INTERSPEECH 2023
20-24 August 2023, Dublin, Ireland

Real time spectrogram inversion on mobile phone

Oleg Rybakov, Marco Tagliasacchi, Yunpeng Li, Liyang Jiang, Xia Zhang, Fadi Biadsy

Google Research

{rybakov,mtagliasacchi,yunpeng,jiangliyang,xiaz,biadsy}@google.com

Abstract

We present two methods of real time magnitude spectrogram inversion: streaming Griffin Lim(GL) and streaming MelGAN. We demonstrate the impact of looking ahead on perceptual quality of MelGAN. As little as one hop size (12.5ms) of lookahead is able to significantly improve perceptual quality in comparison to its causal version. We compare streaming GL with the streaming MelGAN and show different trade-offs in terms of perceptual quality, on-device latency, algorithmic delay, memory footprint and noise sensitivity. For fair quality assessment of the GL approach, we use input log magnitude spectrogram without mel transformation. We evaluate presented real time spectrogram inversion approaches on clean, noisy and atypical speech. We specified conditions when streaming GL has comparable quality with MelGAN: noisy audio and no mel transformation. Streaming GL is 2.4x faster than real time on the ARM CPU of a Pixel4 and it uses 4.5x times less memory than MelGAN.

Index Terms: spectrogram inversion, vocoder, speech2speech

1. Introduction

Spectrogram inversion is an important task for Speech-to-Speech [1, 2, 3] and Text-to-Speech [4, 5, 6] models, where it is often referred to as waveform generation (or vocoder). This is often the last step of the audio processing pipeline that converts a magnitude spectrogram to audio samples. Griffin-Lim [7] is the standard signal processing based approach for solving this task. It is an iterative method that requires processing of the whole audio sequence, so it can not run in streaming mode. Several methods were proposed to run GL in streaming mode: RTISI [8], RTISI-LA [9]. A combination of GL with deep neural network (DNN) was proposed in [10] to balance the quality of the reconstructed signal with the computational load depending on the target application.

Neural network based vocoders have become a popular solution because they can generate speech at a higher quality than GL. A notable example is WaveNet [11], which generates high quality speech, but due to its autoregressive nature has limitations on real time applications. Several improvements were proposed in FFTNET [12], WaveRNN [13], LPCNet [14]. Non-autoregressive models were proposed in ParallelWaveNet [15] and WaveGlow [16]. The latter is characterized by a large model size, which can limit its deployment on mobile phones. Another class of non-autoregressive models is based on GANs [17]: [18, 19, 20, 21, 22].

Previous works mostly focused on mel-spectrogram inversion. Instead, in this paper we focus on linear spectrogram inversion, i.e., without mel transformation, as described in Section 3.2. There are two reasons for that: GL can generate high quality audio and linear spectrograms are used in real production speech-to-speech models, such as for example Parrotron [1].

In this work, we consider on-device audio signal reconstruction from magnitude spectrograms. It has to be speaker independent, have minimal memory footprint, run faster than real time and have latency control at run-time. All above are the reasons we propose streaming GL (based on RTISI-LA [9]). Even though it uses standard signal processing approach we show that it has comparable quality with a model based on GAN[18, 19] on noisy audio and real production Parrotron [1] application. We selected MelGAN as a baseline for streaming vocoder design because it is one of the state of the art and it is fully convolutional, so that we can execute it on multiple input samples in parallel even in streaming mode (for example LPCNet [14] or WaveRNN [13] are also streamable but they have to be executed sequentially as streaming GL). In [18] it was already demonstrated that non streaming MelGAN can run faster than real time on CPU/GPU and outperforms non streaming GL, but there are no papers comparing streaming GL with streaming neural vocoder (e.g. with streaming MelGAN) and there is no subjective evaluation of streaming GL (e.g. [9] reported only SNR evaluation). Our paper fills these gaps.

Main contributions:

1 We propose our redesign of RTISI-LA [9] in TensorFlow and call it streaming GL (open sourced in Figure 1). It allows us to benchmark streaming GL on any kind of hardware (CPU, TPU or GPU). For example on ARM CPU of Pixel4 we show several performance advantages of streaming GL over streaming MelGAN: it is more than 2.4x faster than real time and its memory usage is 4.5x times smaller than streaming MelGAN.

2 We present a streaming MelGAN and explore the perceptual quality of causal and non causal (with lookahead) versions. We show that a streaming MelGAN with only one hop delay (12.5ms) lookahead, outperforms GL approaches and a strictly causal MelGAN on clean audio. We compare it with non streaming base model [18] (it has 187ms delay) and show trade-offs between subjective quality and delay.

3 We present subjective and objective evaluation of streaming GL and show that it has comparable quality with streaming MelGAN on noisy audio and production Parrotron application.

2. Model architectures

All the models considered in this paper receive as input a log magnitude spectrogram generated according to the following steps:

1 The raw audio samples at 16kHz sampling frequency are processed by means of a pre-emphasis filter [23] with the coefficient set to 0.97;

2 The STFT is computed with FFT size *fft_size*=2048, frame size equal to 50ms (*frame_size*=800 samples@16kHz), frame step (a.k.a. hop size) equal to 12.5ms (*frame_step*=200 sam-

10.21437/Interspeech.2023-1137

ples@16kHz) and Hann windowing [24];

3 The complex-valued STFT is converted to a real-valued spectrogram by computing the magnitude of each STFT coefficient;

4 The magnitude spectrogram is processed with a logarithmic compression function which is applied element-wise with added *delta*=1e-2.

The resulting log-magnitude spectrogram is fed as input to a vocoder, which processes it in streaming mode and generates output audio frame in the time domain with length equal to 12.5ms (200 samples).

2.1. Streaming Griffin-Lim algorithm

We hypothesize that streaming GL can have performance advantages (latency and model size) over streaming MelGAN. We also would like to compare streaming GL with streaming MelGAN on subjective and objective quality metrics (there is no such previous research).

The original design of causal streaming GL (also RTISI [8]) and its non causal streaming version with look ahead RTISI-LA [9] can requires design of a special kernel (for example written in c++). This kind of kernel is hard to benchmark on different hardware vs a neural vocoder model which is written in TensorFlow or PyTorch. For side by side comparison it is important that competing approaches are compared in the same environment with the same deep learning framework. That is why in this section we present our design of [9] in TensorFlow.

Our streaming GL is a redesign of RTISI-LA [9]. It is open sourced on Figure 1. The algorithm receives as input a log magnitude spectrogram *mag_f* with size 1025, i.e., equal to the FFT size divided by two, plus one. Then, it inverts the natural logarithm by exponentiating the input magnitude frame (line 6 on Figure 1). The magnitude spectrogram is converted to a complex-valued spectrogram by combining *mag_f* with zero phase. A sliding window queue *mag_w* is updated, by appending the current magnitude frame *mag_f* to the sequence of previously stored frames *mag_w* and then keeping the latest *w_size* frames. With this, *mag_w* always has a fixed number of *w_size* frames with the last dimension equal to 1025. A sliding window queue *stft_w* is updated with the current complex-valued spectrogram, as described in the previous step. We pre-compute phase of committed frames (in line 16) and use them as a phase constrain, so that phase of committed frames do not change during GL iterations below. A number of *n_iters* GL iterations are executed based on the current content of the sliding window queues (line 18). Namely, this consists of computing the inverse and forward STFT, estimating the uncommitted phase and recomputing *stft_w* by combining committed phase *commit_phase* and uncommitted phase *uncommit_phase* with the magnitude spectrogram *mag_w* (line 29 on Figure 1). It allows to flow information between committed and uncommitted frames and use it for phase estimation of uncommitted frames in STFT domain. The output frame *stft_o* is extracted by reading the values of the STFT window queue *stft_w* at index *ind*. Where *ind* is an index of the current uncommitted frame in sliding window, so that all frames with indexes $<$*ind* and indexes $>$*ind* are committed and uncommitted (looking ahead) accordingly.

The algorithm described above is executed in streaming mode, whenever a new log magnitude spectrogram frame is available. Once *stft_o* is computed, a new frame of 200 samples of audio are synthesized running the streaming inverse STFT, according to the implementation in [25]. Streaming inverse STFT introduces an algorithmic delay equal to *frame_size* - *frame_step*. Finally, a de-emphasis filter is computed in the time domain to invert the pre-emphasis filter [23] (with *coef*=0.97)

```
1  # Initialize magnitude and STFT sliding window
2  mag_w = tf.zeros((1,w_size,1025),tf.float32)
3  stft_w = tf.zeros((1,w_size,1025),tf.complex64)
4  def streaming_griffin_lim(mag_f,mag_w, stft_w):
5    # Inverse log
6    mag_f = tf.exp(mag_f) - delta
7    # Magnitude frame to complex frame
8    stft_f = tf.complex(mag_f, 0.0) * tf.exp(
9        tf.complex(0.0, tf.zeros_like(mag_f)))
10   # Magnitude sliding window
11   mag_w = tf.concat([mag_w, mag_f], 1)
12   mag_w = mag_w[:, -w_size:, :]
13   # STFT sliding window
14   stft_w = tf.concat([stft_w, stft_f], 1)
15   stft_w = stft_w[:, -w_size:, :]
16   commit_phase = tf.math.angle(
17       stft_w[:, 0:ind, :])
18   for _ in range(n_iters):  # GL iterations
19     audio_w = tf.signal.inverse_stft(
20         stft_w, frame_size, frame_step,
21         fft_size, window_fn=None)
22     stft_w = tf.signal.stft(
23         audio_w, frame_size, frame_step,
24         fft_size, window_fn=hann_window)
25     uncommit_phase = tf.math.angle(
26         stft_w[:, ind:w_size, :])
27     phase_w = tf.concat([commit_phase,
28         uncommit_phase], 1)
29     stft_w = tf.complex(mag_w, 0.0) * tf.exp(
30           tf.complex(0.0, phase_w))
31   stft_o = stft_w[:,ind:ind+1,:] # Output frame
32   return stft_o, mag_w, stft_w
```

Figure 1: *Streaming GL*

and the final output is normalized by (1.0 + *coef*).

We could not find open source version of original [9] and there was no subjective evaluation metrics reported in [9], so we can not compare our design with the original implementation of [9].

In Section 3 we benchmark the streaming GL algorithm with window size *w_size*=4, *n_iters*=4 iterations and *ind*=2. So that it uses 2 hops from the past and one hop from the future (one hop lookahead) for GL iterations, thus it produces an algorithm delay equal to 1 hop or 12.5ms. Additional delay is introduced by streaming inverse STFT (discussed above), so the total delay is equal to one frame. We label this approach as *sGL1*. We compare it with non-streaming GL, labeled as *nGL*, which takes the whole audio sequence and runs 70 iterations on it, thus it can not run in streaming mode.

2.2. Streaming MelGAN

We present streaming version of *MelGAN* [26]. Its models topology with parameters is the same with [26] as shown in Table 1. In streaming mode the input magnitude spectrogram with time dim 1 (this approach also supports processing multiple frames) and number of channels *ch*=1025, is processed by a 1D convolution *conv1D* with 512 channels and kernel size *ks*=7. This is followed by four upscale1D blocks with channels *ch*, kernel size *ks* and stride listed in Table 1. Then, after the ELU activation function [27], a final 1D convolution is applied with the number of channels *ch*=1 and kernel size *ks*=7. In Table 1 we show how the time dimension and the number of channels are changing with every layer execution, so that in the last conv1D layer we get 200 audio samples with channel dimension 1, thus matching the hop size of the STFT (12.5ms

Table 1: *Model architecture.*

operation	time dim	ch	ks	stride
input	1	1025		
conv1D	1	512	7	1
upscale1D	5	256	10	5
upscale1D	25	128	10	5
upscale1D	100	64	8	4
upscale1D	200	32	4	2
elu	200	32		
conv1D	200	1	7	1

Table 2: *upscale1D structure.*

operation	ch	ks	stride	dilation
elu	ch			
conv1Dtranspose	ch	ks	stride	1
resBlk	ch	3	1	1
resBlk	ch	3	1	3
resBlk	ch	3	1	9

at 16kHz sampling frequency).

The structure of the *upscale1D* block is shown in Table 2: after ELU activation, the signal is upscaled by a Conv1DTranspose layer with the following parameters: number of channels *ch*, kernel size *ks* and *stride*, which are defined in Table 1. The upscaled signal is processed by a sequence of three residual blocks *resBlk* with kernel size *ks* and dilation shown in Table 2.

The structure of the residual block *resBlk* is shown in Table 3: after ELU activation, the signal is processed by 1D convolution with kernel size *ks*=3 and dilation shown in Table 2 corresponding to *resBlk*. Then, the output of the last 1x1 convolution in Table 3 is added to the input of *resBlk* and returned as the final output of the *resBlk*.

To guarantee real-time inference, all convolutions are *causal* and running is streaming mode as in [28]. The core streaming components are open sourced in [25]. In order to provide lookahead, during training the target samples are shifted with respect to the input samples in the waveform domain, before computing the STFT.

We train the neural vocoder (on 4 TPU with 1M training steps during 4.4 days) with the same mix of losses used in [29] to achieve both signal reconstruction fidelity and perceptual quality, following the principles of the perception-distortion trade-off discussed in [30]. The adversarial loss is used to promote perceptual quality and it is defined as a hinge loss over the logits of the discriminator, averaged over multiple discriminators and over time, operating both in the time domain and in the STFT domain. To promote fidelity of the decoded signal with respect to the input waveform as in HifiGAN [19] we adopt two additional losses: i) a "feature" loss, computed in the feature space defined by the discriminator(s) [26]; ii) a multi-scale spectral reconstruction loss [31].

The training data consists of a mix of clean and noisy speech. We introduced noise in the training data, to make a model less sensitive to noisy speech (noise sensitivity will be observed in section 3.2. For clean speech, we use the LibriTTS dataset [32] with the following training splits: *train-clean-100*, *train-clean-360* and *train-other-500*. For noisy speech, we synthesize samples by mixing speech from LibriTTS with noise from Freesound [33]. We apply peak normalization to randomly selected crops of 3 seconds and adjust the mixing gain of the noise component sampling uniformly in the interval $[-30\,\text{dB}, 0\,\text{dB}]$.

Table 3: *resBlk structure.*

operation	ch	ks	stride	dilation
elu	ch			
conv1D	ch	3	1	dilation
conv1D	ch	1	1	1

Table 4: *Delay, streaming latency, file and memory size*

Models	Delay [ms]	Latency [ms]	FileSize [MB]	Memory [MB]
nGL	2000	N/A	0.1	
nMelGAN	187	N/A	25	
sGL1	12	5.2	0.1	7.6
sMelGAN1	12	6.7	25	34
sMelGAN0	0	6.7	25	34

In Section 3 we evaluate standard non streaming MelGAN [26], labeled as *nMelGAN*. It has has the highest quality with high delay 187ms (it is hard to use for real time applications). We hypothesize that with only one hop delay we can get acceptable quality. So we introduce streaming *sMelGAN1* with lookahead 1 hop (12.5ms delay) and compare it with causal model with zero delay *sMelGAN0*. These models have the same topology (defined in Table 1) with 12M parameters.

3. Experimental results

3.1. Models benchmarks on mobile phone

For real time applications it is important to have low delay (difference between time when signal is received and time when corresponding output is generated, excluding processing time: it is also called algorithmic delay), low streaming latency (time required to process one hop of audio in streaming mode). small model size (file size of TFLite module measured in MB) and memory footprint. We benchmarked *nGL*, *sGL1*, *sMelGAN1*, *sMelGAN0* and *nMelGAN* on single-threaded CPU of a Pixel4 CPU and reported results on on Table 4. Models were executed with TFLite [34]. As expected Streaming GL has negligible TFLite model size (100KB), it is 250x times smaller than model size of the MelGAN model. We observe that *sGL1* is 23% faster. *sGL1* uses only 7.6MB of CPU memory and it is 4.5x times less run-time memory than *sMelGAN1* (34MB as shown on Table 4). Non streaming models *nGL* and *nMelGAN* have the worst delay. As expected, causal model *sMelGAN0* has zero delay but it has the worst quality as shown on Figure 2. So we explore a trade-off between delay, model size and quality by introducing models *sGL1* and *sMelGAN1*.

3.2. Subjective quality evaluation

We perform subjective evaluation using the MUSHRA methodology [35] with 10 test audio clips (2–5 seconds each) from the VCTK dataset [36] and show it on Figure 2. Input audio is down-sampled to 16kHz, converted to magnitude spectrogram, and then inverted using the various approaches described in this paper. We compare the ground truth audio, together with *nGL*, *sGL1*, *sMelGAN1*, *nMelGAN*, *sMelGAN0*. We run the evaluation on each of the 10 audio clips, which results in 10 groups of 7 audio clips (outputs of *nGL*, *sGL1*, *sMelGAN1*, *nMelGAN*, *sMelGAN0* and ground-truth, where clips within a group have the same content but varying quality. The speakers in the 10 testing audio clips were not present in the training data of the neural vocoders. To calibrate our 10 raters, we first present them

non-streaming GL with only 3 iterations *nGLI3* (score = 20) with their corresponding ground truth audio (score = 100). We then ask the raters to assign scores between 0 and 100 to each of the 10x7 clips. We ranked the presented methods: *nMelGAN* $\approx$ *sMelGAN1* $>$ *nGL* $>$ *sGL1* $\approx$ *sMelGAN0*, using the Mann-Whitney U rank test (for p-value ≤ 0.01). We use sign $\approx$ to label pairs with p-value > 0.01. The best model (in terms of quality-delay trade-off) on clean data is *sMelGAN1*. With only one-hop lookahead, it significantly outperforms all GL approaches and the causal *sMelGAN0*. Note that *sGL1* has comparable quality with causal *sMelGAN0*.

To assess the noise sensitivity of these vocoders, we repeated our subjective evaluation on the noisy version of the same 10 audio clips from VCTK [36]. In this case, the pairwise ranking comes out to be different from the clean-speech scenario. Again using the Mann-Whitney U rank test (for p-value ≤ 0.01), we observe *nMelGAN* $\approx$ *nGL* $\approx$ *sMelGAN1* $\approx$ *sGL1* $>$ *sMelGAN0*. We believe that the quality reduction of the MelGAN is likely caused by the difference in noise distribution between training and testing time. GL approaches, on the other hand, are less sensitive to noise such that *sGL1* and *sMelGAN1* have similar perceptual quality on this test. In addition *sGL1* outperforms causal *sMelGAN0*, we explain it by 1 hop lookahead in *sGL1*. Hence *sGL1* can be a better choice for faster inference (with less memory consumption), whereas *sMelGAN1* would be a better option on clean audio and only one hop delay. All data with the code to run demo models are open sourced at link[1].

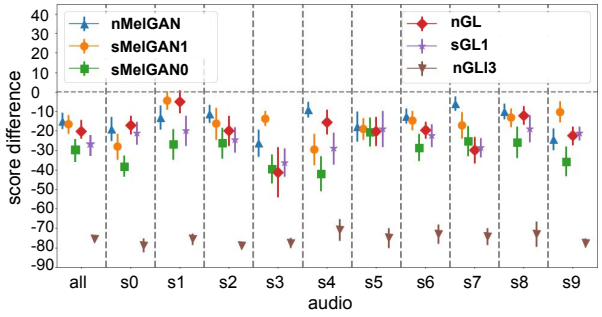

Figure 2: *Average MUSHRA score differences on VCTK clean audio samples, between model outputs and the ground truth input audio, with 95% confidence intervals.*

3.3. Objective quality evaluation: impact on WER of atypical speech conversion

Our final goal is to run the vocoder in real production. For example, on production Parrotron application. Parrotron [1] is an end-to-end speech conversion model that is trained to convert atypical speech to fluent speech of a canonical synthesized voice. This model takes a log-mel spectrogram as an input and directly produces linear spectrogram. It requires a vocoder to synthesize a time-domain waveform in real time, thus a lightweight streaming vocoder is a crucial requirement for such an application. Parrotron normalizes atypical speech, so people with speech disabilities can communicate with others as well as speech enabled interfaces, such as Google Home, Siri or Amazon Alexa. In this case we need to see the WER metric explicitly. For example, state of the art Google ASR can have the WER on atypical speech more than 50% (as shown in paper [1]), but if we pass atypical speech through the Parrotron model then WER can be reduced by 2-4x or more (so that Google Home or Amazon Alexa will able to understand normalized speech). Default production Parrotron model used non streaming GL (as a result it introduced additional several seconds delay impacting customer experience), in our paper we replaced it by streaming GL (reducing the delay down to 20ms) and evaluated the Parrotron model on atypical speech, and showed that streaming GL has no impact on WER and can be used in production. Here is a final demo [37] of production Parrotron with Streaming GL, presented in our paper (we can see that speech is easy to understand and normalized speech is generated as soon as the speaker finishes talking: due to streaming GL there is no delay).

[1] https://github.com/google-research/google-research/tree/master/specinvert

Table 5: *Comparing WER from different vocoders after running Parrotron on atypical speech*

Model	Deaf	ALS	MD
nGL	21.2	22.5	10.4
sGL1	22.2	22.5	10.4
sMelGAN1	20.6	23.0	10.9

In this section, we employ Google's state of the art ASR engine to automatically evaluate the vocoded Parrotron's output while testing our three vocoders. In Table 5 we report word error rate (WER) of ASR engine by passing the converted speech from: a profoundly deaf speaker, a speaker with ALS, and a speaker with Muscular Dystrophy (MD). All those speakers are considered to have severe speech impairments. Comparing WER across different vocoders is a good approximation to check if the linguistic content is preserved after vocoding the converted spectrogram. It is important to note that the absolute WER value is irrelevant here, as it evaluates how Parrotron normalizes atypical speech, not the vocoding phase.

In Table 5 we observe that overall all vocoders preserve the linguistic content for all tested speakers. Interestingly, although the neural vocoder has not been tuned for auto generated spectrograms, it is still performing relatively well when compared to GL. Both GL approaches seem to perform similarly, but it is 1% worse in one of the speakers when compared to the non-streaming GL. We can comfortably conclude that the use of either of streaming approaches are appropriate solution for speech conversion. Presented streaming vocoders can run in real time not only on mobile phone ARM CPU, but on cloud x86 CPU too, so streaming GL was launched in production for Parrotron cloud application, shown in demo [37].

4. Conclusion

We presented our design of streaming GL for inverting log magnitude spectrogram. It is 2.4x faster than real time (on a Pixel4 mobile phone ARM CPU), has only 0.1MB model size (vs MelGAN with 25MB), and similar quality with MelGAN on noisy audio data and real Parrotron application. This makes it attractive for wearable devices.

We also presented streaming MelGAN models and explored the impact of lookahead on the perceptual quality of generated audio. We showed that the model with even one-hop lookahead outperformed GL algorithms and causal MelGAN on clean audio clips. These models are also capable of real-time audio processing, achieving $\approx$2x real-time factor on Pixel4 CPU. However these MelGAN models need to be trained on speech data, and are more sensitive to noise vs GL approach. We showed that both streaming GL and streaming MelGAN are appropriate solutions for atypical speech conversion, e.g. Parrotron [1] application.

5. References

[1] F. Biadsy *et al.*, "Parrotron: An end-to-end speech-to-speech conversion model and its applications to hearing-impaired speech and speech separation," in *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*. ISCA, 2019, pp. 4115–4119.

[2] J. Zhang *et al.*, "Sequence-to-sequence acoustic modeling for voice conversion," *IEEE ACM Trans. Audio Speech Lang. Process.*, vol. 27, no. 3, pp. 631–644, 2019.

[3] Y. Jia *et al.*, "Direct speech-to-speech translation with a sequence-to-sequence model," in *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*. ISCA, 2019, pp. 1123–1127.

[4] Y. Wang *et al.*, "Tacotron: Towards end-to-end speech synthesis," in *Interspeech 2017, 18th Annual Conference of the International Speech Communication Association, 2017*. ISCA, 2017, pp. 4006–4010.

[5] J. Shen *et al.*, "Natural TTS synthesis by conditioning wavenet on MEL spectrogram predictions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018*. IEEE, 2018, pp. 4779–4783.

[6] W. Ping *et al.*, "Deep voice 3: 2000-speaker neural text-to-speech," in *International Conference on Learning Representations*, 2018.

[7] D. Griffin and J. Lim, "Signal estimation from modified short-time fourier transform," *IEEE Transactions on Acoustics, Speech, and Signal Processing*, vol. 32, no. 2, pp. 236–243, 1984.

[8] G. Beauregard, X. Zhu, and L. L. Wyse, "An efficient algorithm for real-time spectrogram inversion," in *Proceedings of the 8th international conference on digital audio effects*, 2005.

[9] X. Zhu, G. T. Beauregard, and L. Wyse, "Real-time iterative spectrum inversion with look-ahead," in *2006 IEEE International Conference on Multimedia and Expo*, 2006, pp. 229–232.

[10] Y. Masuyama *et al.*, "Deep griffin-lim iteration," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019*. IEEE, 2019, pp. 61–65.

[11] A. van den Oord *et al.*, "Wavenet: A generative model for raw audio," in *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*. ISCA, 2016, p. 125.

[12] Z. Jin *et al.*, "FFTNet: a real-time speaker-dependent neural vocoder," in *The 43rd IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Apr. 2018.

[13] N. Kalchbrenner *et al.*, "Efficient neural audio synthesis," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 2415–2424.

[14] J. Valin and J. Skoglund, "LPCNET: improving neural speech synthesis through linear prediction," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019*. IEEE, 2019, pp. 5891–5895.

[15] A. van den Oord *et al.*, "Parallel wavenet: Fast high-fidelity speech synthesis," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018*, ser. Proceedings of Machine Learning Research, vol. 80. PMLR, 2018, pp. 3915–3923.

[16] R. Prenger, R. Valle, and B. Catanzaro, "Waveglow: A flow-based generative network for speech synthesis," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2019*. IEEE, 2019, pp. 3617–3621.

[17] I. J. Goodfellow *et al.*, "Generative adversarial nets," in *Advances in Neural Information Processing Systems*, vol. 27. Curran Associates, Inc., 2014.

[18] K. Kumar *et al.*, "Melgan: Generative adversarial networks for conditional waveform synthesis," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, 2019, pp. 14 881–14 892.

[19] J. Kong *et al.*, "Hifi-gan: Generative adversarial networks for efficient and high fidelity speech synthesis," in *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[20] R. Yamamoto, E. Song, and J. Kim, "Parallel wavegan: A fast waveform generation model based on generative adversarial networks with multi-resolution spectrogram," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE, 2020, pp. 6199–6203.

[21] A. Marafioti *et al.*, "Adversarial generation of time-frequency features with application in audio synthesis," in *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, ser. Proceedings of Machine Learning Research, vol. 97. PMLR, 2019, pp. 4352–4362.

[22] K. Oyamada *et al.*, "Generative adversarial network-based approach to signal reconstruction from magnitude spectrogram," in *26th European Signal Processing Conference, EUSIPCO 2018, Roma, Italy, September 3-7, 2018*. IEEE, 2018, pp. 2514–2518.

[23] S. J. Young *et al.*, *The HTK Book Version 3.4*. Cambridge University Press, 2006.

[24] F. Harris, "On the use of windows for harmonic analysis with the discrete fourier transform," *Proceedings of the IEEE*, vol. 66, no. 1, pp. 51–83, 1978.

[25] O. Rybakov *et al.*, "Streaming keyword spotting on mobile devices," in *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association*. ISCA, 2020, pp. 2277–2281.

[26] K. Kumar *et al.*, "Melgan: Generative adversarial networks for conditional waveform synthesis," in *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019*, 2019, pp. 14 881–14 892.

[27] D. Clevert, T. Unterthiner, and S. Hochreiter, "Fast and accurate deep network learning by exponential linear units (elus)," in *4th International Conference on Learning Representations, ICLR 2016*, 2016.

[28] Y. Li *et al.*, "Real-time speech frequency bandwidth extension," in *IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2021*. IEEE, 2021, pp. 691–695.

[29] N. Zeghidour *et al.*, "Soundstream: An end-to-end neural audio codec," *IEEE ACM Trans. Audio Speech Lang. Process.*, vol. 30, pp. 495–507, 2022.

[30] Y. Blau and T. Michaeli, "The perception-distortion tradeoff," in *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018*. Computer Vision Foundation / IEEE Computer Society, 2018, pp. 6228–6237.

[31] J. H. Engel, L. Hantrakul, C. Gu, and A. Roberts, "DDSP: differentiable digital signal processing," in *8th International Conference on Learning Representations, ICLR 2020*, 2020.

[32] H. Zen *et al.*, "Libritts: A corpus derived from librispeech for text-to-speech," in *Interspeech 2019, 20th Annual Conference of the International Speech Communication Association*. ISCA, 2019, pp. 1526–1530.

[33] E. Fonseca *et al.*, "Freesound datasets: A platform for the creation of open audio datasets," in *Proceedings of the 18th International Society for Music Information Retrieval Conference, ISMIR 2017*, 2017, pp. 486–493.

[34] "Tflite: www.tensorflow.org/lite."

[35] "Method for the subjective assessment of intermediate quality levels of coding systems," ITU-Recommendation 655 BS.1534-3, 2015.

[36] C. Veaux, J. Yamagishi, and K. MacDonald, "CSTR VCTK Corpus: English multi-speaker corpus for cstr voice cloning toolkit," 2016.

[37] "Parrotron demo," https://www.youtube.com/watch?v=BDgJnOivsMM.