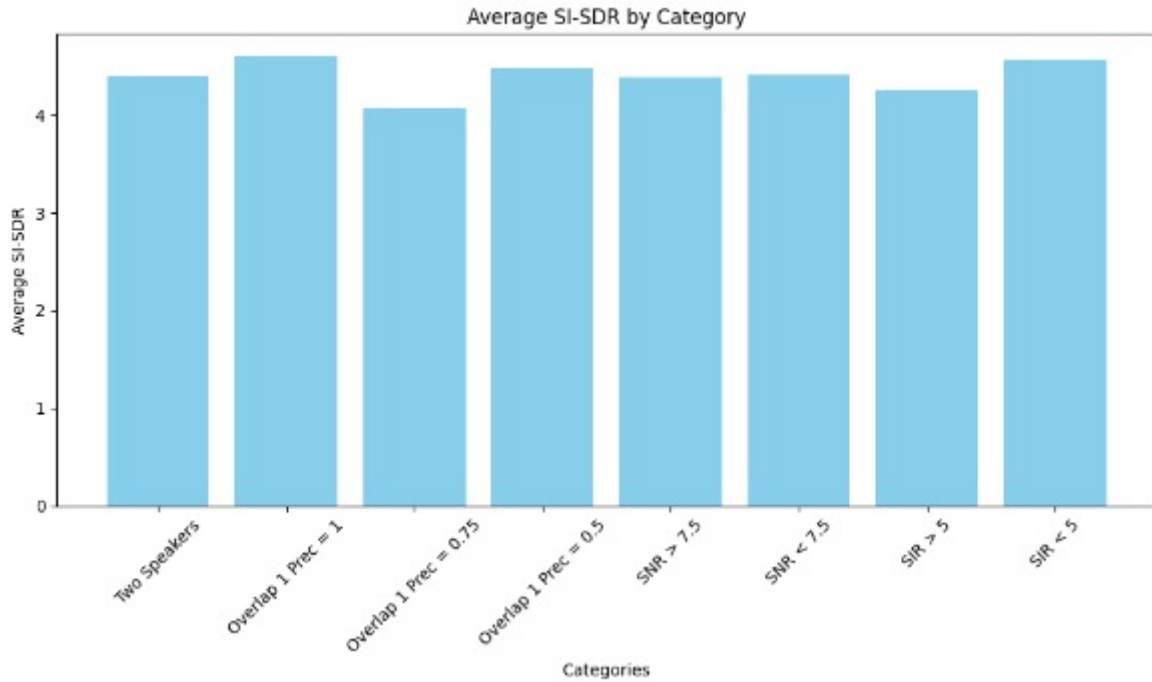


Figure 25

Insights

The model achieved excellent results, with the separated signals sounding clear and closely matching the original signals. Although there remains a slight difference between the original and reconstructed signals due to the added noise, the model effectively managed the separation process. The original signal is presented before noise was introduced, while the reconstructed signal reflects the separation after noise was added, demonstrating the model's strong performance. Notably, this example presented a challenging scenario, as one signal was significantly stronger than the other, yet the model still achieved high-quality separation.

Additionally, the categories have minimal impact on the SI-SDR, for the reasons discussed previously.

9.5 model 4- 2 speakers with noise

For the training and testing process we created 57,000 samples. 88% for training, 8% for validation and 4% for testing. In total:

50,160- for training

4,560- for validation

2,280- for testing

$C = 3$ – for the masks and loss

Same as in model 3 but with a mask for the noise

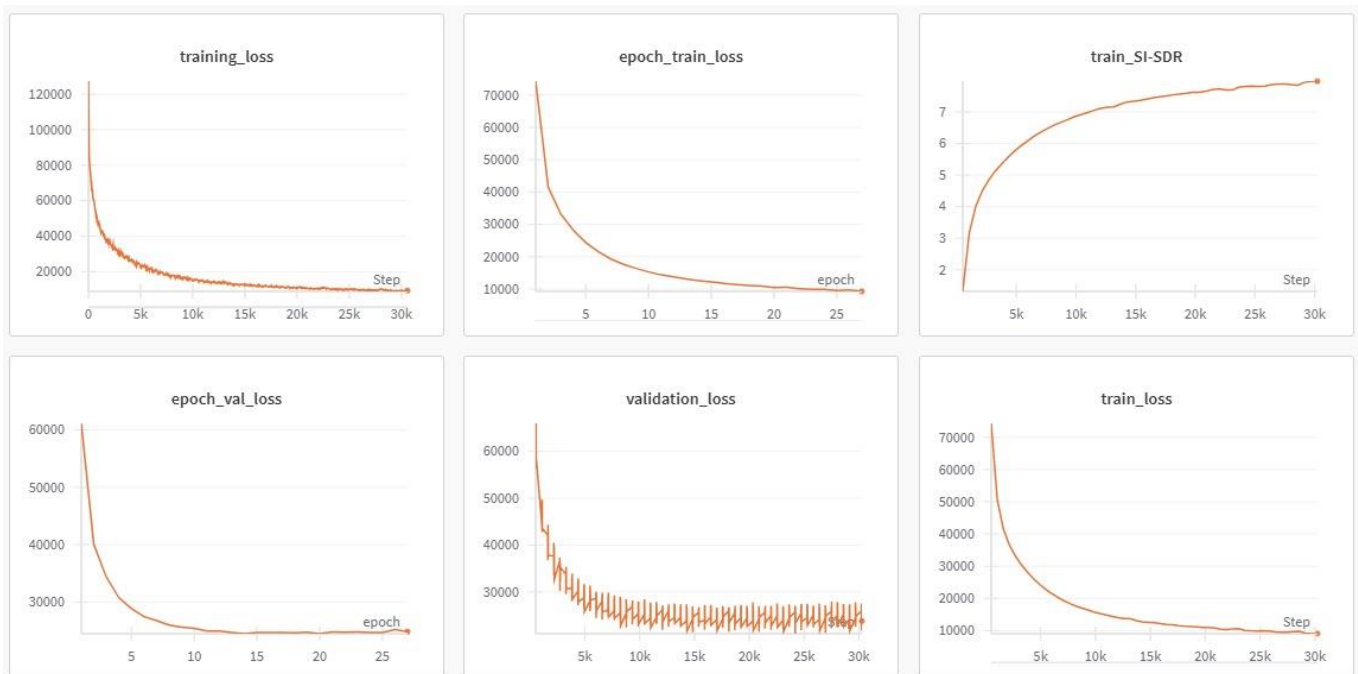
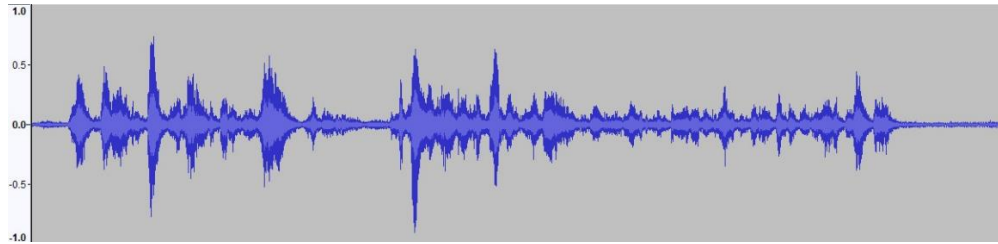


Figure 26

We would like to make a comparison between the results of model c2 and c3 therefore we will use the same mix signal.

Mix signal:



Mix signal spectrogram:

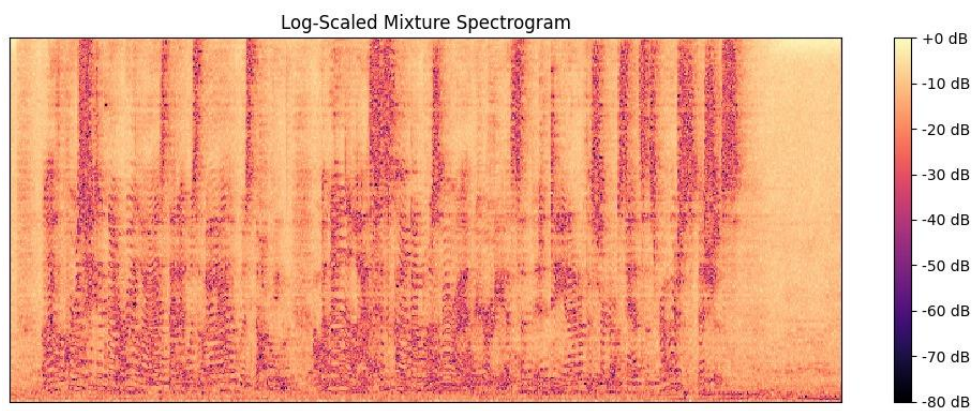


Figure 27

Comparison between the separated noise signal and the original:

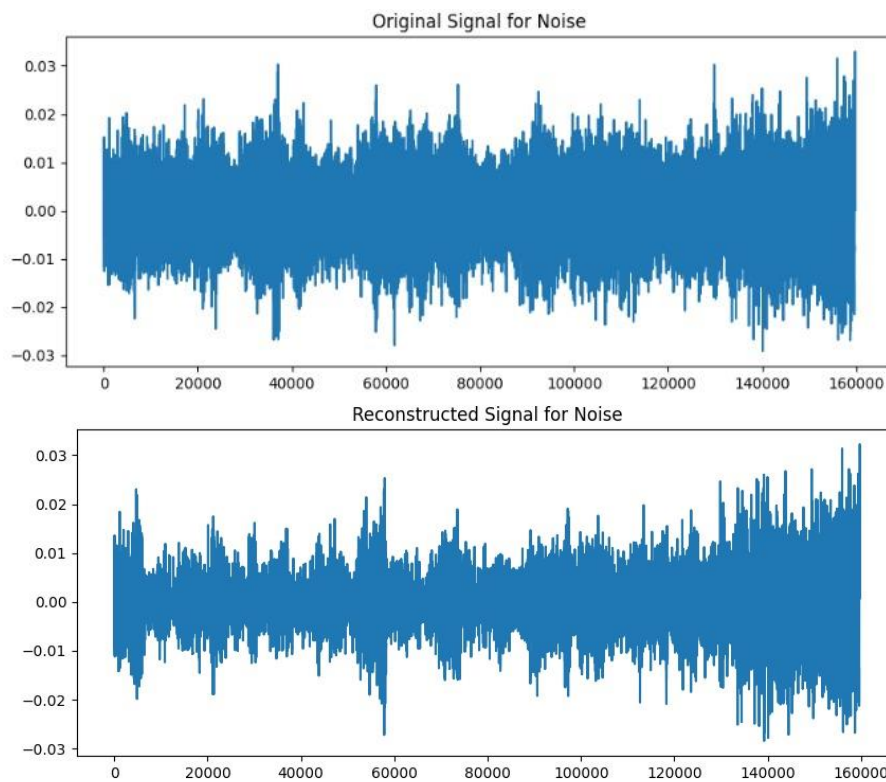
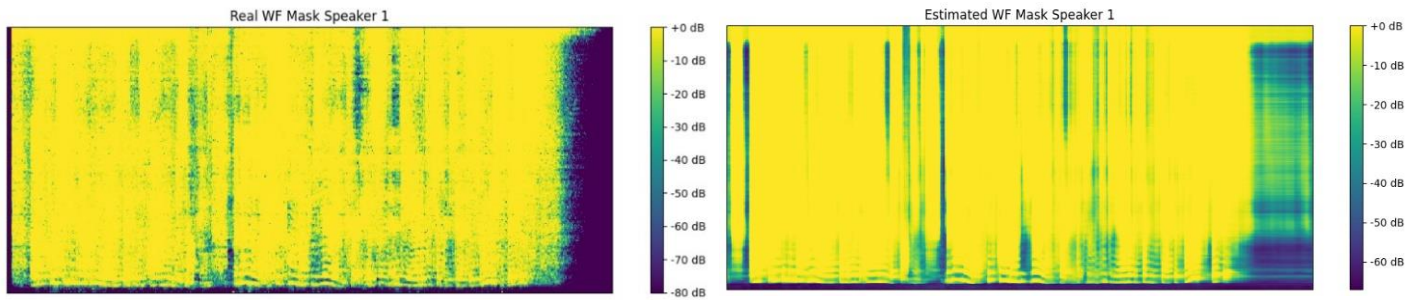
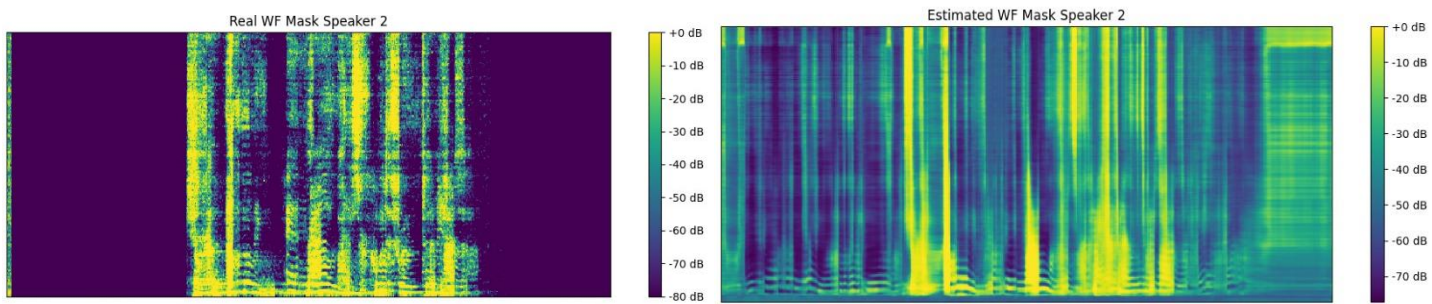


Figure 28

Real and estimated WF mask of speaker 1:



Real and estimated WF mask of speaker 2:



Real and estimated WF mask of noise:

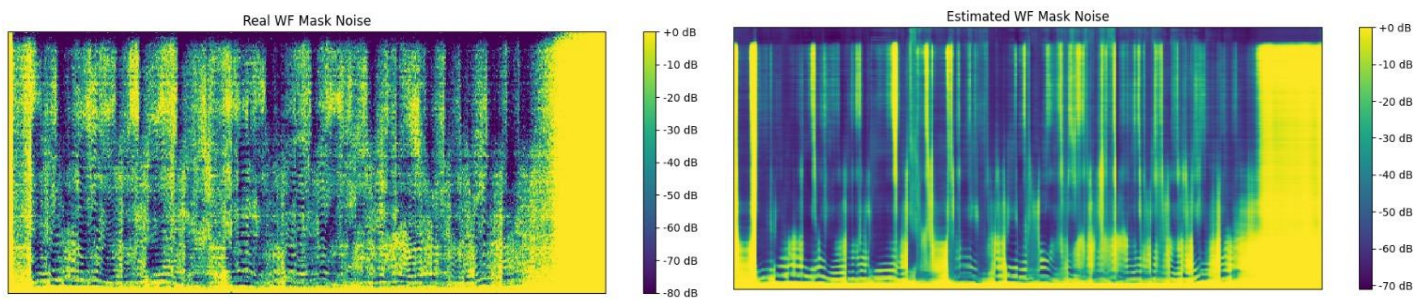


Figure 29

Comparison between the separated signal and the original:

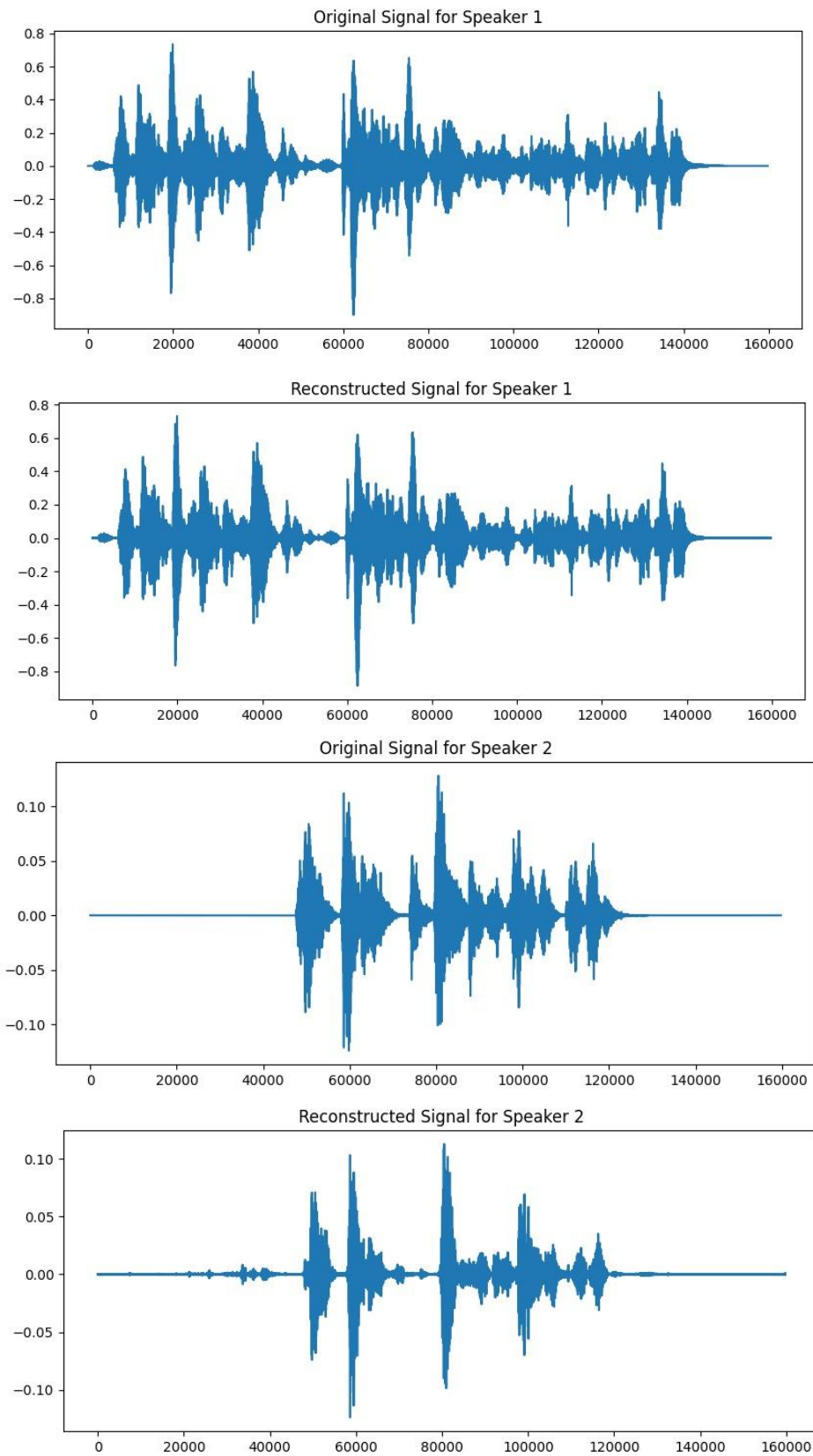


Figure 30

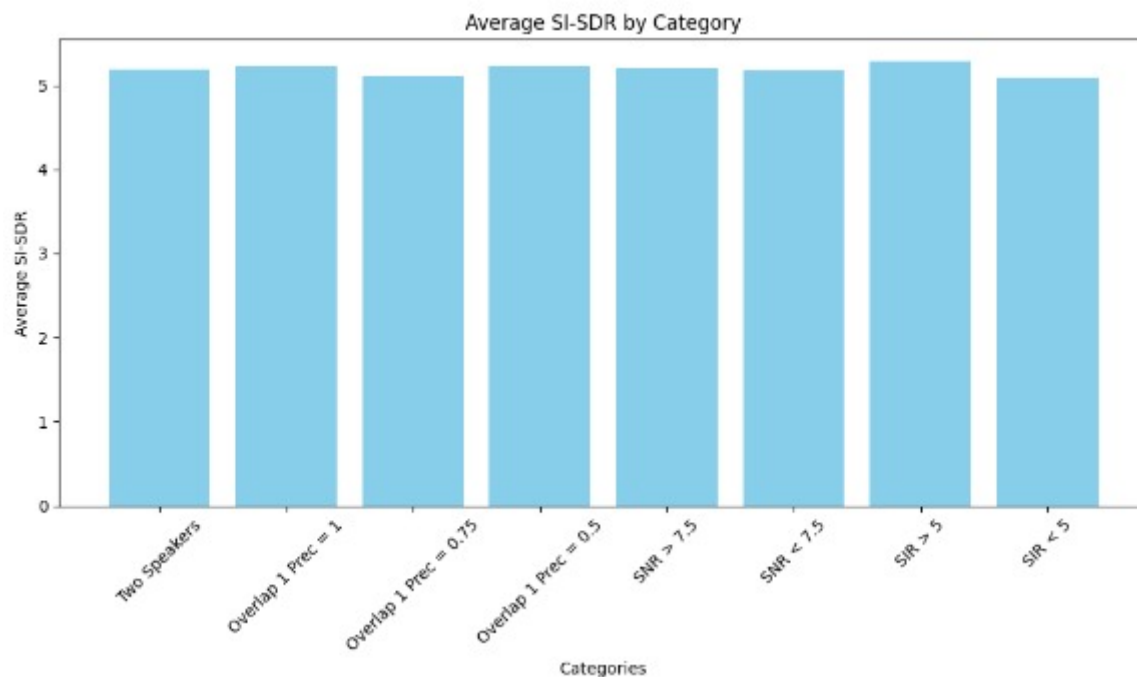


Figure 31

Insights

The model achieves strong results, with the reconstructed signals closely resembling the originals and clear-sounding separation. Notably, the model effectively handles noise by outputting three distinct signals: two for the speakers and one for the noise. As with model 3, we tested a challenging example with a complex mix, yet this model still delivered excellent separation results.

For this model as well, the SI-SDR remains relatively uniform for the reasons discussed earlier. However, it is now higher, indicating improved separation performance.

10 Summary and Comparison of Models

Model 2 demonstrates exceptional performance, producing reproduced signals that closely resemble the originals with clear separation quality. The high SI-SDR values indicate strong separation, with minimal impact from the categories, likely due to the random nature of data creation where overlapping categories balance each other out. Operating without noise allows this model to achieve the highest SI-SDR among all models.

In contrast, Model 3 also achieves excellent results, with separated signals that sound clear and closely match the originals. However, slight differences arise from the added noise. Despite this challenge, the model manages effective separation, even when one signal is significantly stronger. The categories still show minimal impact on the SI-SDR, though noise slightly affects the overall separation quality.

Model 4 builds on the strengths of the previous models, delivering strong results while effectively handling noise by outputting three distinct signals: two for the speakers and one for the noise. The SI-SDR remains relatively uniform across categories but is higher than in Model 3, indicating improved separation performance. The inclusion of a specific noise mask enables Model 4 to outperform Model 3 in noisy conditions, leading to enhanced separation quality.

In summary, Model 2 offers the best SI-SDR without noise, facilitating easier separation. Models 3 and 4 effectively manage noisy mixes, with Model 4 showing superior performance due to its additional noise mask, resulting in better separation quality and a higher SI-SDR.

11 Audio files

Model 2:



Model 3:



Model 4:



12 Bibliography

Stephenson, C., Callier, P., Ganesh, A., & Ni, K. (2017, May 12). *Monaural audio speaker separation using source-contrastive estimation*.

Luo, Y., Chen, Z., & Mesgarani, N. (2018, April 18). *Speaker-independent speech separation with deep attractor network*.