Reverse Voice Filter - Remove One But Keep All Others

Alsu Vakhitova [1] **Batyrkhan Gainitdinov** [1] **Mohammed Deifallah** [1] **Sergey Blintsov** [1] **Nikita Koritsky** [1]

Abstract

This document provides description of the project that is applied to separate target multi-speaker signals from the voice of a single speaker, by making use of a reference signal from the target speaker. We achieve this by reversing an effect of the VoiceFilter system, which consists of two neural networks: A speaker recognition network that produces speaker-discriminative embeddings; (2) A spectrogram masking network that takes both noisy spectrogram and speaker embedding as input, and produces a mask.

Github repo: link
Video presentation: link

1. Introduction

Nowadays speech recognition algorithms have demonstrated great advances that have led to performance improvements in challenging conditions such as noisy and unclear recordings. Nevertheless, speech recognition systems still perform poorly when the target speaker is recorded in crowded and noisy environments that can interfere the speech.

One of the most popular solutions for separation of the voices from different speakers is a *speech separation* system on the noisy audio. The main idea of the task would be receiving a mixed audio, identifying number of speakers in it, and then separating the signals. However, this introduces several challenges. Firstly, the true number of speakers is almost impossible to identify correctly. Secondly, an adequate system might need to be invariant to permutation of speakers' order.

In this project we experiment with the task of isolating the voices of a subset of speakers of interest from the commonality of all the other speakers and noises. Most importantly, we reverse the task, so that it takes a following form: given the mixed input remove one speaker and leave all the others.

As the data set we use three types of audio files which contain the noisy part, it is consisting of several speakers, thus we can relate it to the "cocktail-party problem". To highlight particular speaker we are to use reference audio, which is also used when computing embeddings in speaker encoder stage, so it relates to speaker-dependent task; and clean audio in order to calculate the loss function and by means of combining several clean speakers create noisy audio.

The process of removing all the interfering speakers from the source audio is performed by using pre-trained LSTM model to obtain speaker vectors, thus we leave only the target speaker. The vectors go to the voice filter which provides a time-frequency soft mask predicted as binary output with input spectral magnitude elementwise multiplication. The input spectrogram to the Voice Filter is calculated by means of using the short time Fourier transform (STFT) from the noisy audio.

The main contributions of this report are as follows:

- In Section 2 we provide a comprehensive overview of old, recent and state-of-the art methods for solving the problem of voice separation.
- In Section 3 we explicitly describe algorithms, models, methods and approaches used for solving our project's problem.
- In Section 4 we provide an extensive description of the experiments we have completed.

2. Related Work

For many years, audio source separation methods, and machine listening methods in general, have been based on spectrograms computed using the short-time Fourier transform (STFT). Usage of magnitude spectrogram representation of the signal as an input for further processing with a neural network is extremely popular and widely used (Grais et al., 2014; Simpson et al., 2015; Huang et al., 2014).

[1]Skolkovo Institute of Science and Technology, Moscow, Russia. Correspondence to: Alsu Vakhitova <A.Vakhitova@skoltech.ru>, Batyrkhan Gainitdinov <B.Gainitdinov@skoltech.ru>, Mohammed Deifallah <M.Deifallah@skoltech.ru>, Sergey Blintsov <S.Blintsov@skoltech.ru>, Nikita Koritskiy <N.Koritskiy@skoltech.ru>.

Final Project of the Machine Learning 2021 Course, Skoltech, Moscow, Russian Federation, 2021.

There are two main parts to contribute in voice separation: speaker encoder with the d-vector output of clean speaker audio, i.e speaker-discriminative voice embedding and voice filter which takes the d-vector as the input and makes voice extraction and separation. There are also another methods to perform voice separation in addition to VoiceFilter (Wang et al., 2019), those are DENet (Wang et al., 2018), SpeakerBeam (Delcroix et al., 2018).

The advantage of the presented workflow is that there are no requirements to use some additional information, such as in these papers: (Afouras et al., 2018; Ephrat et al., 2018). In the listed works the information authors use in addition to audio files are lips movements, i.e. the visual input data have to be used in workflow, resulting in difficulties related to new requirements on data gathering.

Further works in Voice separation are present e.g. in this work: (Wang et al., 2020). For the separation of target audio VoiceFilter-Lite is presented in the workflow. In addition, the different approach is about using another evaluation metric, in this case to avoid the over-supression the asymmetric loss is used, thus the authors are aimed to detect under-suppression errors; limitation of convolutional layer dimension (1D instead of 2D layer) and the ability to take streaming inputs. Also the voice filter can easily perform without providing the embedding speaker d-vector.

According to (Nachmani et al., 2020) authors use new approach, the model which apply RNN model for voice separation in the crowded audio file. Instead of WER and SDR the utterance level permutation invariant training (uPIT) is presented in the work.

In addition, there is another approach regarding the binary mask which is the result of prediction by the voice filter. In the work (Simpson, 2015) a probabilistic estimate of the ideal binary mask was obtained.

3. Algorithms and Models

Basically, the architecture of the system consists of 2 models that are trained independently: the speaker encoder, and the VoiceFilter system. The overview of the model can be seen in the Figure 1.

3.1. Speaker Encoder

Encoder takes an audio sample of the target speaker and produces a 256 dimensional *d-vector* that represents the speaker. This approach was proposed by Wan (Wan et al., 2020) and shows great perpormance. Encoder is used to represent the speaker and give this information to the VoiceFilter for it to be able to understand what voice ot extract. As in Wan paper, the model is a 3-layer LSTM network that takes a log-mel filterbank energies and trained with a generalized end-to-end loss.

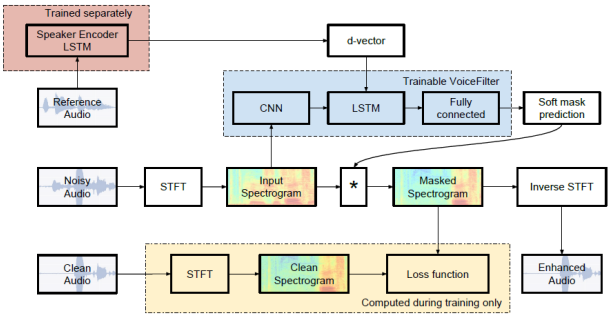

Figure 1. System architecture (Wang et al., 2019)

3.2. VoiceFilter System

The VoiceFilter model takes 2 inputs: d-vector – encoded information of the target speaker–, and a noisy audio in a form of magnitude spectrogram. As an output the network provides a soft mask that is then element-wise multiplied with the noisy audio (it's magnitude spectrogram). To obtain the clean, enhanced waveform, the phase of the original audio and the enhanced magnitude spectrogram is merged and inverse STFT is applied. Training of the network consist of the minimizing of the difference between the target and the masked magnitude spectrograms.

The network is based on the speech enhancement work of Wilson (Wilson et al., 2018). It consists of 8 convolutional, 1 LSTM and 2 fully connected layers. The last layer has a sigmoid activation, all the rest – a ReLU.

The encoded d-vector in the VoiceFilter is concatenated to the output of the last convolutional layer in every time step and the result goes to the LSTM. This was done mainly because we do not need to apply convolutions to the embedding since it already a dense representation of the target speaker. Also, convolutions should be applied only to homogenious input, while magnitude spectrogram and speaker embedding are the different signals.

3.3. Dataset

In order to train the model we have used LibriSpeech dataset (Panayotov et al., 2015) [1]. The LibriSpeech corpus is derived from audiobooks that are part of the LibriVox project, and contains 1000 hours of speech sampled at 16 kHz. Because we were extremely limited in computational resources we could not use the 23GB training dataset. Instead all of our experiments were performed on the development set (dev-clean) which contains 40 recordings (5.4 hours) of 20

[1] https://www.openslr.org/12

female and 20 male readers.

3.4. Data Preprocessing & Generation

In the very beginning all audio-samples are being normalized. The normalization is performed with the loudnorm filter from FFmpeg [2], which was originally written by Kyle Swanson [3]. This ensures that multiple files normalized with this filter will have the same perceived loudness. The window size of 1600 ms was chosen according to the original VoiceFilter paper (Wang et al., 2019).

Each training step of the VoiceFilter network requires three components:

- The clean audio from the target speaker, which is the ground truth;
- The noisy audio containing a mix of multiple speakers;
- A reference audio from the target speaker.

The reference audio is picked randomly among all the recordings of the target speaker, and is different from the clean audio. The noisy audio is generated by mixing the clean audio and an interfering audio randomly selected from a different speaker. More specifically, it is obtained by directly summing the clean audio and the interfering audio, then trimming the result to the length of the clean audio. Figure 2 displays the data generation process described.

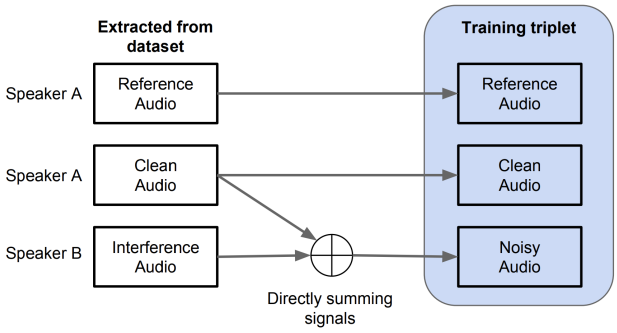

Figure 2. Input data processing workflow (Wang et al., 2019)

3.5. Reversing The VoiceFilter Algorithm

In order to perform a reverse task (remove one speaker and leave everything else) we came up with two strategies:

- Leave the VoiceFilter model unaltered, but change the way the soft mask is applied during inference stage;
- Generate the new reverse the dataset: make the noise to be the target value.

The second approach is pretty straightforward: during the data generation stage the noise is being saved in a target variable. Then, the whole VoiceFilter Algorithm is executed in an original manner.

We came up with the first approach while examining soft masks that resulted prom running the replication task. All values of a mask lie in 0-1 range, and after multiplication by a mixed signal, the noise is being toned down. We hypothesize, that there are ways of "inversing" the mask that would lead to removing the speaker, while leaving the noise. Specifically, we used two formulas:

$$\text{result audio} = \text{mixed audio} * (1 - \text{soft mask}) * (1/\alpha), \quad (1)$$

$$\text{result audio} = \text{mixed audio} * (1/\text{soft mask}) * \alpha, \quad (2)$$

where α is a factor that controls loudness of the resulting audio.

3.6. Evaluation Metric

For evaluation of the system's performance we have decided to use Source to Distortion Ratio. The SDR is a very common metric to evaluate source separation systems (Vincent et al., 2006), which requires to know both the clean signal and the enhanced signal. It is an energy ratio, expressed in dB, between the energy of the target signal contained in the enhanced signal and the energy of the errors (coming from the interfering speakers and artifacts). Thus, the higher it is, the better.

4. Experiments and Results

4.1. Pretrained Speaker Encoder

VoiceFilter utilizes speaker recognition system (d-vector embeddings). For obtaining the d-vector embeddings we have used a pretrained model[4]. This model was trained with VoxCeleb2 dataset (Chung et al., 2018), where utterances are randomly fit to time length [70, 90] frames. Tests are done with window 80 / hop 40 and have shown equal error rate about 1%. Data used for test were selected from first 8 speakers of VoxCeleb1 test dataset (Nagrani et al., 2017), where 10 utterances per each speakers are randomly selected.

[2] http://ffmpeg.org/ffmpeg-filters.html#loudnorm

[3] https://k.ylo.ph/2016/04/04/loudnorm.html

[4] https://drive.google.com/file/d/1YFmhmUok-W76JkrfA0fzQt3c-ZsfiwfL/view?usp=sharing

4.2. Experimental Setup

All the experiments were performed on Google Collab platform [5]. It is important to note that our experimental setup requires usage of Tesla P100, because all other types of GPUs yielded a CUDA compatibility error.

For the VoiceFilter System we have decided to stick with the hyperparameters stated in the (Wang et al., 2019). The VoiceFilter model takes a specrtogram as an input. Firstly, it feeds the input to 8 convolutional layers then to an LSTM with 400 hidden size and lastly, through two linear layers of size 600 and 601, respectively. The last dimension was chosen in order for the resulting soft mask to match the number of channels in an audio sample. Since our dataset is dramatically smaller that the original, we have decreased the number of training steps to 400. The performance is evaluated on a test set throughout training the whole training process every 100 steps. For the optimization mean squared error (MSE) loss was applied. It was because this loss if simple and fast to compute.

In our project we are using 40 train samples and 20 test samples. We are aware that these numbers cannot be considered sufficient for a real neural network training. However, we were significantly limited in computational resources, hence this number was all we could get. Moreover, we are confident that the model will behave in the same manner when introduced to a larger dataset. Therefore, we consider these population sufficient for the educational environment.

4.3. VoiceFilter Replication

We have successfully completed replication of the (Wang et al., 2019) using the unofficial PyTorch implementation [6]. We had to tweak certain parts of the code in order for it to fit the changed setup (smaller dataset). The notebook in which the replication was performed (*reverse_voice_filter.ipynb*) is available in our GitHub repository [7].

4.4. Replication Results Analysis

We have completed the replication of the VoiceFilter System. Figures 3, 4 display some of the statistics gathered over 400 training steps. From the graph we see that both train and test loss decrease is slowing down, thus converging towards the minimum.

In the original paper (Wang et al., 2019) the authors achieve 12.6 SDR score. Our model has converged to 5.9. This decrease is perfectly normal, because, as was stated before, we had limited computational resources and had to use a tiny portion of data and train on it only for 400 steps.

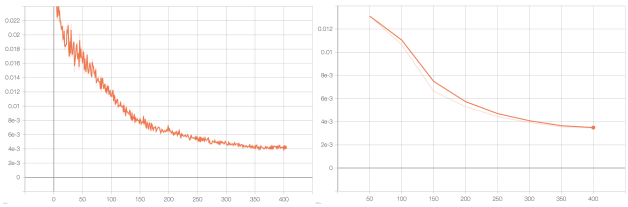

Figure 3. Training (right) and Testing (left) losses over 400 steps.

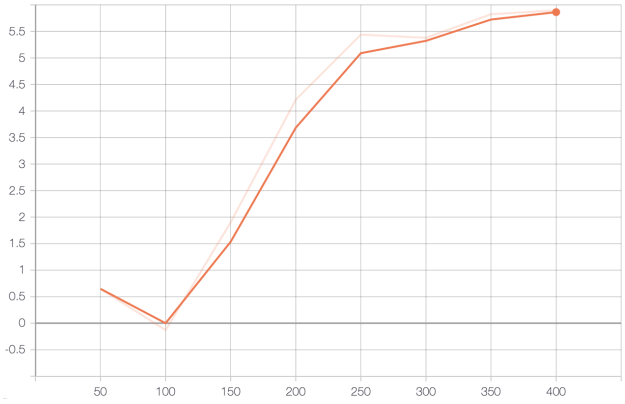

Figure 4. SDR over 400 steps.

Figures 5, 6, 7 illustrate the masking capabilities of the replicated VoiceFilter. In the Figure magnitude spectrogram of the input mixed signal. Figure 6 shows the target spectrogram, which is an clean speech of the target speaker. Finally, Figure 7 displays the output spectrogram, which resulted from point-wise multiplication of the mixed signal by the predicted soft mask. We observe, that the result is in fact pretty close to the target, which signals that our model is "on the right track".

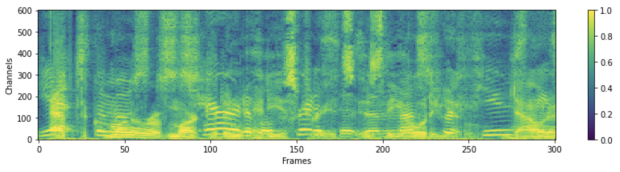

Figure 5. Replication task: spectrogram of mixed signal

To make approbation of our VoiceFilter performance further downstream tasks were conducted. There were large amount of available models, which can perform speech recognition and translating to different languages; We solved the task

[5] https://colab.research.google.com/

[6] https://github.com/mindslab-ai/voicefilter

[7] https://github.com/palette-knife25/reverse-voice-filter/

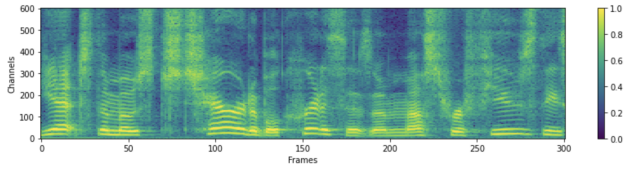

Figure 6. Replication task: spectrogram of target signal

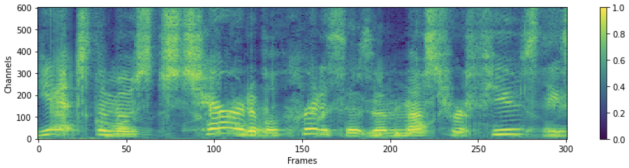

Figure 7. Replication task: spectrogram of resulting signal

by means of using the SpeechRecognition library, which showed-up for us user-friendly.

Especially in our case we compared 2 types of audio files with the target one. Those are the result, which is the filtered audio sample and the noisy speaker audio.

For the analysis of whether VoiceFilter improves performance of other downstream tasks, we have manually evaluated 10 audio files on a pretrained model for a speech recognition task and obtained quite good results. Mostly, we faced such cases, when recognizer command couldn't detect information from the noisy audio and for the filter result it leads to: e.g. noisy speaker: "Could not understand audio"; after voicefilter: "sweetheart"; target speaker: "sweetheart answered". In addition, there were cases, when VoiceFilter audio had larger fraction of correctly recognized words compared to Noisy Audio, 100% and 33% correspondingly. Due to the fact, that we used only a subset of 40 samples to train our model, the quality is not as perfect as we want, but still VoiceFilter can extract some words from the noisy audio, and after speech recognition the model is not able to identify the noisy speaker, but incorrectly detect the filter result, e.g. noisy speaker: "Could not understand audio"; after voicefilter: "Italy"; target speaker: "all the time has stopped would run out to his home and see the scene". Thus, improvement of our model will lead to better recognition of resulting audio samples.

4.5. Reversing VoiceFilter

Following experiments used approaches described in subsection 3.5.

4.5.1. Reversing Effect Of Soft Mask

This experiment did not require any additional training, thus it was performed right after the replication. The mixed signals and the targets are stayed the same. Figures 8 and 9 illustrate the results of alternative application of mask generated for samples displayed in Figures 5 and 6. We have experimented with a variety of α factors for the both inversion approaches, but in both cases $\alpha = 0.7$ gave subjectively the best results.

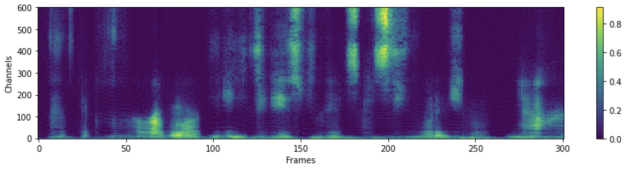

Figure 8. Spectrogram of resulting signal, 1^{st} type mask inversion (equation 1)

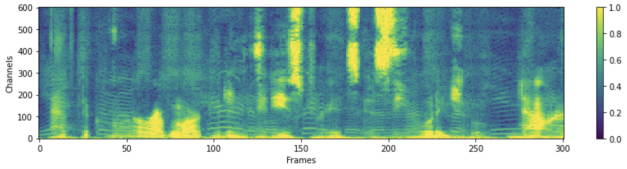

Figure 9. Spectrogram of resulting signal, 2^{nd} type mask inversion (equation 2)

Both methods for mask inversion demonstrate quite poor results. It is obvious that our hypothesis must be rejected.

4.5.2. Reversing Dataset

For this experiment an alternative version of the dataset was generated, where the noise became a target. The training procedure was the same as in the replication task. Figures 10, 11 and 12 display the results.

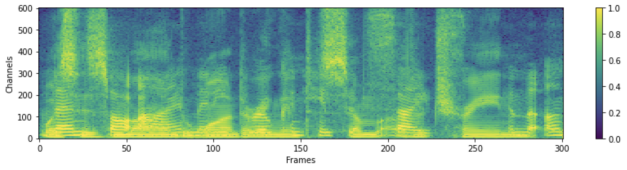

Figure 10. Reversed dataset: spectrogram of mixed signal

Essentially, the Reverse-Dataset approach possesses all the advantages, that we observed in a replication task. We observe, that the model managed to create a mask that brings the resulting signal pretty close to the target.

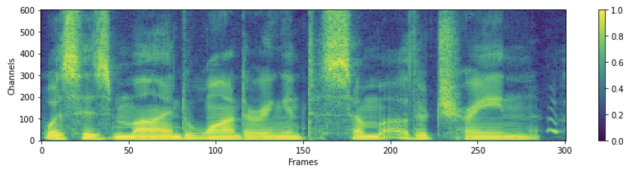

Figure 11. Reversed dataset: spectrogram of target signal

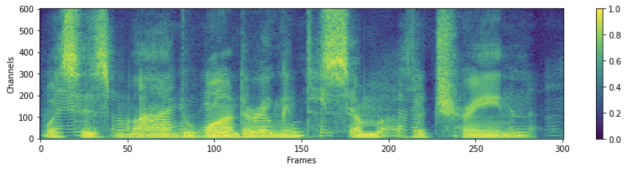

Figure 12. Reversed dataset: spectrogram of resulting signal

5. Conclusion & Future Work

In this project we have completed three tasks. First task was to replicate the results of the (Wang et al., 2019) paper. We have performed the replication on a subset of data and achieved satisfactory results. Second task was to analyze if filtered audio samples improve quality for other downstream tasks such as speech recognition. We have performed the analysis for the Speech Recognition downstream task and demonstrated the improvement. Finally, the last task was to reverse the system so that it keeps everything, but the target speaker. We have experimented with two different strategies: reversing the effect of a mask and reversing the dataset. The former did not demonstrate satisfactory results, while the latter proved to be successful.

There are several improvements to this project that can be made in the future. First and foremost, the optimization metric (MSE loss) can be replaced power-law compressed reconstruction error or peak signal-to-noise ratio (PSNR). These functions are more advanced and can boost the performance of the whole system.

References

Afouras, T., Chung, J. S., and Zisserman, A. The conversation: Deep audio-visual speech enhancement. In *INTERSPEECH*, 2018.

Chung, J. S., Nagrani, A., and Zisserman, A. Voxceleb2: Deep speaker recognition. In *INTERSPEECH*, 2018.

Delcroix, M., Žmolíková, K., Kinoshita, K., Ogawa, A., and Nakatani, T. Single channel target speaker extraction and recognition with speaker beam. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5554–5558, 2018.

Ephrat, A., Mosseri, I., Lang, O., Dekel, T., Wilson, K., Hassidim, A., Freeman, W. T., and Rubinstein, M. Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation. 37(4), July 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201357. URL https://doi.org/10.1145/3197517.3201357.

Grais, E. M., Sen, M. U., and Erdogan, H. Deep neural networks for single channel source separation. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 3734–3738, 2014.

Huang, P., Kim, M., Hasegawa-Johnson, M., and Smaragdis, P. Deep learning for monaural speech separation. *2014 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 1562–1566, 2014.

Nachmani, E., Adi, Y., and Wolf, L. Voice separation with an unknown number of multiple speakers. In *ICML*, 2020.

Nagrani, A., Chung, J. S., and Zisserman, A. Voxceleb: a large-scale speaker identification dataset. In *INTERSPEECH*, 2017.

Panayotov, V., Chen, G., Povey, D., and Khudanpur, S. Librispeech: An asr corpus based on public domain audio books. *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pp. 5206–5210, 2015.

Simpson, A. J. Probabilistic binary-mask cocktail-party source separation in a convolutional deep neural network. *ArXiv*, abs/1503.06962, 2015.

Simpson, A. J., Roma, G., and Plumbley, M. D. Deep karaoke: Extracting vocals from musical mixtures using a convolutional deep neural network. In *LVA/ICA*, 2015.

Vincent, E., Gribonval, R., and Fevotte, C. Performance measurement in blind audio source separation. *IEEE Transactions on Audio, Speech, and Language Processing*, 14(4):1462–1469, 2006. doi: 10.1109/TSA.2005.858005.

Wan, L., Wang, Q., Papir, A., and Moreno, I. L. Generalized end-to-end loss for speaker verification, 2020.

Wang, J., Chen, J. J., Su, D., Chen, L., Yu, M., Qian, Y., and Yu, D. Deep extractor network for target speaker recovery from single channel speech mixtures. In *INTERSPEECH*, 2018.

Wang, Q., Muckenhirn, H., Wilson, K., Sridhar, P., Wu, Z., Hershey, J., Saurous, R. A., Weiss, R. J., Jia, Y., and

Lopez-Moreno, I. Voicefilter: Targeted voice separation by speaker-conditioned spectrogram masking. In *INTERSPEECH*, 2019.

Wang, Q., Lopez-Moreno, I., Saglam, M., Wilson, K., Chiao, A., Liu, R., He, Y., Li, W., Pelecanos, J., Nika, M., and Gruenstein, A. Voicefilter-lite: Streaming targeted voice separation for on-device speech recognition. *ArXiv*, abs/2009.04323, 2020.

Wilson, K., Chinen, M., Thorpe, J., Patton, B., Hershey, J., Saurous, R. A., Skoglund, J., and Lyon, R. F. Exploring tradeoffs in models for low-latency speech enhancement, 2018.

A. Team member's contributions

Explicitly stated contributions of each team member to the final project.

Alsu Vakhitova (20% of work)

- Wrote following parts of the report: Abstract, half of Section 1, Subsections 3.3-3.6, Section 4, Section 5.
- Research for the reverse task.
- Running, debugging and completing the replication code.
- Conducting all experiments and adapting the code for them.
- Fully preparing the GitHub Repo.
- Preparing 90% of the slides.

Batyrkhan Gainitdinov (20% of work)

- Wrote following parts of the report: Section 1, Section 2, half of Subsection 4.4.
- Analyzed if filtered audio samples improve quality for speech recognition.
- Preparing the slides.
- Recording the presentation.

Mohammed Deifallah (20% of work)

- Running and debugging the replication code.
- Research of the reverse task.

Nikita Koritskiy (20% of work)

- Help in group work organizing.
- Wrote following parts of the report: Subsections 3.1 and 3.2.

Sergey Blintsov (20% of work)

- Wrote following parts of the report: Section 2.
- Analyzed the speech recognition downstream task.

B. Reproducibility checklist

Answer the questions of following reproducibility checklist. If necessary, you may leave a comment.

1. A ready code was used in this project, e.g. for replication project the code from the corresponding paper was used.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   General comment: If the answer is **yes**, students must explicitly clarify to which extent (e.g. which percentage of your code did you write on your own?) and which code was used.

   Students' comment: 70% of the code obtained from the unofficial implementation of the original paper (https://github.com/mindslab-ai/voicefilter). However, it required a lot of bugfixing and adapting to our conditions and experiments.

2. A clear description of the mathematical setting, algorithm, and/or model is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

3. A link to a downloadable source code, with specification of all dependencies, including external libraries is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

4. A complete description of the data collection process, including sample size, is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

5. A link to a downloadable version of the dataset or simulation environment is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

6. An explanation of any data that were excluded, description of any pre-processing step are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

7. An explanation of how samples were allocated for training, validation and testing is included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

8. The range of hyper-parameters considered, method to select the best hyper-parameter configuration, and specification of all hyper-parameters used to generate results are included in the report.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

9. The exact number of evaluation runs is included.

   ☑ Yes.
   ☐ No.
   ☐ Not applicable.

   Students' comment: None

10. A description of how experiments have been conducted is included.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

11. A clear definition of the specific measure or statistics used to report results is included in the report.

    ☑ Yes.
    ☐ No.
    ☐ Not applicable.

    Students' comment: None

12. Clearly defined error bars are included in the report.

    ☐ Yes.
    ☐ No.
    ☑ Not applicable.

Students' comment: our project does not require according graphs.

13. A description of the computing infrastructure used is included in the report.

    - [x] Yes.
    - [ ] No.
    - [ ] Not applicable.

    Students' comment: None