DYNAMIC SLIDING WINDOW FOR REALTIME DENOISING NETWORKS

Jinxu Xiang$^{\star 1}$ *Yuyang Zhu*$^{\star 1}$ *Rundi Wu*^{1} *Ruilin Xu*^{1} *Yuko Ishiwaka* ^{2} *Changxi Zheng*^{1}

^{1} Columbia University ^{2} SoftBank Group Corp.

ABSTRACT

Realtime speech denoising has been long studied. Almost all existing methods process the incoming data stream using a sliding window of fixed-size. Yet, we show that the use of fixed-size sliding window may lead to an accumulating lag, especially in presence of other background computing processes that may occupy CPU resources. In response, we propose a new sliding window strategy and a lightweight neural network to leverage it. Our experiments show that the proposed approach achieves denoising quality on a par with the state-of-the-art realtime denoising models. More importantly, our approach is faster, maintaining a stable realtime performance even when the available computing power fluctuates.

Index Terms— realtime, denoising, sliding window, neural network, data stream

1. INTRODUCTION

Realtime speech denoising is a highly demanded audio processing task—perhaps more demanded than ever—as our world is still shadowed by the COVID pandemic and the online meeting is becoming a "new normal" of our daily social life. Most recent, state-of-the-art realtime denoising techniques are all based on neural networks [1, 2, 3, 4, 5, 6, 7, 8]. They seek novel network structures to achieve plausible denoising quality while retaining network simplicity to reduce processing time.

Unlike offline speech denoising, an audio signal in realtime setting is provided in a streaming fashion. The network must process signal samples as soon as they arrive, and generate output samples in the shortest possible delay. As a result, in almost all realtime techniques, a common strategy is to use a sliding window buffer of fixed length.

This seems a natural choice: the network waits until the sliding window buffer is fulfilled with incoming audio samples, and then denoises the data in that buffer. Afterwards, the resulting signal data are fed into a realtime audio player. In this way, the network expects an input signal with a fixed length L. As long as the network processing time is always less than L, the total lag from the arrival of a signal sample to the playing time of the corresponding denoised sample is bounded, larger than L but less than $2L$ (see analysis in Sec. 2.1).

In practice, however, it is difficult, if not impossible, to choose a buffer length L that ensures realtime performance. A large L causes a long lag; a small L may lead to a network processing time longer than L, resulting in choppy audio playback. This is because in a real computing environment, there are always other background processes (e.g., 4K video playback and games). The shorter the L is, the more prone is the denoising network to the CPU occupancy of other processes. In short, the sliding window strategy, albeit fundamental to realtime denoising, remains elusive from careful examination.

We propose a different sliding window strategy, namely the dynamic sliding window. In our approach, the input buffer length is not fixed. The network takes in currently buffered data regardless of its length, and starts to process it with no wait. While the network is running, the newly received data are accumulated in a buffer, ready to be processed when the network finishes its current round of denoising. This sliding window strategy, although conceptually simple, is more robust against CPU occupancy of other processes, and thereby resulting in shorter and more stable lag. To show its advantage, we formally analyze the audio playback delays caused by our approach and the commonly used fixed-size sliding window. To our knowledge, this is the first time the network delays under different sliding window strategies are examined.

Many existing realtime denoising networks (*e.g.*, [1, 2, 3]) are not able to incorporate dynamic sliding window easily (see discussion in Sec. 2.2). We therefore propose a lightweight denoising network tailored for realtime setting: accepting streaming signals and processing them using the dynamic sliding window. By carefully padding and reusing data across sliding windows, our network undergoes *no* degradation on denoising quality compared to the offline non-streaming case.

We conduct extensive experiments comparing the proposed model with several state-of-the-art realtime denoising methods. Results suggest that our proposed model produces the smallest playback lag amongst all compared methods while obtaining on-par denoising quality on all the quality metrics.

Most importantly, compared with the prior realtime denoising method [1], our model is more robust to maintaining realtime performance in real-world scenarios, where other background tasks such as Zoom conferences, 4K video editing, and video games may preempt CPU cycles.

2. METHOD

We start by analyzing the audio playback lag when the fixed-size sliding window is used. This is compared to the lag using

* Equal Contribution

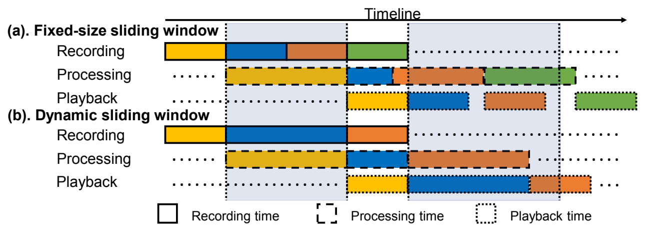

Fig. 1: **Dynamic vs. fixed-size sliding windows.** Different windows are indicated by different colors. Each window is processed by the network individually. The gray areas indicate the period in which CPU cycles are occupied by other processes and thus the network processing time increases.

our proposed dynamic sliding window (Sec. 2.1). Motivated by our analysis, we then propose a lightweight denoising network that leverages dynamic sliding window for faster and more robust denoising (Sec. 2.2).

2.1. Sliding Window for Streaming Data

Fixed-size sliding window. In almost all existing network-based denoising models, the incoming audio signal is treated as a time series of non-overlapping windows $[X_1, X_2, \cdots]$. Each window hosts a constant length L of audio samples (i.e., $X_i \in \mathbb{R}^L$), filled in a streaming fashion. The network F takes in the latest unprocessed window X_i, outputs denoised result $F(X_i)$, and then waits until the next window, X_{i+1}, is fulfilled (see Fig. 1-a). Let t_k denote the network processing time for window X_k. Note that although the window size is fixed, in practice t_k varies over time due to other background processes' CPU occupancy. Our analysis shows that the delay (or lag) d_i, measured by the time difference from the moment of receiving X_i to the moment of network outputting $F(X_i)$, is expressed as

$$d_i = 2L + \max_{1\le p\le q\le i} \sum_{k=p}^{q} (t_k - L). \tag{1}$$

The derivation of Eq. (1) is nontrivial. Because of limited space here, we skip the derivation details; instead, we have made it publicly available online[1].

Our analysis (1) is revealing: in the ideal case wherein $t_i < L$ is always satisfied, the denoising network runs smoothly in realtime with no accumulating lag; the playback delay is upper-bounded by $2L$. However, in reality, the network execution is often affected by other background computing processes, and its processing time t_i may become larger than L. At the same time, a length t_i of audio samples arrives, accumulated in the buffer. To process this amount of data, the network needs to run $\lceil \frac{t_i}{L} \rceil > 1$ times. This can in turn cause audio playback lag to accumulate (and hence the summation term in (1)).

Dynamic sliding window. We propose to dynamically adjust the sliding window size. Immediately after the network finishes processing a data window X_i, the newly received data in the buffer have a length t_i, which may or may not be larger than L. Regardless of the buffer length, we denoise the available buffered data with no wait. Figure 1-b illustrates the process. Under this strategy, the delay d_i for playing back the window X_i is

$$d_i = \max_{k\le i} (t_{k-1} + t_k), \quad i \ge 2. \tag{2}$$

When $i = 1$, the delay for the first window is $d_1 = L_0 + t_1$, where L_0 is an initial window size to start the denoising process at the beginning. We again refer the reader to our online document for the derivation of Eq. (2).

This analysis shows that d_i depends only on the network processing time of two consecutive windows X_{k-1} and X_k. In contrast to the fixed-size sliding window (shown in (1)), there is no accumulating delay. Thus, our approach is more robust against computing power fluctuation. This is a remarkable advantage, since in a real computing environment, the computing power for denoising constantly varies (see Sec. 3.3).

2.2. Network Structure and Data Padding

While the dynamic sliding window strategy is independent from specific network structures, many existing realtime denoising networks [1, 2, 3, 4, 5] can not be easily adapted to utilize it. Some of them require a predetermined sliding window size prior to the network execution [1]. Others focus on reducing the network inference cost, but how they handle the incoming data stream remains unclear [2, 3, 4, 5].

Proposed network structure. We propose a lightweight denoising network using the dynamic sliding window. Our network is built upon the noise removal component in [9] (and thus much simpler than the denoising model therein). Input to our network is a spectrogram $\mathbf{s_x}$ obtained by applying STFT on a data window X_i. The spectrogram $\mathbf{s_x}$ is first processed by a 2D convolutional layer with kernel size $(5, 5)$ and dilation $(1, 1)$ in time-frequency domain. The resulting feature map is fed into a unidirectional LSTM [10] of hidden size 400. Finally, three fully-connected layers of hidden size $(400, 600, 512)$ are applied for each time bin. Similar to other speech enhancement models [9, 11, 12], our network outputs a complex-valued mask $\mathbf{c}$ of the same dimension as $\mathbf{s_x}$. Lastly, the denoised audio signal is obtained by applying the inverse STFT to $\mathbf{c} \odot \mathbf{s_x}$, where $\odot$ denote the Hadamard product.

At training time, we optimize the following loss function:

$$\mathcal{L} = \mathbb{E}_{\mathbf{x}\sim p(\mathbf{x})} ||\mathbf{c} \odot \mathbf{s_x} - \mathbf{s}^*_{\mathbf{x}}||_2, \tag{3}$$

where $\mathbf{s}^*_{\mathbf{x}}$ denotes the ground truth spectrogram of the clean audio. When computing the STFT, we set the number of FFT bins to 510, Hann window size to 400, and hop length to 128.

Data padding. Our network (and also many others) has convolutional layers, which require padding to process data on the boundary. To handle streaming data in a sliding window fashion, this means we need to buffer enough "future" data

[1]Link to the derivation: http://www.cs.columbia.edu/cg/rtdenoise/.

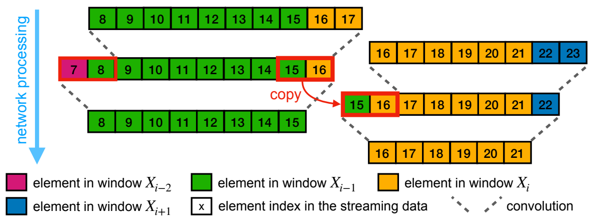
network processing
copy
element in window X_{i-2}
element in window X_{i-1}
element in window X_i
element in window X_{i+1}
element index in the streaming data
convolution

Fig. 2: **Illustration of data padding.** Two consecutive windows of size 8 (green) and 6 (yellow) are processed by two 1D convolutions of kernel size 3. In this case, we pad two future elements for each window (22, 23 for window X_i) and reuse two elements of the first convolution results from the previous window (15, 16 obtained from the processing window X_{i-1}).

when a data window is processed (e.g., block 16 and 17 for window X_{i-1} in Fig. 2). Waiting for the arrival of padding data introduces an additional lag (48ms in our practice). But we can reuse the convolution results of the padding data in the next sliding window (see illustration in Fig. 2). Apart from saving some computational cost, reuse of padding data is critical to the network's denoising quality. It ensures that our network maintains the same denoising quality as if it takes in the entire signal all at once. Perhaps surprisingly, such a guarantee remains lacking in existing realtime denoising networks (see experiments in Sec. 3.2).

3. EXPERIMENTS

Our experiments are twofold: we evaluate our network's denoising quality by comparing it to the state-of-the-art realtime denoising models (Sec. 3.2). We then demonstrate the performance advantages of the dynamic sliding window over the conventional fixed-size sliding window approach (Sec. 3.3).

3.1. Experiment Setup

Datasets. We conduct experiments on two publicly available datasets. The first, provided by Xu et al. [9], has clean audios selected from AVSPEECH [11] and noises from AudioSet [13] and DEMAND [14]. We refer to this dataset as `AAD` dataset. In addition, we also test on `Valentini` [15] benchmark, which contains audio clips from 28 speakers; each clip has its corresponding clean and noisy versions.

Evaluation metrics. To evaluate the denoising quality, we use the following widely used objective metrics: (i) STOI: Short-Time Objective Intelligibility [16]; (ii) PESQ: Perceptual evaluation of speech quality (we use narrowband version) [17]; (iii) CSIG: MOS predictor of signal distortion [18]; (iv) CBAK: MOS prediction of the intrusiveness of background noise [18]; (v) COVL: MOS predictor of overall quality [18]; (vi) SSNR: Segmental Signal-to-Noise Ratio [19].

We use two metrics to evaluate the networks' realtime performance: the average network processing time (D_{N}) and the maximum audio playback lag (D_{A}). D_{N} is defined as $\frac{1}{M}\sum_{i=1}^{M} t_i$, where M is the total number of sliding windows, and t_i is the processing time for window X_i. D_{A} measures, as the audio samples come in, the maximum lag between the arrival time of an audio sample and its playback time (after denoising). Unless otherwise stated, the denoising networks are run on CPU (3.60GHz Intel 8-Core i7-9700K), and the metrics are measured in milliseconds.

Table 1: **Denoising quality** on `AAD` dataset in both offline (top) and realtime (bottom) settings. All scores are the average results for input audios with SNR [-10, -7, -3, 0, 3, 7, 10]. Second best is highlighted in cyan.

Mode	Methods	STOI	CSIG	CBAK	COVL	PESQ	SSNR
	noisy	0.73	2.46	2.29	2.03	1.70	-1.99
Non-realtime	`Demucs48`	0.78	2.38	2.39	2.06	1.83	1.28
	`FullSub`	**0.80**	**2.89**	**2.77**	**2.50**	**2.25**	**2.62**
	`RNN-Mod`	0.69	2.49	2.42	2.07	1.72	0.73
	`Ours`	0.77	2.68	2.63	2.31	2.04	1.92
Realtime	`Demucs48`	**0.78**	2.35	2.37	2.04	1.81	1.20
	`FullSub`	0.76	2.60	2.59	2.24	2.00	1.65
	`RNN-Mod`	0.72	1.79	2.32	1.77	1.88	-1.82
	`Ours`	0.77	**2.68**	**2.63**	**2.31**	**2.04**	**1.92**

3.2. Realtime Speech Denoising Quality

We compare our denoising model against several recently proposed realtime denoising networks, including `Demucs48` [1], `FullSub` [2], and `RNN-Mod` [3]. We train these models on the same aforementioned datasets and evaluate denoising quality in both realtime and offline settings. In offline setting, the audio signal is provided at once, and thus no sliding window is needed. By comparing denoising quality of the two settings from the same network, we wish to understand to what extent a sliding window strategy affects the denoising quality.

For `Demucs48`, we use their provided sliding window implementation. `FullSub` and `RNN-Mod` do not provide streaming implementations, and we found that their denoising quality becomes unstable when dynamic sliding window is added. Therefore we adopt for them the fixed-size sliding window of size 80ms with 16ms padding on the past and future ends. We choose this sliding window configuration because our experiments show that it leads to the best possible realtime denoising quality while keeping an acceptable delay.

Table 1 summarizes the evaluation results on `AAD` dataset. Our model has the best or on-par realtime denoising quality for all the quality metrics. Also worth noting is that our model is the only one that ensures the realtime denoising quality the same as its offline counterpart, thanks to the data padding strategy. All other models experience quality degradation when they are switched from offline setting to realtime setting. Moreover, the results of realtime denoising quality on `Valentini` benchmark are reported in Table 2.

3.3. Realtime Performance

Controlled experiments. First, we measure D_{N} and D_{A} of different network models (see Table 3). Since these models

Table 2: **Realtime denoising quality** on `Valentini`.

Methods	STOI	CSIG	CBAK	COVL	PESQ	SSNR
`Demucs48`	**0.94**	**4.46**	**3.63**	**3.92**	**3.34**	7.77
`FullSub`	0.92	3.76	3.51	3.46	3.22	8.40
`RNN-Mod`	0.92	3.96	3.02	3.48	3.02	0.04
`Ours`	0.93	4.09	3.62	3.67	3.27	**9.56**

Table 3: **Comparison of timings.** In addition to D_{N} and D_{A}, we also report cost of network inference for 200ms audio (S_{N}). Here the numbers include the mean and std. of the timings.

Methods	D_{N}	D_{A}	S_N
`Demucs48`	8.9 ± 0.1	$\mathbf{58.4} \pm 0.7$	23.7 ± 1.2
`FullSub`	64.7 ± 0.8	182.0 ± 6.3	96.6 ± 9.6
`RNN-Mod`	3.3 ± 0.1	101.8 ± 0.9	$\mathbf{4.5} \pm 0.8$
`Ours`	$\mathbf{2.7} \pm 0.1$	61.6 ± 0.9	9.0 ± 1.0

take in different lengths of input data, we also measure their network running time for processing a 200ms signal at once without splitting it into multiple windows. All the measurements are done without heavy background processes. The results indicate that in a dedicated computing environment, our network is as fast as the state-of-the-art models.

Next, we perform controlled experiments to understand the realtime performance of dynamic sliding window and fixed-size sliding window in presence of CPU resource fluctuation. To this end, we create two denoise models: both use the same denoising network (in Sec. 2.2) trained on `AAD` dataset; the first one uses dynamic sliding window (referred to as D-model) while the second uses fixed-size sliding window (referred to as F-model). For fair comparison, the initial window size L_0 in D-model is set the same as the fixed window size in F-model ($L_0 = 16$ms). We use the two models to denoise the same set of audios, each of which has a length of 5s. To impose computing power fluctuation, after 2 seconds we deliberately delay the network processing by a factor of s where s is randomly chosen from $[1, 7]$. This is to simulate CPU occupancy by background processes in a controlled way, as we ensure the same amount of delays are added to both D-model and F-model.

Figure 3 shows the measured D_{N} and D_{A} as the audio stream arrives over time. In the beginning, both can run smoothly in realtime (with the playback lag $N_{\mathrm{A}} < 100$ms). After 2 seconds, the computing power starts to fluctuate, causing some sliding windows not to be processed in time. As a result, the playback lag D_{A} of the F-model accumulates, and the output audio playback becomes choppy. In contrast, D_{A} of the D-model stays stable, because it can dynamically increase window size to catch up with the delay. This experiment confirms our theoretical analysis in Eqs. (1) and (2).

Real-world experiments. We then examine the realtime performance of our model in real scenarios wherein the background processes may preempt CPU cycles. Here, we denoise the same set of audios while different software is running in the background, including 4K video playing, Zoom conference, iMovie video editing, and a video game *Apex*. To run the Apex game, we use a Windows10 PC with an 8-core Intel CPU (3.60GHz i7-9700K) and a GPU (NVIDIA GeForce RTX 2070 SUPER); tests with other software are performed on a Macbook Pro (2.3GHz Intel Quad-Core i5). We choose these software because they demand increasingly more CPU cycles.

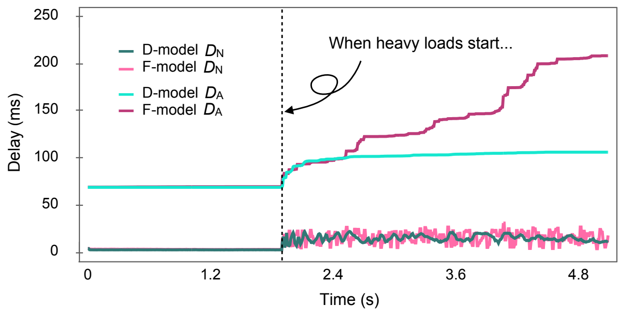

Fig. 3: **Comparison of realtime performance** between dynamic and fixed sliding window approaches. We artificially increase the network processing time randomly by 1-7 times after 2 seconds to simulate CPU power fluctuation. The curves are averaged results over 100 trials with a fixed random seed.

Table 4: **Realtime denoising lag** for `Demucs48` and `Ours` when running other background software. Each cell shows $D_{\mathrm{A}}(D_{\mathrm{N}})$ with both mean and std. Numbers are measured using a 20s audio.

Software	`Demucs48`	`Ours`
None	67.5±2.7 (9.7±0.5)	65.3±1.2 (3.6±0.0)
4K Video	69.2±4.8 (12.0±0.4)	67.8±6.9 (3.6±0.1)
Zoom	74.7±16.1 (13.2±0.4)	65.8±3.0 (3.9±0.1)
iMovie	3938.5±1167.3 (19.1±0.9)	82.4±3.7 (6.3±0.3)
Apex	15789.3±9501.6 (28.1±8.2)	167.2±28.7 (11.4±2.4)

We run these software individually, and meanwhile measure D_{N} and D_{A} using our model and `Demucs48`, respectively. We compare with `Demucs48` because it has its own streaming implementation and offers comparable denoising quality to ours. The results are reported in Table 4. As the background process becomes computationally intensive (e.g., iMovie and Apex), the playback lag of `Demucs48` increases drastically, whereas the lag of our model stays mild and stable. This results clearly indicate the performance advantage of our model and the dynamic sliding window strategy.

4. CONCLUSION

We have proposed a dynamic sliding window strategy for realtime speech denoising. Through careful analysis and experiments, we demonstrated its advantages over the widely used fixed-size sliding window strategy. Our denoising network achieves realtime denoising quality comparable to the SOTAs, while keeping the lag low by utilizing the dynamic sliding window. Remarkably, it allows our model to run robustly in real-world scenarios in presence of other background tasks.

5. REFERENCES

[1] Alexandre Defossez, Gabriel Synnaeve, and Yossi Adi, “Real time speech enhancement in the waveform domain,” *arXiv preprint arXiv:2006.12847*, 2020.

[2] Xiang Hao, Xiangdong Su, Radu Horaud, and Xiaofei Li, “Fullsubnet: A full-band and sub-band fusion model for real-time single-channel speech enhancement,” in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 6633–6637.

[3] Tyler Vuong, Yangyang Xia, and Richard M. Stern, “A modulation-domain loss for neural-network-based real-time speech enhancement,” 2021.

[4] Qiquan Zhang, Aaron Nicolson, Mingjiang Wang, Kuldip K Paliwal, and Chenxu Wang, “Deepmmse: A deep learning approach to mmse-based noise power spectral density estimation,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 28, pp. 1404–1415, 2020.

[5] Hyeong-Seok Choi, Sungjin Park, Jie Hwan Lee, Hoon Heo, Dongsuk Jeon, and Kyogu Lee, “Real-time denoising and dereverberation with tiny recurrent u-net,” *arXiv preprint arXiv:2102.03207*, 2021.

[6] Yangyang Xia, Sebastian Braun, Chandan KA Reddy, Harishchandra Dubey, Ross Cutler, and Ivan Tashev, “Weighted speech distortion losses for neural-network-based real-time speech enhancement,” in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 871–875.

[7] Longbiao Cheng, Junfeng Li, and Yonghong Yan, “Fsc-net: Feature-specific convolution neural network for real-time speech enhancement,” *IEEE Signal Processing Letters*, 2021.

[8] Ke Tan and DeLiang Wang, “A convolutional recurrent neural network for real-time speech enhancement.,” in *Interspeech*, 2018, pp. 3229–3233.

[9] Ruilin Xu, Rundi Wu, Yuko Ishiwaka, Carl Vondrick, and Changxi Zheng, “Listening to sounds of silence for speech denoising,” in *Advances in Neural Information Processing Systems*, H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, Eds. 2020, vol. 33, pp. 9633–9648, Curran Associates, Inc.

[10] Sepp Hochreiter and Jürgen Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] Ariel Ephrat, Inbar Mosseri, Oran Lang, Tali Dekel, Kevin Wilson, Avinatan Hassidim, William T. Freeman, and Michael Rubinstein, “Looking to listen at the cocktail party: A speaker-independent audio-visual model for speech separation,” *ACM Transactions on Graphics*, vol. 37, no. 4, pp. 1–11, July 2018.

[12] Y. Wang, A. Narayanan, and D. Wang, “On training targets for supervised speech separation,” *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, vol. 22, no. 12, pp. 1849–1858, 2014.

[13] Jort F. Gemmeke, Daniel P. W. Ellis, Dylan Freedman, Aren Jansen, Wade Lawrence, R. Channing Moore, Manoj Plakal, and Marvin Ritter, “Audio set: An ontology and human-labeled dataset for audio events,” in *Proc. IEEE ICASSP 2017*, New Orleans, LA, 2017.

[14] Joachim Thiemann, Nobutaka Ito, and Emmanuel Vincent, “The diverse environments multi-channel acoustic noise database (demand): A database of multichannel environmental noise recordings,” in *21st International Congress on Acoustics*, Montreal, Canada, June 2013, Acoustical Society of America, The dataset itself is archived on Zenodo, with DOI 10.5281/zenodo.1227120.

[15] Cassia Valentini-Botinhao, Xin Wang, Shinji Takaki, and Junichi Yamagishi, “Investigating rnn-based speech enhancement methods for noise-robust text-to-speech,” in *9th ISCA Speech Synthesis Workshop*, 2016, pp. 146–152.

[16] Cees H Taal, Richard C Hendriks, Richard Heusdens, and Jesper Jensen, “An algorithm for intelligibility prediction of time–frequency weighted noisy speech,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 19, no. 7, pp. 2125–2136, 2011.

[17] ITU-T Recommendation, “Perceptual evaluation of speech quality (pesq): An objective method for end-to-end speech quality assessment of narrow-band telephone networks and speech codecs,” *Rec. ITU-T P. 862*, 2001.

[18] Yi Hu and Philipos C. Loizou, “Evaluation of objective quality measures for speech enhancement,” *IEEE Transactions on Audio, Speech, and Language Processing*, vol. 16, no. 1, pp. 229–238, 2008.

[19] Mark A. Clements Schuyler R. Quackenbush, Thomas P. Barnwell, *Objective Measures Of Speech Quality*, Prentice Hall, Englewood Cliffs, NJ, 1988.