Technical University of Denmark

Improving AI-based Automatic Video Dubbing

William Kock-Andersen, s234803@dtu.dk
Tobias Rodrigues Bjerre, s230323@dtu.dk
Laura Munch Bjerg, s234865@dtu.dk

June 2025

Abstract

In a world where video is central to communication, education and entertainment, the language barrier remains a key obstacle for global accessibility. An existing baseline automated dubbing system suffer from missing background sounds and unnatural voice output due to poor speaker attribution, misalignment and the use of global voice references. These issues limit the usability and quality of automatically dubbed content, especially in complex multi-speaker or noisy environments.

We propose a redesigned dubbing pipeline that improves three critical areas: (1) audio-based speaker diarization using AssemblyAI, (2) source separation with Hybrid Demucs to isolate clean vocal and background tracks, and (3) pitch-preserving time-stretching with Audiostretchy to better synchronize dubbed audio with visual content. Additionally, we introduce localized reference windows for voice conversion to preserve speaker-specific vocal traits.

Our model was tested on a curated dataset of 24 diverse videos. Compared to the baseline, our system reduced global Diarization Error Rate (DER) from 45.3% to 36.6%, and nearly halved the cpWER from 24.6% to 12.2%, showing improved accuracy in both speaker labeling and transcription alignment. A paired t-test showed that these improvements were not statistically significant for both DER and cpWER with a p-value greater than 0.01. In a user study with 9 participants, Mean Opinion Scores (MOS) improved significantly: +1.19 for voice conversion and +2.49 for overall dubbing quality. Statistical tests confirmed these differences as significant ($p < 0.01$). Background audio mixing, enabled by Demucs, also received a solid MOS of 7.28, contributing to an overall better viewer experience.

Our refined pipeline substantially improves the naturalness, accuracy, and timing of automatic dubbing, making it a more practical and scalable solution for multilingual video production. Challenges remain with speaker count estimation, but the overall system demonstrates strong potential for real-world use.

Contents

1 Introduction

In today's global digital landscape, video content plays a central role in entertainment, education, and communication. However, language remains a persistent barrier that limits the accessibility and reach of such content. Traditional dubbing workflows, while effective, are both expensive and labor-intensive, requiring teams of professional translators, voice actors, and audio engineers. These constraints make high-quality dubbing impractical for many smaller creators, educational institutions, and non-commercial projects. As a result, there is a growing need for more scalable and cost-effective solutions that can democratize access to dubbing. Advances in machine learning and speech processing have opened new possibilities for automatic dubbing. These tools promise to significantly reduce the manual effort involved while nearing the quality of manually dubbed content. In this project, we build on a baseline system developed in a previous DTU bachelor project [1], and propose a series of targeted improvements to enhance the intelligibility, accuracy, and naturalness of automatically dubbed speech. Our aim is to create a system that can produce high-quality dubbed videos with minimal human intervention, making multilingual content creation more feasible and affordable.

While the original baseline system demonstrated that end-to-end automatic dubbing is feasible using off-the-shelf models, it exhibited several limitations that impacted the overall quality of the output. First, it relied on GPT-4o for speaker diarization, a fully text-based method, which can lead to speaker misattributions, especially in cases of acoustic ambiguity or overlapping dialogue. Furthermore, transcription was performed directly on raw audio, often containing background music and noise, which degraded transcription accuracy. The baseline also used a voice conversion strategy based on large, globally defined reference segments for each speaker, which limited the ability to capture local, speaker-specific vocal characteristics. Lastly, to align the timing of dubbed speech with the original, the baseline employed a poorly documented length-adjustment parameter within the voice conversion model, an approach that, in some cases, failed to preserve the pitch and naturalness of the original speech.

To address these shortcomings, we redesigned the pipeline focusing on three critical areas: **speaker diarization**, **source separation**, and **time-stretching with pitch preservation**. Our model first separates speech from background elements using Demucs before transcription, improving both transcription accuracy and downstream processing. For diarization, we adopt an audio-based method using AssemblyAI, which outputs time-stamped speaker labels to enable more reliable speaker identification. We further improve voice conversion by introducing a localized reference window strategy, in which we use surrounding utterances from the same speaker, rather than all their speech throughout the video, as target examples for each segment. This allows the model to capture subtle, local variations in tone and delivery. Finally, instead of relying on the voice conversion model's built-in length adjustment, we apply an external time-domain pitch-preserving time-stretching algorithm using audiostretchy, which gives us more control and higher quality when synchronizing dubbed audio with the original.

Our full pipeline integrates several state-of-the-art components: transcription via the Whisper-based wrapper Stable-TS, speaker diarization via AssemblyAI, source separation using Demucs, translation through DeepL, text-to-speech synthesis with Google Cloud TTS, and voice conversion via SEED-VC. Many of these tools are also used in the baseline system. However, our work refines how they interact and introduces targeted improvements that make the pipeline more modular, accurate, and robust. Rather than developing entirely new components, we focus on reengineering the structure of the dubbing process. Our improvements aim to address specific weaknesses in the baseline, such as poor speaker labeling, missing background sounds and limited voice conversion fidelity, while introducing a new modular and accessible approach that is easier to extend [2].

This report investigates the effectiveness of these refinements in improving the voice conversion and overall enjoyment of dubbed audio. In the following section, we present the central research questions that guided our design decisions and evaluation strategy.

1.1 Research Questions

To evaluate the effectiveness of our pipeline refinements, we focus on three key problem areas where the baseline system showed notable weaknesses. These research questions aim to assess whether our modifications improve the overall quality, consistency, and timing of the dubbed output.

How can speaker diarization technologies be used to improve auto dubbing with respect to voice synthesis? To produce realistic voice conversion in automated dubbing, it is essential to accurately determine who is speaking and when. In our project, we use AssemblyAI's speaker diarization system to detect speaker segments directly from the audio. This replaces the baseline's approach, which relied on prompting language models to group text segments by speaker identity. AssemblyAI provides time-stamped speaker labels, which we align with transcription segments from Stable-TS to ensure that each translated utterance is assigned to the correct speaker. These speaker labels are particularly important in the voice conversion step, where we modify the output of the text-to-speech (TTS) model to match the identity of the original speaker. To evaluate the performance of the diarization system, we use Diarization Error Rate (DER), Speaker Error Rate, concatenated minimum-permutation Word Error Rate (cpWER), and to quantify the perceptual quality of the resulting voices, we collect feedback using the Mean Opinion Score (MOS) metric.

To what extent can source separation models improve the mixing of dubbed speech and background audio by separating noise and background sounds from speech? Our goal is to retain the background sound from the original video while replacing the speech with translated voices. To achieve this, we use the Demucs model to separate the original audio into two tracks: speech and background. This improves the audio clarity of the input used for transcription and makes it easier to isolate speech for processing, while preserving the original background audio in the final mix. We evaluate the impact of this approach by asking users to rate how natural the final mix sounds, considering both speech and background, using the Mean Opinion Score (MOS).

To what extent can audio time-stretching with pitch preservation and the use of localized reference windows improve voice conversion? TTS-generated audio rarely matches the timing of the original speech, which can result in misaligned lip sync and unnatural pacing. To address this, we use the Python tool audiostretchy to stretch or compress the converted speech while preserving pitch. This ensures that the dubbed voice fits the original timing, improving the rhythm and visual alignment. Additionally, we improve the voice conversion step by using a localized reference window around each utterance, specifically, adjacent segments spoken by the same speaker, as target audio. This helps the model capture subtle, local voice characteristics such as tone and energy. We evaluate the effect of this approach by comparing the alignment of the dubbed and original speech and by collecting MOS ratings for the naturalness and consistency of speaker identity in the converted voices.

The remainder of this report is structured as follows. Section 2 describes the data used for evaluation and how we expanded the original test set from the baseline project. Section 3 details the methods behind our improved pipeline, including a breakdown of architectural differences, theoretical background on source separation, speaker diarization, and time-stretching, and a description of the evaluation metrics and collection procedure. Section 4 presents the results and statistical analyses. Finally, Section 5 discusses the findings in relation to the research questions and outlines the broader implications of our work.

2 Data

The quality and diversity of the evaluation dataset plays a crucial role when evaluating an automatic dubbing system. A well-constructed dataset must reflect the range of acoustic and structural challenges the system is designed to handle. Without this, it becomes difficult to determine whether performance truly generalizes or the system merely performs well in handpicked settings. In reviewing the previous DTU bachelor project that this work builds on [1], we found that the baseline system did not include a formal description or analysis of the evaluation dataset. While the final GitHub repository contained dubbed outputs for 7 videos, the source and properties of these videos were not discussed in the report. No systematic effort was made to assess the dataset's coverage in terms of number of speakers, clip length, or background conditions, all of which are critical factors in dubbing performance.

To address this limitation, we curated a more extensive and well-documented dataset of 24 videos, combining the original 7 baseline videos with 17 additional clips sourced primarily from YouTube. These clips vary in length, speaker configuration, background conditions, and dialogue complexity. This extended dataset allows us to better assess how well the pipeline generalizes under diverse real-world conditions and enables more reliable statistical evaluation across a broader range of test cases. The rest of this section describes the dataset in detail, outlines the annotation process, presents key statistics, and briefly discusses ethical considerations related to data sourcing and storage.

2.1 Dataset

To enable detailed evaluation of the pipeline, we manually annotated all 24 videos in the dataset with segment-level ground truth data. Each utterance was labeled with:

- **Timestamps:** Precise start and end boundaries for each utterance, annotated with a resolution of 10 ms.
- **Speaker Labels:** Speaker identities assigned consistently across segments.
- **Transcriptions:** Verbatim transcriptions of the original speech.

These annotations form the backbone of our evaluation, allowing us to compute objective metrics such as DER, Speaker Error Rate, and cpWER. Note that, although timestamps are defined with high resolution (10 ms), we acknowledge a degree of uncertainty inherent in manual annotation, as some utterances can be challenging to annotate precisely. Despite the time-consuming nature of manual annotation, this process enabled us to construct a higher quality benchmark set suitable for both quantitative and perceptual evaluation. A summary of the dataset, including video duration, speech duration, number of utterances, speaker count, average utterances per speaker and utterance length, is presented in Table 1.

The full list of videos is available under input in our shared folder[3]. The first seven clips in the dataset come from the original baseline project, while the remaining 17 were added in this work to expand coverage. As shown, the full dataset spans over 61 minutes of video and contains 1,110 utterances across 85 unique speaker tracks. The average video contains roughly 46 utterances and 3–4 speakers, with a wide range in utterance length (0.97–3.50s) and speaker density. This diversity allows us to evaluate how well individual components of the pipeline — including diarization (DER, speaker error), transcription (cpWER), and voice conversion (MOS) — perform under varied acoustic and structural conditions for each video. The dataset includes both clean monologues and complex multi-party dialogues with background interference, providing a valuable stress test for system robustness.

Table 1: Summary statistics for evaluation videos.

Video	**Dur.**	**Speech dur**	**Utt**	**Spk.**	**Utts/spk**	**Avg. utt. len.**
	mm:ss	mm:ss	count	count	count	s
1	3:24	2:56	63	1	63.0	2.81
2	1:21	0:29	21	2	10.5	1.43
3	1:45	1:20	32	2	16.0	2.53
4	2:29	2:23	74	3	24.7	1.93
5	1:10	1:03	25	3	8.3	2.55
6	0:54	0:31	20	2	10.0	1.58
7	1:14	1:00	22	2	11.0	2.77
8	2:25	1:52	60	3	20.0	1.87
9	1:14	0:31	20	3	6.7	1.59
10	3:05	1:23	43	2	21.5	1.94
11	4:06	3:51	66	2	33.0	3.50
12	2:39	1:40	51	5	10.2	1.97
13	4:27	2:37	107	3	35.7	1.47
14	1:00	0:44	31	4	7.8	1.43
15	1:38	1:02	41	7	5.9	1.51
16	1:21	0:24	25	3	8.3	0.97
17	3:36	2:36	62	7	8.9	2.52
18	2:06	1:37	32	2	16.0	3.04
19	2:06	1:28	38	4	9.5	2.33
20	2:33	1:06	38	2	19.0	1.74
21	3:10	1:49	67	2	33.5	1.63
22	5:43	1:11	36	6	6.0	2.00
23	3:56	2:25	87	5	17.4	1.68
24	3:39	2:35	49	10	4.9	3.17
Avg	2:32 ± 1:06	1:36 ± 0:54	46.2 ± 23.0	3.5 ± 2.1	17.0 ± 13.4	2.08 ± 0.65
Max	5:43	3:51	107	10	63.0	3.50
Min	0:54	0:24	20	1	4.9	0.97
Total	61:10	38:45	1110	85	–	–

It should be noted that all videos use English as input and Danish as output, reflecting a realistic dubbing scenario for the authors' linguistic context. However, structural differences between the languages, such as sentence length and word order, can introduce biases in timing and naturalness during synthesis. Furthermore, the selection of videos could potentially introduce biases, however we did our best to select a wide variety of videos with varying features to obtain a representative dataset. Regarding ethical considerations all material was sourced from publicly available online content and is used solely for non-commercial academic purposes. No private or sensitive data is included, and annotated data and outputs are stored in a secure, access-controlled environment.

3 Methods

This section details the core improvements we introduced to enhance the performance and robustness of the baseline automatic dubbing system. Specifically, we focus on the three key areas where the baseline showed limitations: source separation, speaker diarization, and voice conversion with time-stretching and pitch preservation. We begin by outlining the overall structure of the original baseline pipeline, identifying which components remain unchanged and where our modifications are applied. The full code with all the implementation can be found in our Github repository[2]. For each modified component, we present a conceptual overview, discuss relevant theoretical foundations and prior work, and explain how the new implementation integrates with the rest of the system. Components that we did not significantly alter — such as machine translation and TTS synthesis — are not treated in detail here; we refer the reader to the baseline report [1] for a discussion of those modules. Finally, we describe our evaluation approach. We outline the metrics used to assess the performance of each pipeline component, explain how they were gathered, and describe how they relate to the research questions posed in Section 1.1. These include both objective metrics — such as Diarization Error Rate (DER), concatenated minimum-permutation Word Error Rate (cpWER), and Speaker Error Rate — and quantified subjective metrics collected through Mean Opinion Score (MOS) evaluations.

3.1 Pipeline comparison

Our full dubbing pipeline consists of a sequence of processing steps, many of which are shared with the original baseline system. However, we introduce targeted improvements to key components where the baseline showed limitations. Figure 1 provides a visual comparison between the two systems, highlighting both the shared structure and our key modifications. In the remainder of this subsection, we walk through each stage of the pipeline in order, describing how each part is handled and what changes we introduced.

Audio Extraction. Both pipelines begin by extracting audio from the input video using FFmpeg, converting it to high-quality WAV format suitable for downstream processing.

Source Separation. In the baseline system, transcription was performed directly on the raw audio, which often contained background music, ambient noise, or sound effects, degrading the accuracy of the module and downstream tasks. In our pipeline, we apply the Demucs model [4] to separate the audio into two distinct sources: vocals and background. We then use the isolated vocal track as input for transcription and diarization, which improves the clarity of the speech signal. Crucially, this approach also allows us to retain the original background audio and mix it back in during the final reconstruction, resulting in a more natural and immersive dubbed output.

Transcription. Both systems use Stable Whisper for transcription. However, we use an updated model and make minor parameter adjustments to improve timestamp accuracy and utterance grouping. These improvements are described in more detail in Section 3.5.

Speaker Diarization. The baseline relied on a text-based speaker diarization approach using Chat GPT-4o, which grouped transcript segments by inferred speaker identity. This often resulted in incorrect attributions in difficult conditions. In contrast, our system uses AssemblyAI's audio-based diarization API[5], which provides time-stamped speaker labels. We align these labels to the transcription using overlap matching, ensuring accurate speaker assignments for each utterance, to create a structured transcript with both timestamps and speaker information.

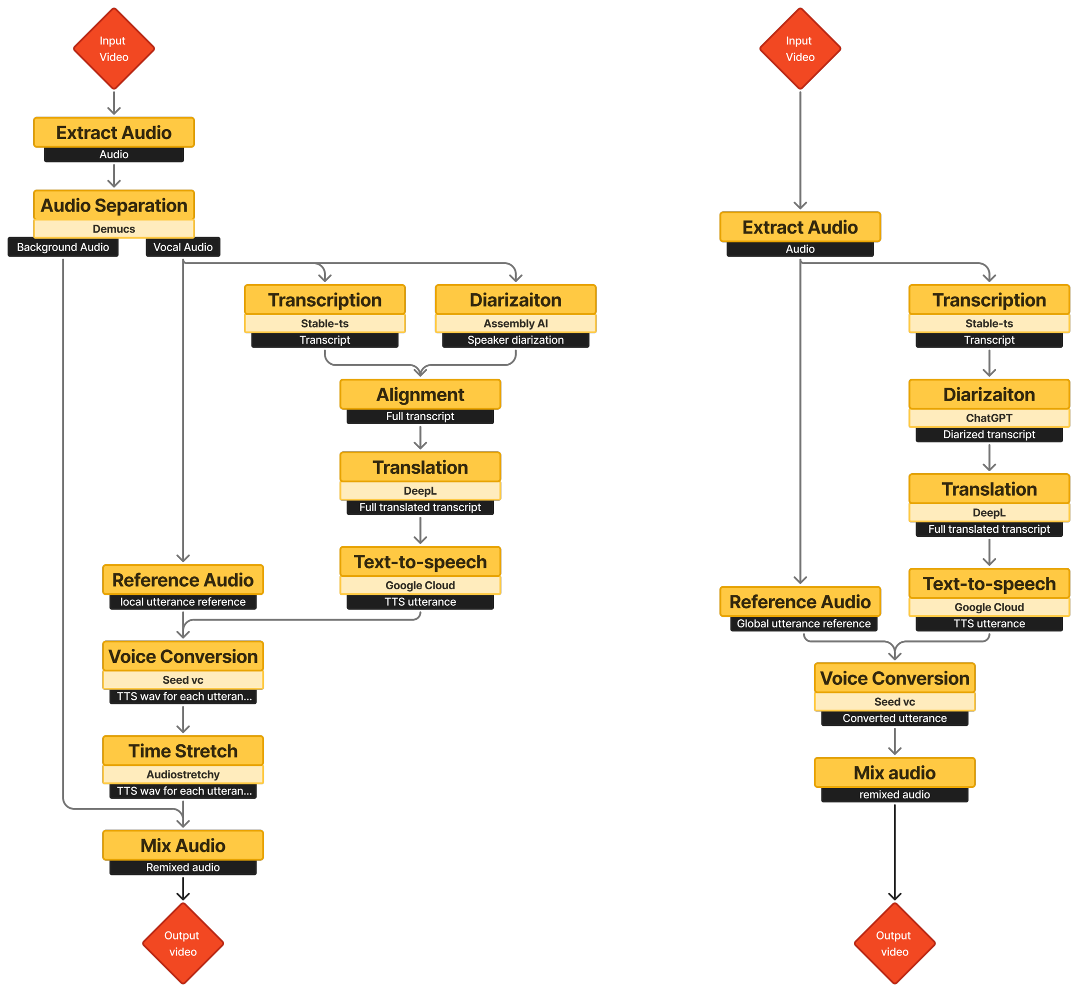


Figure 1: Left: Our pipeline. Right: Baseline pipeline

Translation. Both models translate each text segment in the transcript from English to Danish using DeepL's API. This translated text is then used to generate the dubbed speech.

Text-to-Speech (TTS). Both pipelines use Google Cloud TTS to synthesize speech from the translated text. We specify the Danish voice 'da-DK-Neural2-D' for all utterances in the updated systems.

Voice Conversion. SEED-VC is used in both pipelines for converting TTS voices to match the original speakers. However, the baseline used a global speaker reference—i.e., all speech from that speaker across the video as the target. We instead use a localized reference window: the utterances immediately before and after the target segment from the same speaker. This allows the model to capture more nuanced, context-specific voice characteristics such as tone and emotion (see Section 3.4.2).

Time-Stretching. To align the timing of dubbed audio with the original, the baseline used a built-in length-adjustment parameter in SEED-VC. We found this method unpredictable and poorly documented. Instead, we apply post-processing using the 'audiostretchy' library, which performs pitch-preserving time-stretching using TDHS. This gives us more reliable control over timing and ensures that the dubbed audio aligns with the original video, which improves lip syncing.

Final Assembly. In the final step, the baseline system overlays speech on silence, discarding all original background audio. Our pipeline instead mixes the voice-converted speech with the original background track (from the source separation step), producing a more natural and immersive final dubbed video.

3.2 Speaker Diarization

Speaker diarization is the process of determining "who spoke when" in a multi-speaker audio stream by partitioning the input into segments, each labeled according to speaker identity. In our automated dubbing pipeline, the diarization module plays a key role in both improving the performance of the voice synthesis module, by providing consistent speaker representations, and ensuring each utterance is correctly assigned a speaker identity in the dubbed output.

The baseline system relied solely on the contextual and linguistic reasoning abilities of a **large language model (LLM)**. A full transcript was given as input to ChatGPT 4o via an integrated API, prompting the LLM to infer the number of speakers and assign each utterance accordingly. While this approach could be somewhat effective for simple scenarios, it struggled with accuracy and robustness in more complex cases involving numerous speakers or ambiguous transcriptions.

To address these limitations, we transitioned to a dedicated speaker diarization system provided by AssemblyAI. Their API implements advanced embedding-based diarization using methods inspired by the work of Wan et al. and Wang et al., as detailed in AssemblyAI's blog and corresponding academic publications [6, 7, 8].

The speaker diarization system integrated into our model consists of four main stages: **feature extraction**, **grouping**, **embedding extraction**, and **clustering**. First, the input audio is divided into short frames of 25 ms with a 10 ms step size. For each frame, 40-dimensional log-mel filterbank energies are extracted, capturing perceptually relevant acoustic features. These frame-level features are then grouped into overlapping sliding windows of 240 ms with a 120 ms step size, forming segments that provide sufficient temporal context for speaker representation.

To process each segment, the system uses a multilayer **long short-term memory (LSTM)** neural network. LSTMs are a type of **recurrent neural network (RNN)** specifically designed to handle sequential data by preserving long-range temporal dependencies, which are crucial in tasks like speech analysis. Traditional RNNs often suffer from the vanishing gradient problem, making them ineffective at learning dependencies over long sequences. LSTMs overcome this limitation by introducing a memory cell and a set of gating mechanisms that regulate the flow of information.

As illustrated in Figure 3, each LSTM unit receives the current input frame x_t, the previous hidden state h_{t-1}, and the previous cell state C_{t-1}. It computes three gates using learned weights and biases: a **forget gate** f_t, which decides what information to discard from the previous cell state; an **input gate** i_t, which determines what new information to add; and an **output gate** o_t, which decides what information to pass on to the next time step. The updated cell state C_t combines the filtered past

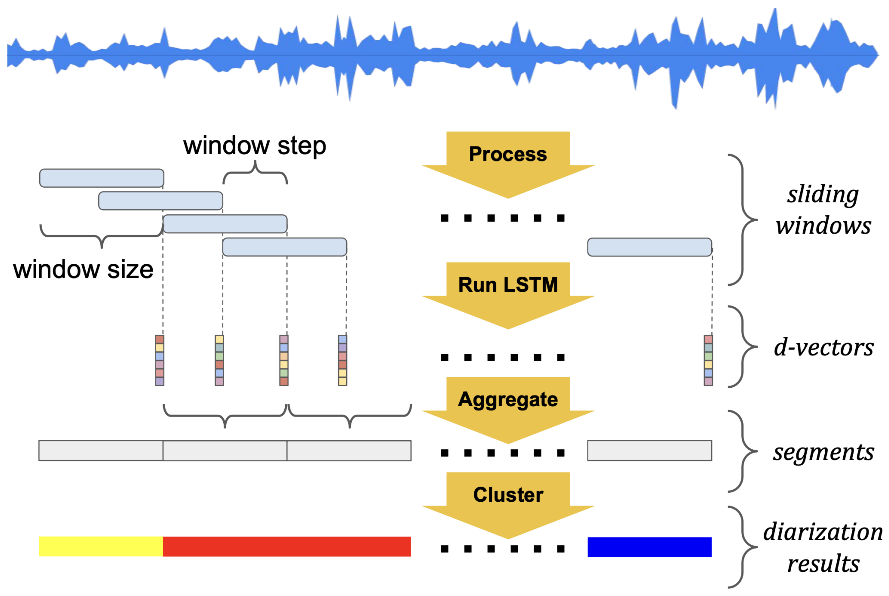


Figure 2: A flowchart of a d-vector based diarization system.

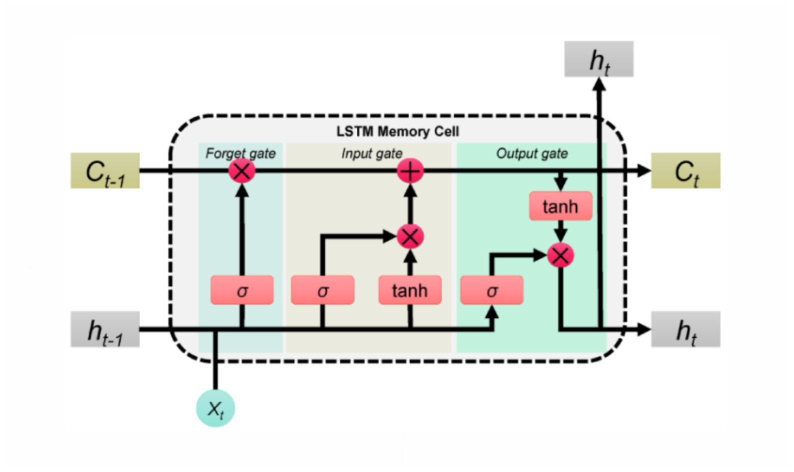


Figure 3: Structure of a single LSTM memory cell.

and newly added information, and the current hidden state h_t is computed based on this updated memory. This architecture enables LSTMs to learn speaker characteristics across multiple frames. In our case, each windowed segment is fed into the LSTM, and the final hidden state is projected and L2-normalized to yield a 256-dimensional **d-vector**, which captures speaker identity:

This design allows LSTMs to maintain and update a memory of relevant speaker information across frames, making them well-suited for learning consistent speaker embeddings. In our case, each windowed segment is fed into the LSTM, and the final hidden state is used to generate a fixed-length

256-dimensional **d-vector**, capturing the unique vocal characteristics of the speaker within that segment.

$$e_{ji} = \frac{f(x_{ji}; \mathbf{w})}{\|f(x_{ji}; \mathbf{w})\|_2}$$

where $f(\cdot; \mathbf{w})$ denotes the LSTM network with parameters $\mathbf{w}$, and x_{ji} is the input segment from speaker j, utterance i. These d-vectors are designed such that utterances from the same speaker are close together in the embedding space, while utterances from different speakers are far apart.

To achieve this separation, the LSTM is trained using the **generalized end-to-end (GE2E)** loss function [6]. GE2E is designed to encourage high similarity between utterances from the same speaker while reducing similarity across different speakers. Each training batch contains multiple speakers, each with several utterances. For every d-vector e_{ji}, the model computes its similarity to all speaker centroids in the batch. The centroid for speaker k is computed as the mean of their d-vectors:

$$c_k = \frac{1}{M} \sum_{i=1}^{M} e_{ki}$$

The similarity matrix is then computed using scaled cosine similarity:

$$S_{ji,k} = w \cdot \cos(\mathbf{e}_{ji}, \mathbf{c}_k) + b$$

where $w > 0$ and b are learnable parameters. The softmax-based objective pushes embeddings toward the correct speaker and pulls them away from all others:

$$\mathcal{L}(e_{ji}) = -S_{ji,j} + \log \sum_{k=1}^{N} \exp(S_{ji,k})$$

Compared to traditional contrastive loss functions, GE2E is more efficient and stable, as each batch jointly optimizes over all speaker pairs in the similarity matrix. The result is a robust embedding space that significantly improves the separability of speakers, especially in downstream clustering tasks.

At inference time, speaker labels are unknown. The trained LSTM model is used to compute d-vectors for each audio segment:

$$e_i = \frac{f(x_i; \mathbf{w})}{\|f(x_i; \mathbf{w})\|_2}$$

Only speech segments are relevant to the diarization task, so **Voice Activity Detection (VAD)** is applied to discard non-speech regions. For each detected speech segment, a segment-level embedding is computed by averaging all d-vectors corresponding to the windows within that segment:

$$e_k = \frac{1}{K} \sum_{i=1}^{K} \frac{f(x_{i_k}; \mathbf{w})}{\|f(x_{i_k}; \mathbf{w})\|_2}$$

where e_k is the segment embedding, K is total amount of windows in segment k and $x_{i,k}$ is the window input i belonging to segment k.

To assign each segment embedding to a speaker, the system employs a two-step clustering process. As shown in Figure 4, speaker embeddings often have complex distributions, imbalanced speaker contributions, or they are influenced by higher-level characteristics like gender or accent. These properties make simple K-Means clustering insufficient to robustly separate speakers due to its assumption of

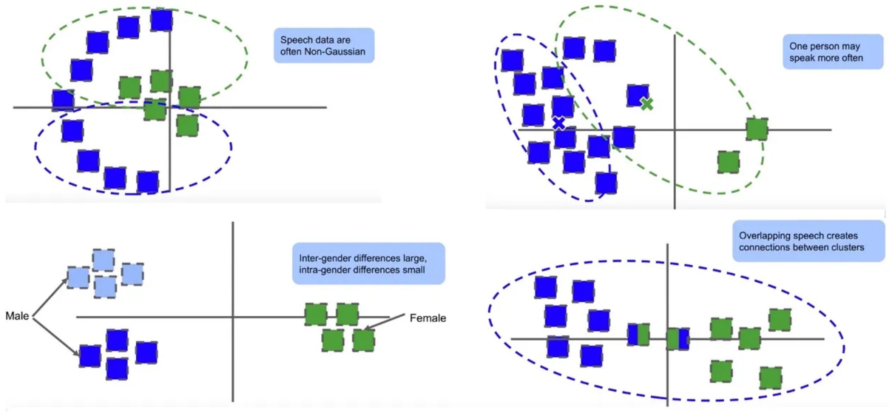

Speech data are often Non-Gaussian
One person may speak more often
Inter-gender differences large, intra-gender differences small
Male
Female
Overlapping speech creates connections between clusters

Figure 4: Challenges of only using K-means to cluster speech embeddings

spherical, equally sized clusters.

To address this, the system first applies **spectral clustering**, which begins by constructing an affinity matrix by computing the pairwise cosine similarity between all segment embeddings. However, the raw affinity matrix may contain noise and spurious similarities. Therefore, a series of refinement steps are applied to denoise and enhance its structure, as illustrated in Figure 5:

1. **Gaussian blurring**, to smooth outliers and local inconsistencies.
2. **Row-wise thresholding**, which zeros out weak similarities based on each row's percentile.
3. **Symmetrization**, ensuring the matrix is symmetric (a requirement for spectral methods).
4. **Diffusion**, amplifying group structure by multiplying the matrix with its transpose.
5. **Row-wise max normalization**, to rescale values and stabilize the spectral embedding.

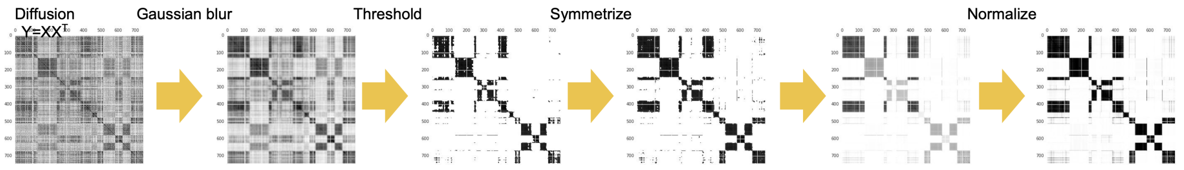


Figure 5: Refinement operations on the affinity matrix.

Next, **eigen-decomposition** is applied to the refined matrix. The number of speakers $\tilde{k}$ is estimated by identifying the largest eigen-gap:

$$\tilde{k} = \arg \max_{1 \leq k < n} \frac{\lambda_k}{\lambda_{k+1}},$$

where $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$ are the eigenvalues. Each embedding is then projected into the $\tilde{k}$-dimensional eigenspace defined by the top eigenvectors. In this transformed space, the embeddings are clustered using K-Means to produce the final speaker labels. This combination of spectral clustering and K-Means improves robustness, especially in challenging conditions where speaker turns are unbalanced or acoustically similar.

3.3 Source seperation with Demucs

Source separation is the general task of separating an input mixture audio signal into K individual source signals, under the assumption that the mixture audio consists of many audio sources. It's essential for our pipeline, since it enables the possibility for separating the input audio into speech and background audio. This both improves the performance of the transcription and voice conversion, and includes the background audio in the final result.

We decided to use Demucs[4] for this task. The model is trained on music tracks, which means it is trained to separate songs into various musical instruments and vocals. This aligns pretty well with our speech extraction task. You can even specify the model to only separate an audio track into two sources: "vocals" and "background". Demucs is a python library, which includes different models, however we used the "mdx_extra_q" model, which was a smaller model developed for the Music Demixing Challenge (MDX), but trained on extra data. This model is a bag of 4 Hybrid Demucs[9] models. Hybrid Demucs models are an extension of regular Demucs, that utilizes both frequency- and time-domian analysis. The regular Demucs model is based on the Wave-U-Net architecture[10].

3.3.1 Wave-U-Net architecture

The Wave-U-Net model is a convolutional network, which operates on wave signals in the time-domain. It takes an input waveform signal M and outputs K different source signals S_k. The architecture (U-shaped) is illustrated in figure 6 (taken from paper).[10]
The model is mostly based on iterative layers of 1D convolution and downsampling or upsampling. The input to the model is the mixture audio M, which is computed as the sum of all the source signals S_k, $k \in \{0, 1, 2, ..., K\}$. The mixture audio is then sent through L downsampling blocks, which are indicated by the yellow boxes in the figure. Each downsampling block DS_i, $i \in \{0, 1, ..., L\}$ is compromised of firstly $F_c \cdot i$ different 1D convolution filters of size f_d. Additionally for each downsampling block, a downsampling method is used which reduces the time resolution. This means it simultaneously increases the number of channels and decreases the amount of time samples. This procedure is then repeated L times before a midway convolution block with $F_c \cdot (L + 1)$ different filters of size f_d is applied. The second part of the model consists of upsampling blocks, which are indicated by the green blocks. The input to each upsampling block is both the previous upsampling block and the corresponding downsampling block with the same dimension. This information feed between the downsampling and upsampling blocks are called skip-connections (indicated by dotted arrows). Upsampling blocks firstly consists of an upsampling layer, which up-scales the time resolution by using a form of linear interpolation. The downsampling blocks essentially work as the encoder for the input and the upsampling blocks work as the decoder.

3.3.2 Demucs architecture

Demucs uses the same model architecture as the Wave-U-Net model with convolutional encoders (downsampling blocks) and decoders (upsampling blocks) and skip-connections between them. Instead of a simple convolution layer between the encoder and decoder, it uses a bidirectional long short-term memory network (BiLSTM). The regular LSTM network is already explained in Section 3.2, but this is a bidirectional, which means it processes information from sequences in two directions. It uses two LSTM networks: one processes the input forwards ($x_0 \rightarrow x_T$), while the other processes the input backwards ($x_T \rightarrow x_0$) as shown in figure 7. (taken from article)[11]
This enables the network to capture wide temporal context in the waveform signal from both past and future. The full architecture of the Demucs model is illustrated on the left in figure 8 (taken from paper).[12]

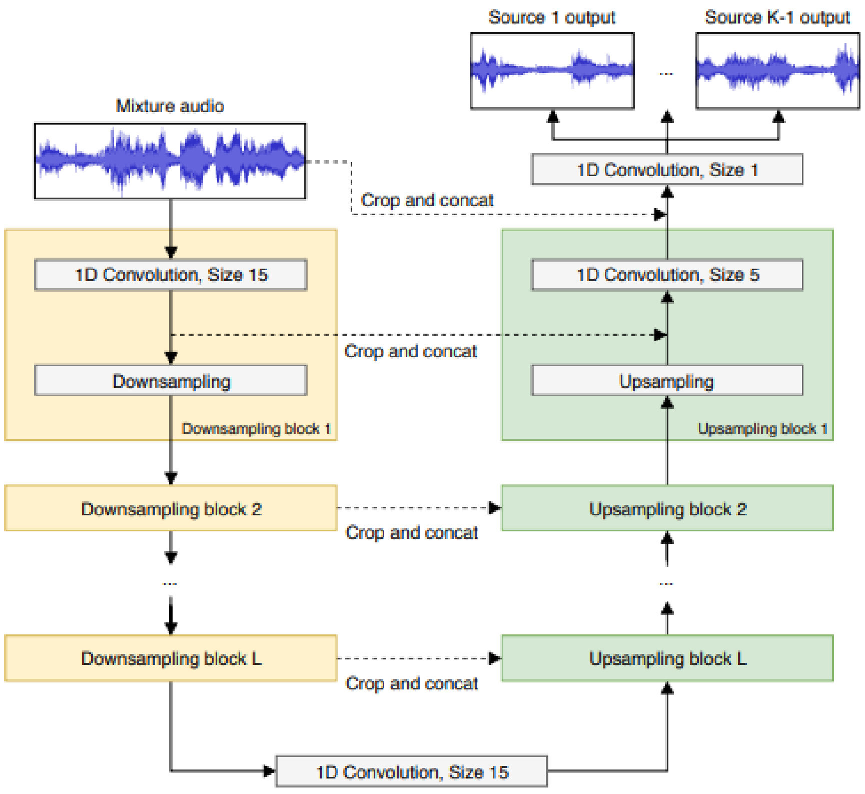


Figure 6: Architecture of the Wave-U-Net model

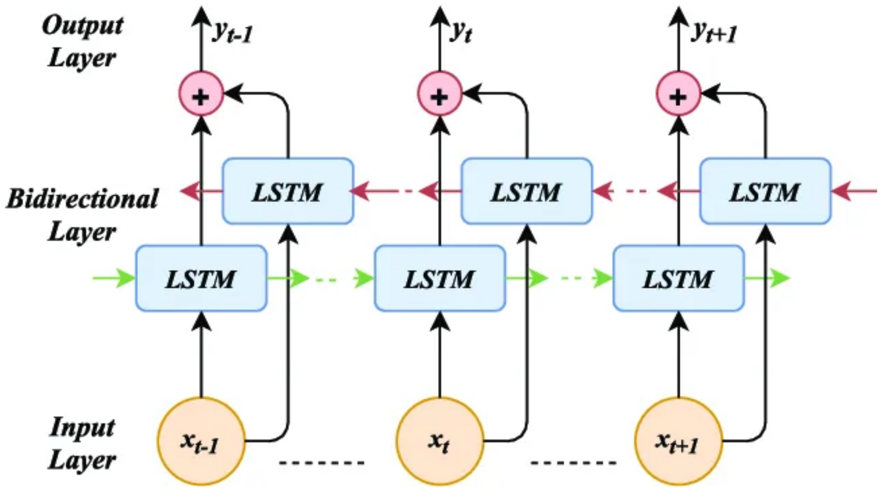
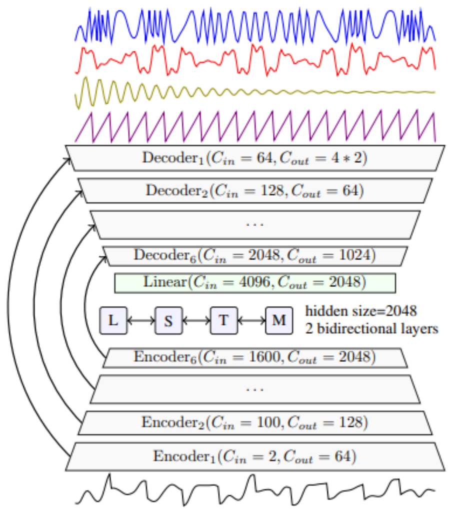
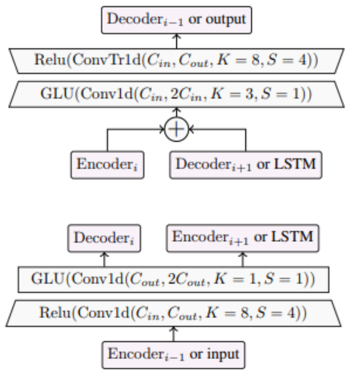


Figure 7: Bi-directional LSTM



Decoder$_1$(C_{in} = 64, C_{out} = 4 * 2)
Decoder$_2$(C_{in} = 128, C_{out} = 64)
...
Decoder$_6$(C_{in} = 2048, C_{out} = 1024)
Linear(C_{in} = 4096, C_{out} = 2048)
L
S
T
M
hidden size=2048
2 bidirectional layers
Encoder$_6$(C_{in} = 1600, C_{out} = 2048)
...
Encoder$_2$(C_{in} = 100, C_{out} = 128)
Encoder$_1$(C_{in} = 2, C_{out} = 64)

(a) Demucs architecture with the mixture waveform as input and the four sources estimates as output. Arrows represents U-Net connections.



Decoder$_{i-1}$ or output
Relu(ConvTr1d(C_{in}, C_{out}, $K = 8, S = 4$))
GLU(Conv1d(C_{in}, $2C_{in}$, $K = 3, S = 1$))
Encoder$_i$
Decoder$_{i+1}$ or LSTM
Decoder$_i$
Encoder$_{i+1}$ or LSTM
GLU(Conv1d(C_{out}, $2C_{out}$, $K = 1, S = 1$))
Relu(Conv1d(C_{in}, C_{out}, $K = 8, S = 4$))
Encoder$_{i-1}$ or input

(b) Detailed view of the layers Decoder$_i$ on the top and Encoder$_i$ on the bottom. Arrows represent connections to other parts of the model. For convolutions, C_in (resp C_out) is the number of input channels (resp output), K the kernel size and S the stride.

Figure 8: Left: Demucs architecture. Right: Encoder and decoder blocks

The input mixture signal is seen at the bottom. The wave signal is then sent up: first through the encoder block, then the BiLSTM and finally through the decoder blocks and the final linear layer. The 4 output sources are seen as waves at the top.

Both the encoder and decoder blocks are illustrated to the right in the figure above. The encoder consists of $L = 6$ blocks, and they have specified input C_{in} and output channels C_{out}. They each start with a convolution filter using kernel size $K = 8$ and stride $S = 4$ (downsampling by a factor of 4) with the specified C_{in} and C_{out} followed by a Relu function. Additionally each encoder block applies another convolution with $K = S = 1$ with input channels C_{out} (from first filter) and output channels $2C_{out}$ followed by a gated linear units (GLU) activation function. This activation function "gates" information flow in inputs, and it works similarly to the LSTM, in that it controls how much information is allowed through. It's defined as:

$$GLU(\mathbf{X}) = (\mathbf{X} * \mathbf{W} + \mathbf{b}) \otimes \sigma(\mathbf{X} * \mathbf{V} + \mathbf{c})$$

Where $\mathbf{X}$ is the input sequence, $\mathbf{W}$ and $\mathbf{V}$ are learnable parameter matrices used for linear projection, $\mathbf{b}$ and $\mathbf{c}$ are learnable biases and σ is the Sigmoid function. Finally $\otimes$ is the element-wise multiplication operation. This equation means, that the GLU learns two different linear projections and applies it to the input. It then multiplies each element from the first projection, with the sigmoid (ensuring $x_{ij} \in [0, 1]$) of the other projection. The network can therefore learn to control how much information is allowed to flow through the GLU function. In practice: Demucs doubles the input channels in the convolution layer, since GLU splits the input along the channel axis and takes half of the channels to use for the first projection (values to keep) and uses the second half of the channels for gating. The output of the GLU is then passed forward to two segments: the corresponding decoder block (skip-connection) and the next encoder block or the BiLSTM.

The BiLSTM is located between the encoder and decoder. It has 2 layers, and therefore doubles the output channels to $C_{out} = 4096$, which is then halved by a linear layer afterwards.

The decoder is essentially the reversed encoder just like in the Wave-U-Net model. There are also $L = 6$ blocks labeled in reverse order. Each decoder block takes as input the sum of the previous decoder output and corresponding encoder output. Firstly a convolution filter is applied with $K = 3$ and $S = 1$ with GLU activation function, and secondly a transposed convolution is applied with $K = 8$ and $S = 4$ (upsampling with a factor of 4) followed by a Relu function. Transposed convolution applies convolution in the opposite direction of regular convolution: it slides each input element across the kernel[13]. It essentially serves as an upsampling function. Decoder1 has output channels $C_{out} = S \cdot C_0$, where S is the number of sources and C_0 is the number of channels of the input mixture signal. The final S sources are then generated with a linear layer.

The loss function used to train the model is simply the Mean Absolute Error (they call it L_1) between the models estimates of the source signals $\hat{S_k}$ and the actual source signals S_k:

$$L_1\left(\hat{S}_k, S_k\right) = \frac{1}{T}\sum_{t=1}^{T} |\hat{S}_{k,t} - S_{k,t}|$$

The loss function goes through each time sample in the signal and computes the absolute difference between the two signals, and in the end averages this value with the number of time samples T. They train the model using the Adam optimizer with a learning rate of $3 \cdot 10^{-4}$.

3.3.3 Hybrid Demucs

Hybrid Demucs is the actual model architecture our selected model ”mdx_extra_q” uses. It extends regular Demucs by utilizing multi-domain analysis. It is compromised of a temporal branch (waveform signals in time-domain), a spectral branch (spectrogram signals in frequency-domain) and shared layers in the middle. This means the model uses information both from the wave signal and the spectrogram and can freely choose which parts to focus on. The architecture of the model is illustrated in figure 9. (taken from paper)[9]
The model resembles regular Demucs with the U-Net shape, except it has two parts (dual U-net structure). The input mixture signal is seen at the bottom right. The waveform signal is processed through the temporal branch just like in regular Demucs (blue blocks to the right) consisting of 5 encoders ($TEncoder_i$), a shared encoder/decoder layer in the middle (grey blocks) and finally 5 decoders ($TDecoder_i$).

Additionally, the input signal is transformed to a spectrogram in the time-frequency domain using Short-Time Fourier Transform (STFT) (bottom left in the figure). The spectrogram signal is computed such that the time dimension matches the temporal encoders output dimension ($\frac{T}{1024}$). The spectrogram is instead processed through the spectral branch (pink blocks to the left) ($ZEncoder_i$ and $ZDecoder_i$). Both branches uses skip-connections between encoders and decoders like previous architectures.

The output of the spectral encoder and temporal encoder is summed and processed through the shared layers. The output of the shared layers' decoder is processed respectively both through the spectral and temporal decoder. Finally the output of the spectral decoder is transformed back to waveform signals using Inverse STFT (top left) and is summed with the temporal encoders output to get the source signals.

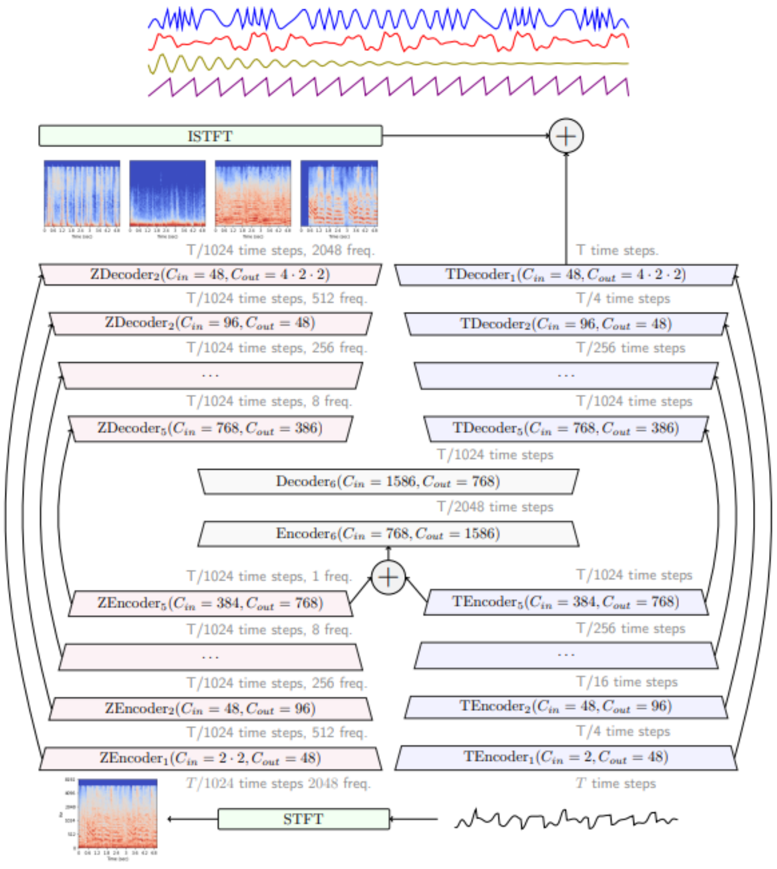

ISTFT
T/1024 time steps, 2048 freq.
T time steps.
ZDecoder$_2$($C_{in} = 48, C_{out} = 4 \cdot 2 \cdot 2$)
TDecoder$_1$($C_{in} = 48, C_{out} = 4 \cdot 2 \cdot 2$)
T/1024 time steps, 512 freq.
T/4 time steps
ZDecoder$_2$($C_{in} = 96, C_{out} = 48$)
TDecoder$_2$($C_{in} = 96, C_{out} = 48$)
T/1024 time steps, 256 freq.
T/256 time steps
...
...
T/1024 time steps, 8 freq.
T/1024 time steps
ZDecoder$_5$($C_{in} = 768, C_{out} = 386$)
TDecoder$_5$($C_{in} = 768, C_{out} = 386$)
T/1024 time steps
Decoder$_6$($C_{in} = 1586, C_{out} = 768$)
T/2048 time steps
Encoder$_6$($C_{in} = 768, C_{out} = 1586$)
T/1024 time steps, 1 freq.
T/1024 time steps
ZEncoder$_5$($C_{in} = 384, C_{out} = 768$)
TEncoder$_5$($C_{in} = 384, C_{out} = 768$)
T/1024 time steps, 8 freq.
T/256 time steps
...
...
T/1024 time steps, 256 freq.
T/16 time steps
ZEncoder$_2$($C_{in} = 48, C_{out} = 96$)
TEncoder$_2$($C_{in} = 48, C_{out} = 96$)
T/1024 time steps, 512 freq.
T/4 time steps
ZEncoder$_1$($C_{in} = 2 \cdot 2, C_{out} = 48$)
TEncoder$_1$($C_{in} = 2, C_{out} = 48$)
T/1024 time steps 2048 freq.
T time steps
STFT

Figure 9: Hybrid Demucs architecture

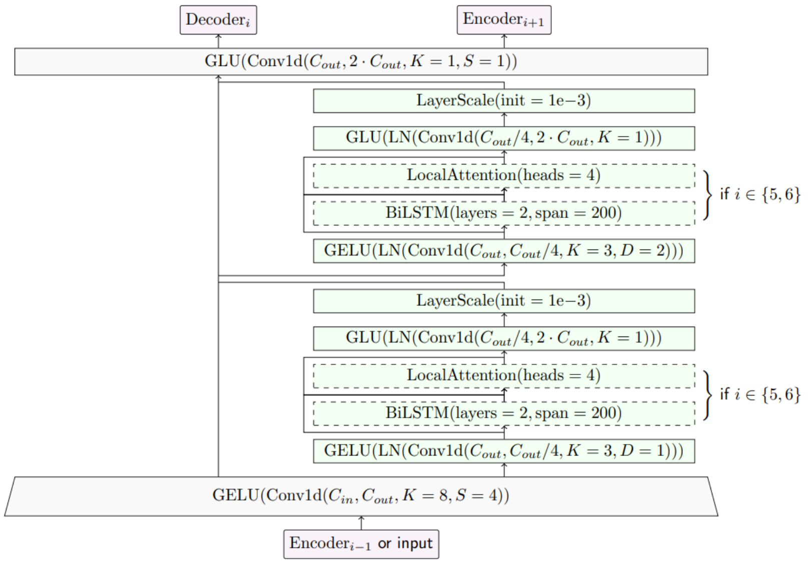
The difference between the temporal branch and spectral branch is, that the temporal applies convolution along the time-axes, while the spectral applies it along the frequency axis. Besides that, the encoder and decoder structure is similar between the branches.

The overall encoder and decoder structure resembles Demucs (figure 8 right). A small difference is, that Relu is replaced by Gaussian Error Linear Units (GELU) activation function. GELU is defined as:

$$GELU(x) = xP(X \leq x) = x\Phi(x)$$

Where X is the stochastic variable of the assumed distribution of x. This function assumes that x follows a standard normal distribution $X \sim \mathcal{N}(\mu = 0, \sigma = 1)$, which makes sense since neuron inputs tend to follow normal distributions. $\Phi(x)$ is the standard normal cumulative distribution function. Intuitively, this function scales each x to how much larger it is than the other inputs.

A bigger difference between the encoder structures is the introduction of compressed residual branches inserted between the two previously mentioned convolution layers used in regular Demucs. Figure 10 illustrates the content of each encoder block using these branches.[9]



Decoder$_i$
Encoder$_{i+1}$
GLU(Conv1d($C_{out}, 2 \cdot C_{out}, K = 1, S = 1$))
LayerScale(init = 1e−3)
GLU(LN(Conv1d($C_{out}/4, 2 \cdot C_{out}, K = 1$)))
LocalAttention(heads = 4)
BiLSTM(layers = 2, span = 200)
if $i \in \{5, 6\}$
GELU(LN(Conv1d($C_{out}, C_{out}/4, K = 3, D = 2$)))
LayerScale(init = 1e−3)
GLU(LN(Conv1d($C_{out}/4, 2 \cdot C_{out}, K = 1$)))
LocalAttention(heads = 4)
BiLSTM(layers = 2, span = 200)
if $i \in \{5, 6\}$
GELU(LN(Conv1d($C_{out}, C_{out}/4, K = 3, D = 1$)))
GELU(Conv1d($C_{in}, C_{out}, K = 8, S = 4$))
Encoder$_{i-1}$ or input

Figure 10: Encoder block from Hybrid Demucs

Hybrid Demucs introduces two compressed residual branches (green blocks), which are located between the two regular convolution layers. The two branches serve to capture additional information, and the outputs are simply added to the output of the first convolutional layer (indicated by the arrows). The first part of each branch is dilated convolution with $K = 3$ and $D = 1$ for the first branch (regular convolution without dilation) and $D = 2$ for the second branch. This is the only difference between the two branches. Dilated convolution skips elements in the input by inserting holes in the filter, which increases the receptive field of the filter and therefore captures a wider context window[14]. Then a Layer Normalization (LN) is applied, which normalizes the output with learnable parameters and finally passed through the GELU function. LN is defined as:

$$LN(x) = \frac{x - \mu}{\sigma}\gamma + \beta$$

Where μ is the empirical average and σ is the empirical standard deviation of each input x_i across feature dimensions (time, channel and frequency) in the layer that x_i belongs to:

$$\mu = \frac{1}{T}\sum_{i=1}^{T} x_i \quad \sigma = \sqrt{\frac{1}{T}\sum_{i=1}^{T}(x_i - \mu)^2}$$

γ and β are two learnable parameters. γ is called ”gain” and this scales the normalized value, while β is the offset bias.

For the innermost layers (5 and 6) a BiLSTM (like in regular Demucs) and Local Attention is used to capture long range context (skip connections are used indicated by the arrows). Local attention includes a penalty term for inputs far away from each other, which lowers the attention value if two values are far apart. It is defined as:

$$w_{i,j} = \text{softmax}\left(Q_i^T K_j - \sum_{k=1}^{4} k\beta_{i,k}|i-j|\right)$$

where $w_{i,j}$ is the attention weight between position i and j, Q_i and K_j are the respective query and key, $\beta_{i,k}$ is the output of a linear layer. The penalty term is $k\beta_{i,k}|i-j|$, which increases when i and j are far apart.

After the local attention a regular convolution with $K = 1$ is applied, and passed through LN again, and finally the GLU function is applied. The output of the residual branch is then scaled by a LayerScale function, which scales each channel with a learnable parameter, so the network can learn how much of the branch output is being used.

The decoder is built symmetrically, which means it essentially has the same structure as the encoder only reversed, also one of the outer gray layers are replaced by the transposed convolution to perform upscaling just like in regular Demucs. The loss function and procedure used to train the Hybrid Demucs is the same as for regular Demucs. The model we use: "mdx_extra_q" is trained on 150 music related audio tracks from the MUSDB HQ dataset.

3.4 Time-Stretching with Audiostretchy

Time-stretching is an important step in our voice conversion pipeline. It ensures that the translated audio matches the timing of the original English speech without changing the pitch. This is especially important for dubbing, where the mouth movements in the video should still align with the spoken audio in the new language.

To do this, we use the Python library `audiostretchy` [15], which implements **Time-Domain Harmonic Scaling (TDHS)**, which is a technique first described by David Malah [16], and later implemented in C by David Bryant [17]. Audiostretchy is a Python wrapper around Bryant's audio-stretch library, and allows us to modify the speed of speech while preserving pitch and keeping the speaker's voice natural.

Time-stretching means making an audio clip longer or shorter. For example, if someone says "hello" in 1.5 seconds, we might need to stretch or compress it to fit exactly into 1.2 seconds — but without making their voice sound deeper or higher. Traditional speed-change methods often change the pitch, but audiostretchy avoids this by working directly in the time domain.

TDHS works by using the repeating (periodic) nature of voiced speech. Voiced sounds, like vowels, have a clear pitch and harmonic structure, which means they repeat regularly over time. TDHS takes advantage of this by copying and aligning these repeating parts to make the speech faster or slower in time, without changing its pitch.
The TDHS algorithm works entirely in the time domain, which means it processes raw audio samples directly, without converting them to the frequency domain. The basic idea is to generate each new output sample by mixing together multiple past samples that are one pitch period apart.
Mathematically, each output sample $y(n)$ is calculated like this:

$$y(n) = \sum_{i=0}^{m-1} x(n-iN) \cdot h(iN)$$

Where: [16]

- $x(n)$ is the original input signal.
- N is the estimated pitch period in samples.
- m is the number of pitch periods used to make the output
- $h(iN)$ is a window function that controls how much each past sample contributes.
- $y(n)$ is the resulting output signal.

3.4.1 How Audiostretchy Works

Audio is made up of many small values called **samples**. At a sample rate of 44.1 kHz, there are 44,100 samples per second. To analyze audio efficiently, the algorithm groups samples into short overlapping chunks called **frames**.
By default, audiostretchy uses a frame size of 25 milliseconds, which corresponds to:

$$\text{Frame length} = \frac{25\,\text{ms}}{1000\,\text{ms/sec}} \times 44100\,\text{samples/sec} \approx 1102\,\text{samples}$$

Voiced speech (like vowels) contains a repeating pattern called the **pitch period**. This period is estimated using pitch detection. By default, the allowed range is:

$$\begin{aligned} \text{Minimum period} &= \frac{44100}{333} \approx 132 \quad \text{samples (max pitch: 333 Hz)} \\ \text{Maximum period} &= \frac{44100}{55} \approx 801 \quad \text{samples (min pitch: 55 Hz)} \end{aligned}$$

Audiostretchy's time-stretching algorithm involves three main stages:

- **Pitch Period Detection:** Each frame is analyzed to find its pitch period (how many samples it takes for the waveform to repeat).
- **Time-Scale Modification:** Based on the wanted stretch ratio, the algorithm modifies the signal in chunks of one or more pitch periods. It either inserts more cycles (for stretching) or removes some (for compression). It uses a custom linear cross-fade function called `merge_blocks` to blend waveforms.

  The formula for merging two segments A and B of length L samples is:

  $$\text{output}[i] = \frac{(L-i)\cdot A[i] + i \cdot B[i]}{L}, \quad i = 0, 1, ..., L-1$$

  This produces smooth transitions without audible clicks.
- **Silence Detection (optional):** The algorithm can detect silence based on energy. A frame is considered silent if its RMS (Root Mean Square) energy is below -40 dB. RMS is calculated as:

  $$\text{RMS} = \sqrt{\frac{1}{N}\sum_{t=1}^{N} x(t)^2}$$

  If a segment is silent, a different stretch ratio (called `gap_ratio`) can be applied to compress it more aggressively.

Overall, audiostretchy makes it possible to adjust the length of speech in a way that keeps the voice natural and clear, which is essential for dubbing where timing and realism are both important. In our implementation, we do not modify any of the internal parameters besides adding a minimum ratio of 0.75 and a maximum ratio of 1.25.

3.4.2 Reference Window in Our Method

In our pipeline, a **reference window** is applied during the voice conversion process to capture a localized representation of the speaker's voice. Specifically, we use a short local segment of the original English speech that includes what the speaker said just before and after the current utterance.
This reference window provides a local voice context, not for timing, but for capturing the speaker's voice characteristics (such as tone and speaking style) during voice conversion. It helps the system generate voices that better resemble the original speaker on a local level.

This is different from the method used in the baseline pipeline, where the entire speech track of the target speaker was used to convert each utterance. This approach aims towards a more global match, but can lead to monotone voices and missed local details. Our method focuses on short, sentence-sized windows, which leads to more precise alignment between the translated speech and the original video's speech.

By combining pitch-preserving, time-stretching and localized voice conversion, we are able to generate dubbed speech that sounds natural, is correctly timed, and closely matches the voice of the original speaker.

3.5 Minor Improvements

The **automatic speech recognition** (ASR) component for transcription was upgraded from Whisper large to Whisper large-v3, which retains the same architecture and parameter count (1.55B) but benefits from improved training data and model weights. Both the baseline and our version use the `stable-ts` wrapper, which adds configurable features for segment grouping and timestamp control. We enhanced this further by implementing a refined timestamp adjustment step, which iteratively mutes portions of the audio and analyzes token probabilities to more precisely align word boundaries; delaying start times and advancing end times when appropriate. The utterance grouping logic was also revised to produce more coherent segments, which should directly benefit downstream tasks like TTS and voice conversion by improving synchronization.

In addition to model-level improvements, the entire codebase was restructured to prioritize clarity and maintainability. The original baseline was implemented as a single monolithic Jupyter notebook with minimal documentation. We refactored the pipeline into a modular structure composed of well-defined, reusable functions, each documented with appropriate comments and docstrings. A comprehensive README was also added to support consistent setup and usage across environments.

To further support iterative development and experimentation, all intermediate outputs were saved in a structured `data/processed` directory rather than being handled through temporary files. This organization streamlined debugging, parameter tuning, and tracking of results across different stages of the pipeline. Together, these improvements enhance the overall usability of the system and make it better suited for future research extensions. Importantly, the entire pipeline was implemented and run at zero cost during the project, using freely available APIs free-tier credits, requiring no monetary expenditure throughout development. This is opposed to the baseline model, where each token processed for speaker diarization and regrouping by chat GPT-4o was paid for.

3.6 Evaluation

The use of well-defined and rigorous evaluation procedures is essential in any AI-driven system—particularly in multi-modal tasks like automatic dubbing, where multiple interdependent subsystems must work together seamlessly. Without standardized and interpretable metrics, it becomes difficult to identify system weaknesses, ensure reproducibility, or make meaningful comparisons between models. In our case, we aim to evaluate several core components of the updated dubbing pipeline: source separation, speaker diarization, pitch-preserving audio stretching, and the use of localized reference windows for voice conversion. Each of these stages introduces distinct challenges that must be assessed using targeted and appropriate evaluation criteria.

The original paper proposed three basic metrics intended to measure transcription accuracy, translation accuracy, and diarization accuracy, alongside a subjective "Personal Rating" score on a scale from 1 to 10. However, these metrics were poorly defined, subject to personal bias and implemented using simplistic accuracy checks, without detailed criteria for correctness. For instance, transcription and diarization accuracy were calculated using basic predicate functions that returned binary correctness per segment, ignoring partial matches, timing misalignments, speaker turn boundaries, and duration weighting. This naive approach led to inflated accuracy figures that did not reflect true system performance. Also, translation quality has been used as a key evaluation metric. However, we do not consider this approach suitable for our purposes, since translation allows for multiple valid outputs, which makes it extremely difficult to define a reliable ground truth.

To address these shortcomings, we adopt a more principled evaluation strategy, grounded in standardized metrics and detailed qualitative procedures. This approach allows for a more accurate understanding of performance across individual components, and supports fair and reproducible model comparisons.

3.6.1 Evaluating the Speaker Diarization System

One of the ways we evaluate the performance of the speaker diarization system is by measuring the **Diarization Error Rate (DER)**, a widely adopted metric for quantifying diarization quality. DER measures the total proportion of time that the system's output deviates from the reference annotation, relative to the total duration of speech in the ground truth. DER consists of three types of errors:

- **False Alarm (FA)**: Non-speech segments incorrectly classified as speech.
- **Missed Detection (MD)**: Speech segments not detected by the system.
- **Speaker Confusion (SC)**: Speech segments attributed to the wrong speaker.

These components are summed and normalized by the total duration of ground-truth speech:

$$\mathrm{DER} = \frac{\mathrm{FA} + \mathrm{MD} + \mathrm{SC}}{\text{Ground truth speech duration}}$$

To account for the imprecision inherent in manual annotations, a collar tolerance of 250 ms is applied before and after each speaker turn boundary. This 500 ms window is excluded from evaluation, meaning that minor misalignments within this collar do not count as errors. This is standard practice in diarization evaluation and helps avoid over-penalizing small timing discrepancies between the ground-truth annotations and system output [7].

Another important metric for evaluating a speaker diarization system is **speaker count error**. A speaker count error occurs when the model fails to correctly identify the number of distinct speakers in an audio recording. For instance, if an audio clip had three individual speakers but the diarization system detected any number other than three, this would constitute a speaker count error. Accurately estimating the number of speakers directly improves diarization accuracy, and in the context of auto-dubbing, it is crucial for creating correct reference audio files. An incorrect speaker count not only inflates the DER, particularly the speaker confusion component, but also disrupts the voice conversion system, leading to degraded dubbing quality.

The final metric we use is the **concatenated minimum-permutation word error rate (cpWER)**. This metric extends the traditional **word error rate (WER)** to evaluate speaker-attributed transcripts by assessing how accurately the words are assigned to the correct speakers. It works by first grouping all utterances per speaker, both in the system output and the ground truth transcript. WER is then computed between these per-speaker concatenations. An error is counted not only when a word is misrecognized, but also when a correctly transcribed word is assigned to the wrong speaker. In this way, cpWER acts as a stricter measure than WER, serving as a lower bound on the combined transcription and speaker attribution error.

To compute cpWER, the system must first correctly determine the number of speakers in the audio. If the predicted speaker count does not match the ground truth, the metric becomes undefined, meaning its more limited than DER. However, cpWER is especially important in the context of automated dubbing because it captures the combined effect of transcription accuracy and speaker attribution. Even if the DER is low, a misalignment between who said what—reflected in a high cpWER—can result in unnatural or confusing dubbed output, particularly when voice conversion is speaker-specific.

Overall, DER, speaker count error, and cpWER provide a complementary set of metrics for evaluating speaker diarization systems. DER offers a detailed breakdown of timing and attribution errors, speaker count error ensures the correct identification of participants, and cpWER captures the practical impact of diarization quality on speech content. Together, these metrics enable a robust and comprehensive assessment of diarization performance in the context of our automated dubbing pipeline, and allow comparison to existing diarization benchmarks.

3.6.2 Evaluation of source separation

Evaluating the performance of source separation quantitatively is difficult, since you would ideally need ground-truth audio signals of the speech and background source. As we cannot acquire those, we chose to evaluate qualitatively. We also decided to qualitatively evaluate the voice conversion, since the user experience is a natural way to evaluate the compatibility of voices. We still wanted some metric for the performance of these two parts of our model, so we designed a test to acquire a **Mean Opinion Score (MOS)**. We did not test the baseline model for the performance of source separation, since it does not include background audio at all, and therefore would not make sense to evaluate. These are the steps that we used in the test:

- 1. Give context to the project and test: "We are researching how automatic dubbing can be used to translate and recreate speech in videos in other languages. You will hear an original video clip and two versions of the same clip generated by two different models."
- 2. Show the participant a short clip of the original input video
- 3. Show the same clip of our models video output. The participant will now be asked to evaluate different metrics by giving a rating from 1-10.
- 4. Ask the participant to evaluate how well background and speech audio are mixed together. In this case a rating of 1 means all background and speech audio is missing, and 10 means all the

background audio is identical to the original, and all the speech audio is included. (**Background**) Additionally ask the participant to evaluate how well the translated dubbed voices match the tone and pitch of the original voice. In this case a rating of 1 means a monotone robot voice with no correlation to the original voice, and 10 means identical tone and pitch to the original voice. (**VC (Our)**)

- 5. Show the clip of the baseline models video output. Ask the participant to evaluate the last metric same way as before: matching of tone and pitch. (**VC (B)**)
- 6. Finally ask the participant to evaluate the overall experience of the two dubbed versions of the clip by giving a rating from 1-10. (**Overall (Our)**, **Overall (B)**)

Each metric is named according to the name in bold in the parenthesis above. These names are used in the result table 3. To keep it simple we decided to perform the test on the first 7 videos, since we already had access to these output videos from the baseline. We decided to show clips of the first 30 seconds of each video, because we felt this was enough to get an overall sense of each of the metrics we researched. We tested 9 participants in total including ourselves. We tried as best we could to remain unbiased when performing the test on ourselves.

4 Results

Table 2 summarizes the performance comparison between the proposed model and the baseline in terms of diarization and transcription accuracy. We report the global Diarization Error Rate (DER), average DER, and the average rates of false alarm (FA), missed detection (MD), and speaker confusion (SC), along with the average concatenated minimum-Permutation word error rate (cpWER) for videos with matched speaker count. It's important to note that the global DER is weighted by the duration of each clip, making it more representative of overall performance, while average DER gives equal weight to all videos regardless of their length.

Table 2: Comparison of the proposed model and baseline in terms of diarization and word error metrics.

Model	Global DER	Avg. DER	Avg. FA	Avg. MD	Avg. SC	Avg. cpWER
Proposed	36.6%	39.6% ± 6.2%	12.3% ± 4.5%	11.0% ± 3.8%	16.2% ± 5.9%	12.2% ± 7.0%
Baseline	45.3%	46.5% ± 6.7%	13.4% ± 5.1%	17.0% ± 6.2%	16.1% ± 6.0%	24.6% ± 16.3%

Overall, both models perform similarly in terms of speaker confusion. The proposed model achieves a modest improvement in false alarm rate but demonstrates a more substantial reduction in missed detections. Here we see an improved rate by 6.01% compared to the baseline, which is the main reason behind the lower DER.

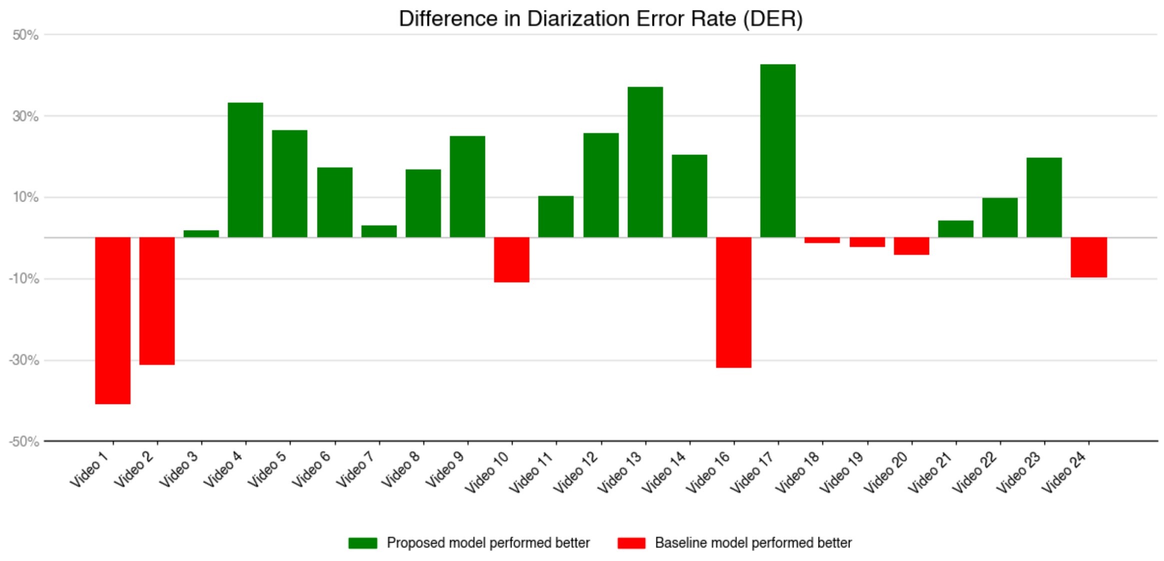


Figure 11: Difference in diarization error rate (DER) between the proposed model and the baseline model

Figure 11 illustrates the difference in Diarization Error Rate (DER) between the proposed model and the baseline model for each video. For a more detailed breakdown of the per-video results, please refer to the full table provided in the Appendix. Overall, the proposed model outperforms the baseline, though there are notable exceptions particularly videos 1, 2, and 16, where the baseline achieves significantly better results.

A key observation is related to video 1, which contains a clip from Charlie Chaplin's speech in *The Great Dictator*. During testing, we found that ChatGPT was able to recognize the source material and leverage its prior knowledge of the speech to perform speaker diarization with unusually high accuracy. This recognition largely accounts for the performance disparity in that case. It is also worth noting that video 15, a clip from the Adult Swim comedy show *Loiter Squad*, was excluded from the test set. ChatGPT declined to perform speaker diarization on the transcript, citing a violation of its usage policies.

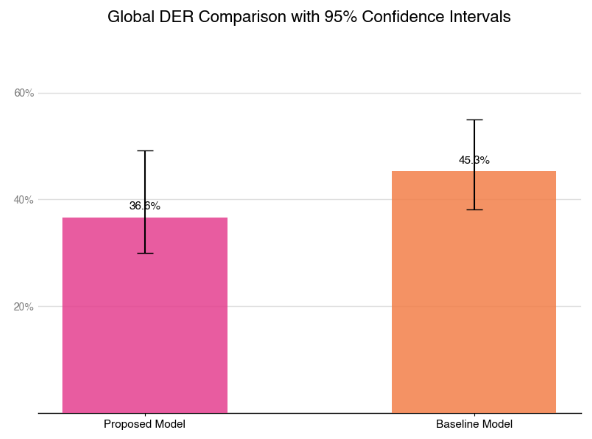


Figure 12: Global diarization error rate (DER) with 95% confidence intervals between the proposed model and the baseline model

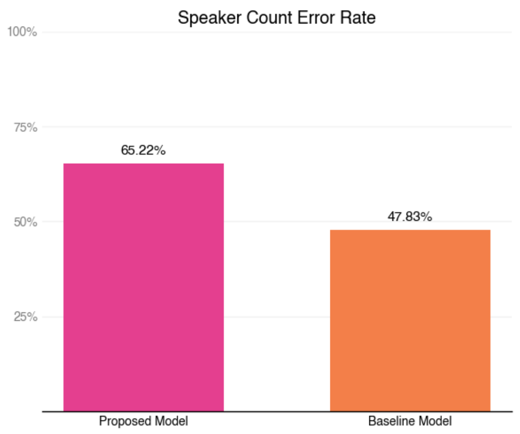


Figure 13: Speaker count error rate between the proposed model and the baseline model

Figure 12 presents the mean Diarization Error Rate (DER) for our enhanced pipeline (36.5%) versus the baseline (45.3%), along with 95% confidence intervals that show an overlap. This overlap implies that, while our model tends to make fewer speaker-labeling mistakes, we would need additional data to conclude statistical significance with confidence. This is supported by a paired t-test on DER scores, which yielded a p-value of 0.1453, indicating that the observed improvement is not statistically significant at the 0.01 level and may be due to random variance.

Figure 13 illustrates the speaker count error rate showing the proportion of videos where the speaker count was misidentified. Here, the proposed model falls short of the baseline model with an error rate of 65.2% compared to 47.8%.

Similar to Figure 11, which illustrates the difference in DER, Figure 14 presents the difference in cpWER between the two models. However, as noted in the Methods section, cpWER can only be computed when the model correctly predicts the number of speakers in a given video, matching the ground truth. Due to the low speaker count accuracy observed in Figure 13, we were only able to compute cpWER for seven videos where both models correctly identified the number of speakers. Although the resulting sample size is limited, the proposed model still outperforms the baseline model in most of these cases. On average, the baseline cpWER was 24.55%, while the main model achieved a mean of 12.22%. A paired t-test yielded a p-value of 0.1254, indicating that the improvement was not statistically significant at the 0.01 level.

Table 3 summarizes the results of the qualitative user test of the source separation module and up-

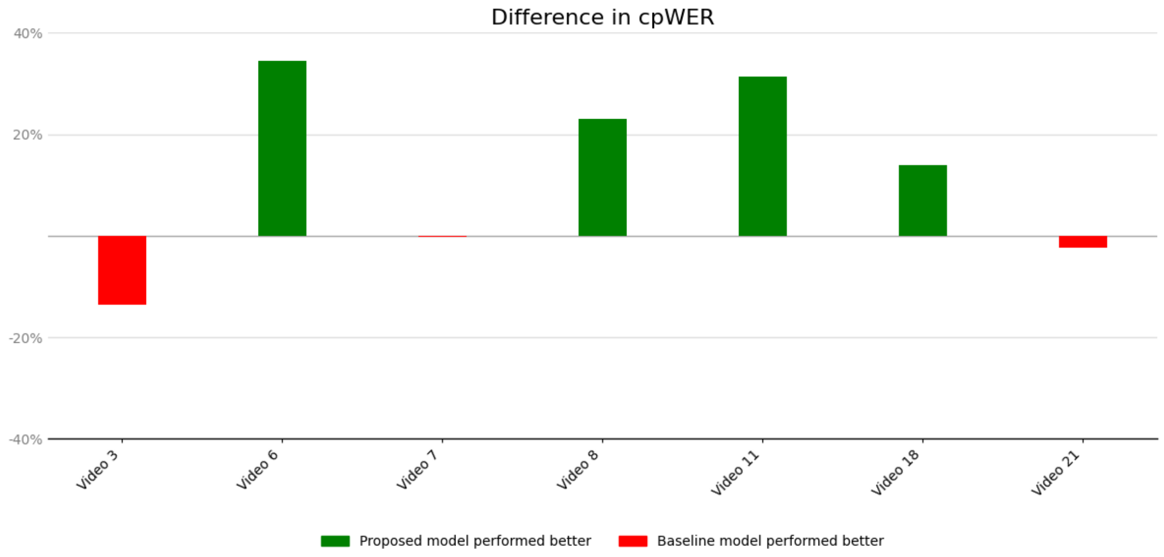


Figure 14: Difference in concatenated minimum-permutation word error rate (cpWER) between the proposed model and the baseline model

dated voice conversion system. Each row represents a participant, and each element in the table is the averaged rating across the 7 videos for that specific metric. For the proposed model, it reports a MOS of 7.28 for the background metric. We can't compare this to the baseline, but the value is satisfactory.

The proposed model performs better than the baseline model in both voice conversion and overall experience. We get a MOS of 6.62 for voice conversion, which is 1.19 higher than the baseline. For the overall experience, users gave an MOS of 6.95 for our model, which is 2.49 higher than the baseline.

We performed two paired t-tests between our model and the baseline to establish significant statistical difference in MOS. First test was for the voice conversion averaged scores (**VC (Our) vs. Vc (B)**), and second test was for the overall experience (**Overall (Our) vs. Overall (B)**). A paired t-test is used, since both samples in each test are from the same participants and therefore each observation is directly related. We test the null-hypothesis, that states samples have the same average (MOS), and obtain the following p-values:

$$p_1 = 0.004$$

$$p_2 = 0.0003$$

Where p_1 is the p-value for the first test, and p_2 is for the second test. With a significance level of $\alpha = 0.01$, we can reject the null hypothesis in both tests, since $p_2 < p_1 < 0.01$. We can therefore conclude statistical significant difference both in voice conversion and overall experience between the two models.

Participant	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	7.29	6.71	6.86	6.14	5.43
2	6.71	6.71	6.86	4.00	3.14
3	7.00	6.29	7.57	5.00	4.00
4	8.00	7.29	7.71	5.14	4.14
5	7.86	6.57	6.86	5.29	4.43
6	7.00	6.43	6.14	5.43	3.57
7	7.29	6.29	6.71	5.29	4.00
8	7.71	7.43	7.71	7.86	7.86
9	6.71	5.86	6.14	4.71	3.71
MOS	7.28	6.62	6.95	5.43	4.46

Table 3: Result table from user tests

5 Disscusion

5.1 Speaker Diarization and Transcription Accuracy

The results presented in Section 4 show that our proposed model outperforms the baseline in both diarization accuracy and transcription quality across the majority of cases in our extended test dataset. This is most clearly reflected in the improvements in DER and cpWER, where the proposed model achieves a substantial reduction in missed detections and nearly halves the cpWER compared to the baseline. These quantitative gains are further supported by our qualitative evaluation: participants generally rated the proposed model higher in both voice conversion quality and overall experience. Furthermore, a statistically significant improvement in the proposed model was found in these two metrics using the MOS scores. This suggests that improved diarization contributes not only to more accurate transcription but also to a more coherent and engaging viewer experience.

Despite these improvements, certain limitations affect the overall performance of the system. In particular, speaker count estimation remains a significant challenge. The proposed model exhibited a notably higher speaker count error rate than the baseline, which in turn impacted the reliability of the cpWER metric. As discussed in the Methods section, cpWER can only be computed for videos where the predicted number of speakers matches the ground truth. Consequently, the high speaker count error substantially reduced the number of valid test cases, limiting the generalization of the cpWER results.

In light of these results, it is insightful to compare our system's performance – built on AssemblyAI's diarization model – with the performance metrics reported by AssemblyAI themselves. In a blog post from 2024, AssemblyAI claims to achieve a DER of 28%, a cpWER of 36%, and a speaker error rate of just 2.9%[5]. In contrast, our evaluations yield a higher DER and a substantially greater speaker count error rate. This discrepancy likely stems from differences in the evaluation data. AssemblyAI does not specify which test set was used for their reported metrics, but similar evaluations of diarization systems often rely on the CallHome dataset. This dataset is known for its relatively clean, telephone-based audio and two-speaker conversations, conditions under which diarization systems typically perform well.

In contrast, our extended test set was deliberately curated to include more challenging conditions. These include multi-scene, multi-speaker videos, recordings with overlapping background noise such as music or crowd chatter, and instances of atypical or emotionally expressive speech. It is therefore reasonable to attribute the increase in DER to the added complexity of our dataset. However, the substantial rise in speaker count error was less anticipated and indicates that even moderately difficult conditions can significantly impair the model's ability to determine the correct number of speakers. This highlights a key area for improvement, especially in real-world applications where clean, controlled audio is not guaranteed.

5.2 Speech Extraction and Mixing

Evaluating the effectiveness of a source separation module is inherently more difficult than assessing transcription or diarization performance, as we lack access to ground truth isolated stems for comparison. Despite this, we conducted a qualitative evaluation in which nine participants rated the quality of the background audio mixing on a Mean Opinion Score scale from 1 to 10, as found in the results section. A score of 10 indicated a perfect background mix with no missing vocal elements, while a score of 1 indicated complete vocal loss and no background audio. Across 63 samples (derived from the first 30 seconds of seven videos), the proposed system received an average background MOS of 7.29. Notably, the baseline system did not incorporate any source separation or background mixing, instead using raw dubbed speech without reintegration of background audio. We hypothesize that this was the primary reason for the significant increase in overall enjoyment ratings, from 4.70 in the baseline to 6.95 in our proposed model. The inclusion of background ambiance, even if imperfectly reconstructed, appears to enhance immersion and perceived quality.

However, implementing the source separation module was not without challenges. A key issue arose from Demucs' handling of non-speech vocalizations such as laughter, grunts, and gasps, which were correctly categorized as vocals by the model. As described in Section 3.3, Demucs is designed for music source separation and thus aims to preserve all vocal content. While technically correct, this behavior posed a problem: our transcription model ignores non-verbal vocalizations entirely, leading to instances where vocal audio was muted during laughter or expressive sounds, resulting in silences or mismatched mouth movement in the output final video. In this sense, we were, paradoxically, ”suffering from success.”

To address this, we explored several strategies with our iterative workflow and experimental folder setup described in Section 3.5. One approach involved selectively mixing the original audio with the separated background track based on utterance boundaries, allowing us to preserve non-verbal vocalizations during transcript-defined silent segments. Another strategy involved experimenting with different VAD configurations and preprocessing techniques to distinguish speech from non-speech vocalizations before passing the audio to Demucs. While both approaches showed promise, occasionally improving perceived continuity and naturalness, they also introduced inconsistencies. Most modifications resulted in skewed outcomes, with some videos benefiting from the changes while others suffered. For example, remixing the original audio with the separated background could lead to a cluttered or confusing soundscape when utterance boundaries were even slightly misaligned, as overlapping speech from the original audio would interfere with the voice-converted output.

Given this variability and the lack of consistent improvement, we ultimately chose the simpler and more predictable approach used in our final pipeline: to exclude non-speech vocalizations entirely. This prioritizes clarity and robustness over emotional fidelity. While this decision involves a clear trade-off, we believe it leads to a more reliable and coherent user experience. That said, this area could be further researched. A more advanced solution could involve training a custom VAD or classifier capable of distinguishing speech from other meaningful vocalizations (e.g., laughter, sighs), thereby enabling more intelligent and context-aware mixing decisions.

Another design choice in our pipeline was to use the isolated vocal track produced by Demucs as input to the speaker diarization and transcription modules, whereas the baseline used the original unseparated audio. We hypothesize that this preprocessing step contributed to improved downstream performance, as background noise and competing audio sources were removed in advance. While we did not quantify this effect directly, if more time was available to us the evaluation could have been expanded with a controlled evaluation comparing diarization and transcription accuracy on raw versus separated vocal tracks to assess the benefit statistically.

Overall, speech extraction using Demucs enabled a more immersive dubbed experience and likely contributed to improved subjective ratings. While some challenges remain, especially around handling non-verbal vocalizations, our results suggest that source separation is a viable and valuable component in an automated dubbing pipeline.

5.3 Voice conversion

In relation to our third research question, our findings indicate that both audiostretchy and reference windows lead to significant improvements in the quality and realism of the dubbed speech.

By applying audiostretchy, we were able to adjust the duration of each utterance to match the timing of the original video, without altering the pitch of the voice. This contributed to a more natural-sounding speech. The baseline system used an internal and less transparent length parameter in the voice conversion model.

In addition, the use of localized reference windows, where we use nearby utterances from the same speaker instead of all speech across the video, allowed the voice conversion model to capture more context-specific vocal features. This includes tone, energy, and emotions that are often lost when using a global speaker reference as the baseline model did. As a result, the synthesized voices more closely resembled the original speakers.

These improvements were reflected in our user evaluation, where participants rated how well the converted voices matched the original in terms of tone and pitch. Our proposed model achieved a Mean Opinion Score (MOS) of 6.62, compared to 5.43 for the baseline. A paired t-test confirmed that this difference was statistically significant ($p = 0.004$), supporting the conclusion that our approach leads to a better listening experience

Overall, the combination of pitch-preserving time-stretching and localized speaker references results in more natural and expressive speech, and represents a clear improvement over the baseline method.

5.4 Broader Context

Beyond the technical improvements demonstrated, our system also aligns with broader principles of responsible and trustworthy AI. In designing our pipeline, we prioritized transparency, reproducibility, and modularity, enabling easier auditing, iterative experimentation, and controlled studies. By isolating individual improvements, such as voice conversion, diarization, and background mixing, we allow for clearer attribution of gains and make it easier for future work to validate, extend, or adapt this contribution.

In relation to the United Nations' **Sustainable Development Goals (SDGs)**, our system contributes most directly to SDG 4 and SDG 10: Quality Education and Reduced Inequalities. By enabling high-quality dubbing with minimal manual intervention, the system supports greater accessibility of audiovisual content across languages and dialects. This is particularly relevant for education and cultural media, where accurate and natural-sounding multilingual content can help bridge global knowledge gaps.

At the same time, these advances also raise important ethical and economic concerns. As automated dubbing technologies become more accessible and cost-effective, there is a risk that they could displace human voice actors, translators, and dubbing professionals, particularly in lower, budget productions. While automation can democratize content creation, it may also undermine creative industries that rely on skilled labor and artistic interpretation. This highlights the need for thoughtful deployment strategies that consider the broader impacts of such tools. Potential responses could include hybrid workflows that combine automated pipelines with human oversight or quality assurance, ensuring that the creative expertise of professionals remains part of the process.

5.5 Limitations and Future Work

While our system demonstrates clear improvements over the baseline, there are several limitations that should be considered when interpreting the results. First, the user test included only nine participants, some of whom were the authors themselves. This small and possibly biased sample makes it difficult to draw strong conclusions from the MOS scores. Even though the difference between our model and the baseline was statistically significant, a larger and more diverse group of participants would provide more reliable and general results. Future tests should include users from different backgrounds, ages, and listening environments.

Second, our evaluation was mostly based on short video clips (typically less than five minutes) taken primarily from movies and dialogue scenes. Future work should test the pipeline on longer videos to better understand how it performs in real-world situations.

Third, we sometimes observed problems in the voice conversion, especially when the reference windows were too short or when the speaker changed tone or emotion quickly. In these cases, the model sometimes produced unstable or noisy speech. This suggests that the reference window method could be improved by adapting the window size to the situation. This could help maintain consistent speaker identity and naturalness further.

Our current system is also not designed for real-time use. Running the full pipeline still takes significant processing time and may not be suitable for live applications. Future versions of the pipeline could explore faster models.

Finally, our research focuses only on English as the input language and Danish as the output. It is not yet clear how well the system will work for other language pairs. Further testing is needed to confirm whether our improvements generalize to other languages.

6 Conclusion

The focus of this report was on improving an existing baseline pipeline for automatically dubbing videos with the use of AI tools. This problem was researched by focusing on three key components in the pipeline. Speaker diarization is the first area, that was researched, where AssemblyAI's audio based diarization was replaced with the text based "GPT-4o". We obtained lower average DER and cpWER, but higher Speaker Count Error. A paired t-test was performed, but resulted in no significant difference in DER and cpWER. The second research area was source separation, and how this would improve the final mixing of speech and background audio. We implemented a Hybrid Demucs model to obtain background and audio tracks. We tested the performance of the final mixing of background and speech, and obtained a MOS of 7.28. The third and final area of research regarded a different time-stretching and reference audio method, and how this would improve the voice conversion. In terms of this, we obtained a MOS of 6.62 for the quality of tone and pitch preservation, which was 1.19 higher than baseline. The overall experience of a dubbed video is an important metric, and is relevant for all three research areas. In terms of this, we obtained a MOS of 6.95, which is 2.49 higher than baseline. A paired t-test showed a significant improvement in the quality of voice conversion and overall experience. The proposed model can be used to improve quality of education and lower inequality by utilizing the model in areas with linguistic challenges. The proposed pipeline showed improvement on most metrics, however there is still room for further improvement. Speaker diarization, especially, is an area, that can be improved on. Our model obtained a high Speaker Count Error, which significantly lowered performance for diarization purposes. This lack in performance served as a bottleneck for our model, but could potentially be improved by researching better ways to determine number of speakers.

Appendix

Table 4: Per-video results for the proposed model (Main).

Video	**DER**	**Conf.**	**Miss**	**FA**	**WER**	**cpWER**
video 1	55.51%	46.43%	6.72%	2.36%	2.61%	-
video 2	58.55%	25.18%	11.16%	22.20%	0.00%	-
video 3	18.14%	3.34%	11.95%	2.85%	15.21%	24.83%
video 4	15.95%	3.69%	9.77%	2.49%	14.48%	22.54%
video 5	16.70%	11.79%	4.10%	0.80%	0.00%	-
video 6	22.36%	0.09%	8.89%	13.38%	5.05%	7.09%
video 7	8.94%	0.00%	2.98%	5.95%	3.29%	4.78%
video 8	18.29%	4.52%	6.68%	7.09%	5.47%	19.52%
video 9	30.30%	9.02%	11.85%	9.43%	14.63%	-
video 10	54.60%	34.02%	9.98%	10.60%	4.92%	-
video 11	6.54%	0.07%	4.67%	1.80%	11.44%	13.00%
video 12	53.92%	30.78%	11.46%	11.67%	20.65%	-
video 13	37.78%	10.37%	12.69%	14.72%	15.70%	-
video 14	17.26%	6.48%	4.10%	6.68%	2.25%	-
video 15	42.52%	22.53%	8.11%	11.87%	18.22%	-
video 16	93.81%	32.58%	19.05%	42.17%	17.65%	-
video 17	22.49%	3.71%	11.29%	7.49%	6.85%	-
video 18	20.97%	0.77%	12.76%	7.44%	3.41%	5.28%
video 19	57.63%	34.94%	9.39%	13.30%	21.17%	-
video 20	69.97%	12.93%	10.30%	46.74%	3.88%	-
video 21	27.93%	0.38%	16.87%	10.68%	10.71%	11.03%
video 22	90.96%	48.86%	25.36%	16.75%	25.56%	-
video 23	27.50%	3.81%	9.56%	14.13%	11.31%	-
video 24	84.35%	49.55%	21.62%	13.18%	28.35%	-
Average	39.71%	16.49%	10.89%	12.32%	10.95%	13.51%

Table 5: Per-video results for the baseline model.

Video	**DER**	**Conf.**	**Miss**	**FA**	**WER**	**cpWER**
video 1	14.48%	0.00%	9.74%	4.74%	3.26%	3.26%
video 2	27.25%	0.00%	16.89%	10.37%	13.16%	23.95%
video 3	19.82%	3.48%	8.61%	7.72%	1.52%	11.23%
video 4	49.15%	40.88%	4.29%	3.98%	14.95%	-
video 5	43.19%	30.38%	8.53%	4.28%	1.15%	47.46%
video 6	39.52%	9.79%	11.07%	18.66%	6.06%	41.57%
video 7	11.96%	0.27%	6.10%	5.59%	3.29%	4.51%
video 8	35.07%	15.17%	5.24%	14.66%	6.53%	42.47%
video 9	55.16%	29.10%	17.12%	8.94%	15.45%	-
video 10	43.54%	0.00%	11.75%	31.79%	2.27%	3.32%
video 11	16.70%	10.76%	4.01%	1.93%	11.82%	44.37%
video 12	79.61%	43.74%	22.13%	13.73%	27.71%	-
video 13	74.78%	38.66%	15.80%	20.32%	18.84%	-
video 14	37.75%	29.32%	5.99%	2.44%	8.43%	-
video 16	61.72%	4.34%	47.09%	10.29%	35.29%	-
video 17	65.08%	42.60%	8.01%	14.47%	9.63%	-
video 18	19.63%	5.25%	3.62%	10.76%	3.72%	19.13%
video 19	55.30%	24.38%	16.08%	14.85%	14.11%	-
video 20	65.76%	5.10%	19.71%	40.95%	9.91%	26.56%
video 21	32.18%	0.00%	15.14%	17.04%	7.50%	8.59%
video 22	100.74%	0.00%	97.18%	3.56%	100.00%	-
video 23	47.08%	12.58%	12.79%	21.71%	11.31%	-
video 24	74.62%	25.24%	24.53%	24.85%	29.66%	-
Average	46.53%	16.13%	17.02%	13.38%	15.46%	23.04%

Video	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	9	8	8	7	6
2	8	6	7	4	3
3	6	5	6	7	6
4	8	6	7	8	8
5	7	7	7	5	5
6	5	8	6	4	3
7	8	7	7	8	7
MOS	7.29	6.71	6.86	6.14	5.43

Table 6: Result table from user test for participant 1

Video	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	2	4	3	3	1
2	10	8	9	5	3
3	8	8	8	4	3
4	4	5	5	5	6
5	7	7	7	5	5
6	9	9	9	3	2
7	7	6	7	3	2
MOS	6.71	6.71	6.86	4.00	3.14

Table 7: Result table from user test for participant 2

Video	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	3	9	7	4	4
2	7	8	8	2	2
3	8	1	6	3	3
4	8	8	8	8	8
5	8	8	8	3	3
6	7	6	8	8	4
7	8	4	8	7	4
MOS	7.00	6.29	7.57	5.00	4.00

Table 8: Result table from user test for participant 3

Video	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	8	8	8	7	6
2	9	8	9	4	2
3	7	5	6	6	4
4	9	7	9	7	7
5	8	7	8	4	3
6	6	8	6	3	1
7	9	8	8	5	6
MOS	8.00	7.29	7.71	5.14	4.14

Table 9: Result table from user test for participant 4

Video	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	7	7	7	6	6
2	9	8	8	6	3
3	9	6	7	6	4
4	7	7	7	7	7
5	9	7	7	4	4
6	7	5	6	5	2
7	7	6	6	5	5
MOS	7.86	6.57	6.86	5.29	4.43

Table 10: Result table from user test for participant 5

Video	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	7	7	7	6	5
2	10	8	8	6	2
3	7	5	5	6	4
4	6	5	5	5	3
5	6	7	6	5	4
6	8	7	6	5	3
7	5	6	5	6	6
MOS	7.00	6.43	6.14	5.43	3.57

Table 11: Result table from user test for participant 6

Video	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	6	7	7	6	4
2	9	6	8	3	2
3	7	4	5	6	4
4	7	6	6	7	6
5	8	7	7	5	4
6	6	8	7	3	2
7	8	6	7	7	6
MOS	7.29	6.29	6.71	5.29	4.00

Table 12: Result table from user test for participant 7

Video	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	7	8	8	6	7
2	9	8	9	9	8
3	8	6	7	9	9
4	8	8	8	9	9
5	8	7	8	8	8
6	6	8	7	6	7
7	8	7	7	8	7
MOS	7.71	7.43	7.71	7.86	7.86

Table 13: Result table from user test for participant 8

Video	**Background**	**VC (Our)**	**Overall (Our)**	**VC (B)**	**Overall (B)**
1	7	2	3	4	4
2	9	7	8	6	2
3	5	4	4	3	3
4	8	7	8	5	5
5	6	7	7	5	5
6	7	9	8	6	4
7	5	5	5	4	3
MOS	6.71	5.86	6.14	4.71	3.71

Table 14: Result table from user test for participant 9

References

[1] Muneer Kayali, "autodubbing." https://github.com/muneer-kayali/autodubbing, 2024.

[2] T. R. B. William Kock-Andersen and L. M. Bjerg, "auto-dubbing." https://github.com/tob-euro/auto-dubbing, 2025.

[3] T. R. B. William Kock-Andersen and L. M. Bjerg, "Shared resources folder." https://drive.google.com/drive/folders/1P9Mm6eVvY-cbSdHFB8JIM5WKkQtZ8xpf, 2025. Google Drive folder containing dataset.

[4] A. D. (Meta), "demucs." https://github.com/facebookresearch/demucs?tab=readme-ov-file, 2025.

[5] AssemblyAI, "Speaker diarization improvements: Better speaker embeddings, clustering, and vad." https://www.assemblyai.com/blog/speaker-diarization-improvements, 2023.

[6] L. Wan, Q. Wang, A. Papir, and I. L. Moreno, "Generalized end-to-end loss for speaker verification," *arXiv preprint arXiv:1710.10467*, 2018.

[7] Q. Wang, C. Jia, Y. Wu, W. Zhang, M. Chen, J. Zhan, Y. Liu, J. Li, L. Si, and L. Xie, "Deep speaker embedding learning with multi-level pooling for text-independent speaker verification," *arXiv preprint arXiv:1710.10468*, 2018.

[8] AssemblyAI, "Speaker diarization: Speaker labels for mono channel files." https://www.assemblyai.com/blog/speaker-diarization-speaker-labels-for-mono-channel-files, 2023.

[9] A. Défossez, "Hybrid spectrogram and waveform source separation," *arXiv preprint arXiv:2111.03600*, 2022.

[10] S. E. Daniel Stoller and S. Dixon, "Wave-U-Net: A multi-scale neural network for end-to-end audio source separation," *arXiv preprint arXiv:1806.03185*, 2018.

[11] Anishnama, "Understanding bidirectional lstm for sequential data processing," *Medium*, 2023.

[12] L. B. Alexandre Défossez, Nicolas Usunier and F. Bach, "Music source separation in the waveform domain," *arXiv preprint arXiv:1911.13254*, 2018.

[13] "What is transposed convolutional layer?," *GeeksforGeeks*, 2025.

[14] "Dilated convolution," *GeeksforGeeks*, 2023.

[15] A. Twardoch, "audiostretchy." https://github.com/twardoch/audiostretchy, 2023.

[16] D. MALAH, "Time-domain algorithms for harmonic bandwidth reduction and time scaling of speech signals," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 27, no. 2, pp. 121–133, 1979.

[17] D. Bryant, "audio-stretch." https://github.com/dbry/audio-stretch, 2022.