INTERSPEECH 2020
October 25–29, 2020, Shanghai, China

High Quality Streaming Speech Synthesis with Low, Sentence-Length-Independent Latency

Nikolaos Ellinas[1], *Georgios Vamvoukakis*[1], *Konstantinos Markopoulos*[1], *Aimilios Chalamandaris*[1], *Georgia Maniati*[1], *Panos Kakoulidis*[1], *Spyros Raptis*[1], *June Sig Sung*[2], *Hyoungmin Park*[2], *Pirros Tsiakoulis*[1]

[1]Innoetics, Samsung Electronics, Greece
[2]Mobile Communications Business, Samsung Electronics, Republic of Korea

{n.ellinas, g.vamvouk, k.markop, aimilios.ch, g.maniati, p.kakoulidis}@samsung.com,
{s.raptis, js6.sung, hm94.park, p.tsiakoulis}@samsung.com

Abstract

This paper presents an end-to-end text-to-speech system with low latency on a CPU, suitable for real-time applications. The system is composed of an autoregressive attention-based sequence-to-sequence acoustic model and the LPCNet vocoder for waveform generation. An acoustic model architecture that adopts modules from both the Tacotron 1 and 2 models is proposed, while stability is ensured by using a recently proposed purely location-based attention mechanism, suitable for arbitrary sentence length generation. During inference, the decoder is unrolled and acoustic feature generation is performed in a streaming manner, allowing for a nearly constant latency which is independent from the sentence length. Experimental results show that the acoustic model can produce feature sequences with minimal latency about 31 times faster than real-time on a computer CPU and 6.5 times on a mobile CPU, enabling it to meet the conditions required for real-time applications on both devices. The full end-to-end system can generate almost natural quality speech, which is verified by listening tests.
Index Terms: Text-to-speech synthesis, real-time, sequence-to-sequence model, streaming inference, end-to-end TTS

1. Introduction

Developments in Deep Learning research and in computer hardware in the recent years have resulted in a paradigm shift in almost all speech processing applications. For text-to-speech (TTS) in particular, there is a shift from either concatenative systems or HMM-based statistical parametric models into neural models. The latter have yielded very high quality synthetic speech while at the same time the overall text-to-speech pipeline has been greatly simplified.

In most cases, the speech synthesis pipeline consists of an end-to-end acoustic model, such as Tacotron [1] and a vocoder, such as WaveNet [2] or WaveRNN [3]. Given a sequence of linguistic features the acoustic model predicts a sequence of speech frames in a parametric form (e.g. mel-spectrograms). The sequence of acoustic features is then fed into the vocoder in order to synthesize the raw audio waveform at the desired sampling rate. In the case of an end-to-end acoustic model the sequence of linguistic features consists of just the character sequence or the phoneme sequence. Such models typically have a very high number of parameters resulting in high runtime complexity. In spite of their impressive and realistic results, this can be prohibitive for applications that require real-time speech synthesis on running devices without GPU, e.g mobile phones, wearables and IoT devices.

1.1. Related work

Tacotron 2 [4] is the most popular acoustic model that produces almost human-level speech when combined with the WaveNet vocoder. It is an attention-based sequence-to-sequence model that leverages Recurrent Neural Networks (RNNs) and location sensitive attention [5]. Escaping the necessity for RNNs which are considered complex and slow to train, convolutional models have also been proposed such as Deep Voice 3 [6] and DCTTS [7], as well as a Transformer [8] model which is based on the original architecture used for machine translation [9].

Autoregressive models are characterized by slow inference speed, as the generation process is done sequentially. There is always a model state, that has to be passed on to the next time-step for the model to be able to produce the acoustic frame. ParaNet [10], ClariNet [11] and GAN-TTS [12] are fully convolutional models that attempt to escape the autoregressiveness of the above models and synthesize speech in parallel. The latter two achieve direct text-to-audio synthesis, using knowledge distillation and adversarial training respectively. FastSpeech [13] is the most consistent attempt at solving the slow inference speed problem. It consists of a Feed-Forward Transformer network that enables parallel acoustic frame generation, guided by a length regulator that produces the alignment between each linguistic unit and its corresponding number of acoustic frames.

As for vocoders, WaveNet [2] is the first autoregressive convolutional model that produces high quality audio when conditioned on mel-spectrograms. It is also an example of very slow inference speed because of its sample-by-sample generation, which is later alleviated in Parallel WaveNet [14] through parallel generation. WaveGlow [15] is a non-autoregressive model that uses normalizing flows and produces state-of-the-art results, but is slow for inference on CPU. Two more recent attempts that instead use adversarial training are MelGAN [16] and Parallel WaveGAN [17].

To our knowledge, the majority of previous research focuses on different architecture designs in order to produce the best speech quality possible, utilizing powerful GPUs both for training and inference. There has not been given much attention to the performance of neural acoustic models when running on CPUs, which represent a more realistic scenario for some real-life applications. In the field of vocoders, WaveRNN [3] and LPCNet [18] are prime examples that a very simple architecture, in this case composed almost only from a couple of RNN cells, when designed properly can produce faster than real-time high quality results.

http://dx.doi.org/10.21437/Interspeech.2020-2464

1.2. Proposed method

In this paper, we emphasize on optimizing the acoustic model, for real-life CPU applications. We build on prior work from [1, 4, 18, 19] with the following contributions:

- A streaming inference method for purely autoregressive models with minimal latency that is independent of the sentence length
- An overall lightweight end-to-end TTS architecture with low complexity that runs faster than real-time on CPU
- The utilization of a robust alignment model that eliminates attention failure errors

We are particularly interested in a TTS system with state-of-the-art quality suitable for applications both on mobile devices and the server-side at low cost. For such applications, there are variables that need to be considered when assessing the feasibility of a model besides its quality. A significant hindrance for many models is that the inference time depends on the sentence length, which can lead to big slowdowns especially now that very long sentence generation is made possible through novel alignment models [19]. Another bottleneck is the latency from the beginning of the synthesis until audible speech is generated.

The proposed acoustic model combined with the streaming method can generate frames about 31 times faster than real-time on a single CPU thread, while simultaneously minimizing its latency down to about 50 ms. On a mobile phone, these numbers change to 6.5 and 240 ms respectively because of the decreased capabilities of the mobile CPU. The full synthesis including the waveform generation is done about 7 times faster than real-time on computer CPU and 2.7 times on mobile.

2. Method

2.1. Acoustic model architecture

The acoustic model maps the input sequence into a sequence of acoustic feature frames that correspond to the representation used by the LPCNet vocoder. It is an attention based sequence-to-sequence model and a direct modification of Tacotron 1 and 2 [1][4].

The encoder converts input sequences $\boldsymbol{p} = [p_1, ..., p_N]$ to learnable embedding vectors, which are then processed by a 2-layer pre-net and a CBHG stack from [1] in order to produce the final encoder representation $\boldsymbol{e} = [e_1, ..., e_N]$. On the decoder side, the inputs are acoustic frames $\boldsymbol{f} = [f_1, ..., f_T]$ processed again by a pre-net. At each decoding step an attention RNN produces a hidden state h_i by consuming the output of the previous step concatenated with the previous attention context vector. The attention module produces the current context vector, which is then fed to a stack of 2 residual decoder RNNs along with the attention RNN hidden state.

The output acoustic frames are predicted by a single fully-connected layer. When the decoding is complete, a residual is constructed from a 5-layer convolutional post-net from [4] and added to the output in order to increase the quality of the final outputs. Similar to Tacotron 2, a binary stop token that signals the end of the acoustic sequence is also predicted from a fully-connected layer with sigmoid activation.

2.2. Alignment model

In [19], a systematic comparison of various attention mechanisms is performed and shows that the purely location-based GMM attention introduced by Graves [20] is able to generalize to arbitrary sequence lengths and not violate the monotonicity of the learned alignment. It does not rely on the encoder outputs for computing the scores, but instead has a state that is passed on to the next step making it the best candidate for our streaming model in order to be sentence-length-independent.

Our model uses a variation of GMM attention, similar to [21], where the mixture of Gaussian distributions is replaced by a Mixture of Logistic distributions (MoL), hence we refer to it as MoL attention. In order to compute the alignments, we directly use the Cumulative Distribution Function (1) of the logistic distribution which is very simple to compute as it is equal to the sigmoid function.

$$F(x; \mu, s) = \frac{1}{1 + e^{-\frac{(x-\mu)}{s}}} = \sigma\left(\frac{x-\mu}{s}\right) \tag{1}$$

Hence, for each decoder step i the alignment probabilities of each encoder timestep j are computed as in (2) and the context vector as the weighted sum of the encoder representations (3).

$$a_{ij} = \sum_{k=1}^{K} w_{ik} \left(F(j + 0.5; \mu_{ik}, s_{ik}) - F(j - 0.5; \mu_{ik}, s_{ik})\right) \tag{2}$$

$$c_i = \sum_{j=1}^{N} a_{ij} e_j \tag{3}$$

The parameters of the mixture are computed at each timestep as in equations (4-6) from the intermediate parameters $\hat{\mu}_{ik}$, $\hat{s}_{ik}$, $\hat{w}_{ik}$ which are predicted by 2 fully connected layers (7) applied to the attention RNN state h_i.

$$\mu_{ik} = \mu_{i-1k} + \exp(\hat{\mu}_{ik}) \tag{4}$$

$$s_{ik} = \exp(\hat{s}_{ik}) \tag{5}$$

$$w_{ik} = softmax(\hat{w}_{ik}) \tag{6}$$

$$(\hat{\mu}_{ik}, \hat{s}_{ik}, \hat{w}_{ik}) = W_2 \tanh(W_1(h_i)) \tag{7}$$

2.3. Vocoder

We use the LPCNet [18] vocoder as adapted for reduced complexity by the parallel work of [22]. By providing an initial estimation of the spectral envelope through Linear Prediction Coefficients (LPC), this method allows the neural model to focus on modeling the excitation signal, thus enabling it to produce state-of-the-art speech quality with a significantly lower complexity. The conditioning parameters required for LPCNet are used as a target for our acoustic model, so that we have an end-to-end speech synthesis pipeline.

2.4. Streaming inference

During speech generation, the acoustic frame sequence must be produced and then fed into the vocoder. A model like FastSpeech [13] can generate the whole sequence and then feed it to a vocoder like WaveGlow [15] which in turn can generate the full raw speech utterance. This process can take very little time on GPU due to the memory efficient parallel computations. However, for longer utterances that need to be synthesized on CPU, this process can take a much longer time, thus increasing the latency from when the synthesis starts until the final audible speech is generated.

We implement a streaming inference process that enables the feeding of acoustic frames into the vocoder before the inference process of the acoustic model is finished as shown in

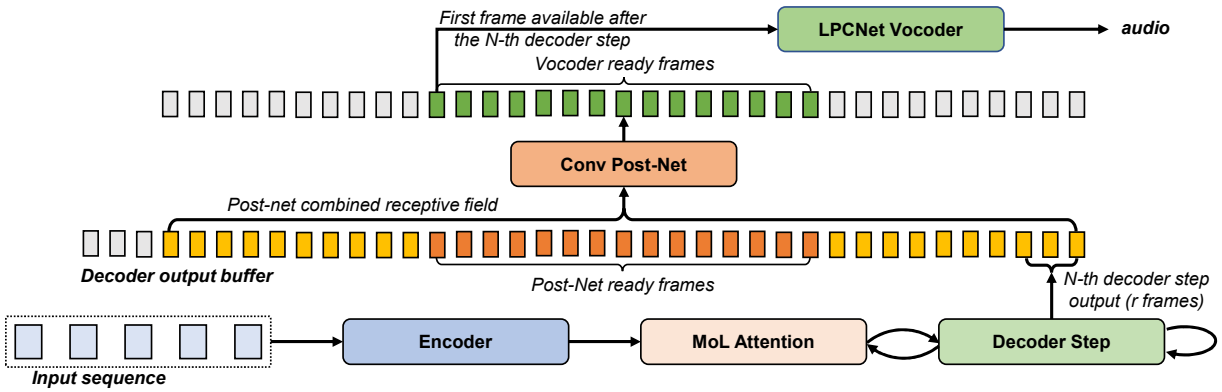

Figure 1: *Streaming inference architecture. The input sequence is fed into the encoder, which produces the linguistic representations. The decoder, conditioned on the context vector produced by the attention module, generates a small batch of acoustic frames at each step which are buffered. When the buffer contains enough frames, they are passed to the post-net in a larger batch. Care is taken so that the buffer includes enough frames to accommodate the total receptive field of the convolutional layers in the post-net. Finally, the output acoustic frames are passed to the LPCNet vocoder, after discarding the initial frames that were already vocoded but were included as part of the receptive field and the most recent frames that need additional future frames to be processed by the post-net.*

Figure 1. The LPCNet vocoder is autoregressive, so it can start generating after receiving the first acoustic frame and because it runs faster than real-time, the resulting audio can be audible almost immediately.

In order to minimize latency, we continuously gather acoustic frames in groups that are passed in parallel to the post-net taking advantage of its convolutional nature. The post-net is trained to refine the full acoustic frame sequence, but since it is fully convolutional it can perform the same process in a smaller sequence just as well. The output frames from each decoder step are accumulated in a buffer and then sent in larger chunks to the post-net. A small window of frames before and after the frame segments to be synthesized should be kept and fed to the post-net as determined by the total receptive field of the convolutional layers (21 in our model), in order for the output to be identical to the non-streaming version of the model. The number of frames in each chunk sent to the post-net is a trade-off between the latency and the real-time-factor (RTF) of the system. If it is small then the latency is small, as the first frames available for vocoding require a few decoder steps, but the computational overhead of the window frames is big hindering the real-time-factor. On the other hand if the number of frames accumulated for post-net processing is too big the window frames overhead diminishes but the latency increases as we need to wait for more decoder steps. A value of 100 frames is chosen for the chunk sent to the post-net that corresponds to 1 second of speech, as it is common practice to buffer some amount of audio before playback in order to avoid static, especially if the audio is transmitted over the network. Then the vocoder synthesizes the first 1 second of audio from the output of the post-net. With the first second of audio synthesized, the user can start listening while the next audio segment is being generated. A detailed visualization of the proposed method can be seen in Figure 1.

The only requirement for this method to be feasible in a production environment, is that the CPU can run the process faster than real time. This way a low and stable latency is guaranteed, regardless of sentence length. In our experiments, we found that a CBHG-based post-net architecture is also feasible and although the final output is not identical to the non-streaming version due to the bidirectional GRU layer, the quality is not significantly affected if the window is adjusted properly.

3. Experiments and results

3.1. Experimental setup

We train our model on the LJ Speech dataset [23], after upsampling the audio data to 24 kHz. The acoustic features used for training are matching the ones by the LPCNet vocoder [18], i.e. 20 Bark-scale cepstral coefficients (increased by 2 bands compared to LPCNet because of the higher sampling rate), the pitch period and pitch correlation.

The input text is first normalized and converted into a phoneme sequence by a traditional TTS front-end module, though without any modification the proposed method can be applied to character-based models. In the encoder, phonemes are mapped into 256 dimensional embeddings and the GRU of the CBHG module has 128 dimensions in each direction. The decoder contains 3 RNNs, a 256-dimensional attention GRU and two 512-dimensional residual decoder LSTMs. The attention module uses a mixture of 5 logistic distributions and its feed-forward layers are 256-dimensional. Pre-net and post-net layers are regularized by dropout [24] of rate 0.5 and the decoder LSTMs by zoneout [25] of rate 0.1.

The network parameters are trained with the Adam optimizer [26], which minimizes the average L1 loss before and after the post-net, batch size 32 and an initial learning rate of 10^{-3} linearly decaying to $3 \cdot 10^{-5}$ after 100,000 iterations. L2 regularization with weight 10^{-6} is also used. For our implementation, we use the PyTorch framework [27].

3.2. Complexity

The decorrelation of the acoustic information into cepstrum and pitch and the very low dimensionality of the used features, allows us to reduce the model parameters without affecting output quality. The proposed model as described in the previous section has a total of 9.5 million parameters.

We also take advantage of the model's ability to generate batches of frames at each decoder time step instead of a single frame. By increasing the number of frames per step (r) the RTF can be reduced, as the decoder runs for fewer steps given the same linguistic input, as shown in Table 1. However, the quality is also dependent on r, so we adopt the value $r = 5$ as we find it provides a good trade-off between speed and quality.

Inference is done on a single thread on an i7-8700K CPU at 3.7 GHz, as we are targeting a realistic server scenario where there are multiple TTS requests running in parallel on the same server. We also report that for $r = 5$ on a mobile Exynos 9820 CPU the RTF is 0.153 ± 0.006 for the acoustic model, while when including the vocoder, the RTF of the total system is 0.136 ± 0.005 ms on computer CPU and 0.372 ± 0.007 ms on mobile.

Table 1: *Average RTF and latency (with standard deviation) for different values of the number of frames per step (r) and corresponding MOS (with 95% confidence interval) on the test corpus. The RTF and latency measurements are for the acoustic model inference only, without the vocoder inference time.*

r	RTF	Latency (ms)	MOS
2	0.067 ± 0.005	93.3 ± 10.9	4.00 ± 0.13
3	0.044 ± 0.002	70.4 ± 9.1	4.23 ± 0.14
5	0.032 ± 0.002	50.1 ± 7.7	4.20 ± 0.10
7	0.025 ± 0.001	40.8 ± 7.0	4.01 ± 0.16
10	0.020 ± 0.001	34.1 ± 6.7	1.63 ± 0.13
		Ground truth	4.5 ± 0.10

3.3. Latency

In order to measure the effectiveness of our method, we compute the latency in both streaming and non-streaming inference setups and also compare the results with other approaches. For comparison we selected the ESPnet toolkit [28] because it provides many state-of-the-art pretrained models. We tested the Tacotron 2 [4], Transformer [8] and FastSpeech [13] models which utilized all available CPU threads for parallelization of operations, as per their original implementation. We also measured the latency of the fastest model, FastSpeech, running on a single thread for a better comparison with our model. Note that these models were trained on audio data at 22.05 kHz sampling rate and with 11.6 ms frame shift, while we use 24 kHz sampling rate and 10 ms frame shift in order to match the vocoder requirements. As a result, our model needs to produce more frames for the same duration of audio.

In these experiments, we are interested in the latency of the acoustic model, i.e. the time interval measured from the beginning of the synthesis until the desired number of frames is generated. A test corpus of 2213 sentences with various lengths was used in order to have accurate measurements of how the sentence length correlates to the latency and the results are shown in Figure 2. Average latency values are also included in Table 1 and show its dependence on the r parameter. On mobile CPU, the value corresponding to $r = 5$ is 240 ms.

By examining Figure 2, we notice that the latency for all non-streaming inference models increases as the sentence length increases. It is clear that our non-streaming model follows a similar pattern, but is faster due to its lightweight architecture. FastSpeech is faster on multi-threaded run mode, but it becomes slower than our non-streaming model when running on a single thread. On the other hand, the streaming model has an almost flat curve showing that its latency increases very slowly as a function of the sentence length. The slight increasing tendency is exclusively attributed to the runtime of the encoder module that being autoregressive is dependent on the input sequence length. However, the percentage of the encoder complexity is minimal, especially if we also consider the complexity of the vocoder, making the overall latency of the system practically sentence-length-independent.

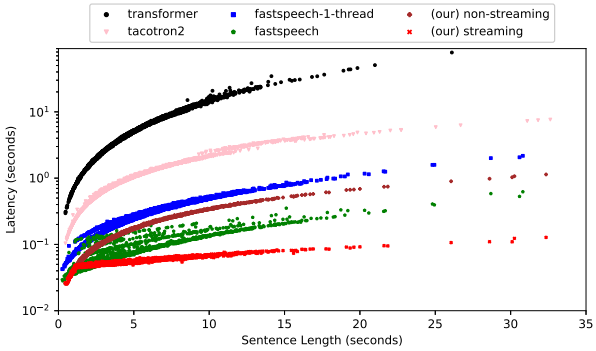

Figure 2: *Latency vs sentence length. A logarithmic scale is used for the latency (y-axis) in order to better visualize the large discrepancies between the various systems shown. The time duration is used to measure sentence length (x-axis) since not all systems have the same frame shift and front-end processing.*

3.4. Quality

We ran a listening test to investigate how quality changes by varying the r parameter. We randomly selected 40 sentences from the benchmark corpus and synthesized them with each system trained on the LJ Speech dataset[1]. The listeners were presented a web-page containing shuffled audio samples and were asked to score the naturalness of each sample on a 5-point Likert scale. The Mean Opinion Score (MOS) for each system including the ground truth is shown in Table 1. For the larger r values 7 and 10, we notice quality degradation, which is attributed to the inability of the model to adequately represent the longer speech information in a single step. Especially for $r = 10$, the model skips some of the phonemes in the input sequence, leading to completely unnatural speech. For r values 3 and 5, the quality is similar while for $r = 2$ the MOS is lower as the model may have not fully converged due to its reduced complexity.

The subjective evaluation shows that our model apart from being very fast, also scores very high in terms of naturalness. At the same time, we found no errors due to failed alignment in the benchmark corpus meaning that the synthesis is very robust thanks to the MoL attention module.

4. Conclusions

We have presented an autoregressive TTS acoustic model that can produce high quality speech many times faster than real time. The combination of the lightweight model architecture with the proposed streaming inference method is ideal for real-time applications on both large scale systems and smaller devices as it offers a low and stable latency without the need of expensive hardware like GPUs. The streaming generation outperforms other tested TTS systems in terms of latency and does not hurt the output quality, because the receptive field of the convolutional post-net is taken into consideration. The attention mechanism that was selected provides great stability even in very long sentences, which is an absolute requirement in real applications. Further work can be made with improvements on runtime by taking advantage of sparsification and quantization methods for better performance which can enable the system to be run on even more low-tier devices.

[1] Samples available at https://innoetics.github.io/

5. References

[1] Y. Wang, R. Skerry-Ryan, D. Stanton, Y. Wu, R. J. Weiss, N. Jaitly, Z. Yang, Y. Xiao, Z. Chen, S. Bengio *et al.*, "Tacotron: Towards end-to-end speech synthesis," in *Proc. Interspeech*, 2017, pp. 4006–4010.

[2] A. v. d. Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," *arXiv:1609.03499*, 2016.

[3] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," in *Proc. International Conference on Machine Learning (ICML)*, 2018, pp. 2410–2419.

[4] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Skerrv-Ryan *et al.*, "Natural tts synthesis by conditioning wavenet on mel spectrogram predictions," in *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4779–4783.

[5] J. K. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho, and Y. Bengio, "Attention-based models for speech recognition," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2015, pp. 577–585.

[6] W. Ping, K. Peng, A. Gibiansky, S. Ö. Arik, A. Kannan, S. Narang, J. Raiman, and J. Miller, "Deep voice 3: Scaling text-to-speech with convolutional sequence learning," in *Proc. 6th International Conference on Learning Representations (ICLR)*, 2018.

[7] H. Tachibana, K. Uenoyama, and S. Aihara, "Efficiently trainable text-to-speech system based on deep convolutional networks with guided attention," in *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4784–4788.

[8] N. Li, S. Liu, Y. Liu, S. Zhao, and M. Liu, "Neural speech synthesis with transformer network," in *Proc. AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 6706–6713.

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2017, pp. 5998–6008.

[10] K. Peng, W. Ping, Z. Song, and K. Zhao, "Parallel neural text-to-speech," *arXiv:1905.08459*, 2019.

[11] W. Ping, K. Peng, and J. Chen, "Clarinet: Parallel wave generation in end-to-end text-to-speech," in *Proc. 7th International Conference on Learning Representations (ICLR)*, 2019.

[12] M. Binkowski, J. Donahue, S. Dieleman, A. Clark, E. Elsen, N. Casagrande, L. C. Cobo, and K. Simonyan, "High fidelity speech synthesis with adversarial networks," in *Proc. 8th International Conference on Learning Representations (ICLR)*, 2020.

[13] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "Fastspeech: Fast, robust and controllable text to speech," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 3165–3174.

[14] A. Oord, Y. Li, I. Babuschkin, K. Simonyan, O. Vinyals, K. Kavukcuoglu, G. Driessche, E. Lockhart, L. Cobo, F. Stimberg *et al.*, "Parallel wavenet: Fast high-fidelity speech synthesis," in *Proc. International Conference on Machine Learning (ICML)*, 2018, pp. 3918–3926.

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

[16] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brébisson, Y. Bengio, and A. C. Courville, "Melgan: Generative adversarial networks for conditional waveform synthesis," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 14 881–14 892.

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

[18] J.-M. Valin and J. Skoglund, "Lpcnet: Improving neural speech synthesis through linear prediction," in *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 5891–5895.

[19] E. Battenberg, R. Skerry-Ryan, S. Mariooryad, D. Stanton, D. Kao, M. Shannon, and T. Bagby, "Location-relative attention mechanisms for robust long-form speech synthesis," in *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6194–6198.

[20] A. Graves, "Generating sequences with recurrent neural networks," *arXiv:1308.0850*, 2013.

[21] S. Vasquez and M. Lewis, "Melnet: A generative model for audio in the frequency domain," *arXiv:1906.01083*, 2019.

[22] R. Vipperla, S. Park, K. Choo, S. Ishtiaq, K. Min, S. Bhattacharya, A. Mehrotra, A. G. C. P. Ramos, and N. D. Lane, "Bunched lpcnet: Vocoder for low-cost neural text-to-speech systems," in *Proc. Interspeech*, 2020.

[23] K. Ito, "The LJ speech dataset," https://keithito.com/LJ-Speech-Dataset/, 2017.

[24] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research (JMLR)*, vol. 15, no. 1, pp. 1929–1958, 2014.

[25] D. Krueger, T. Maharaj, J. Kramár, M. Pezeshki, N. Ballas, N. R. Ke, A. Goyal, Y. Bengio, A. C. Courville, and C. J. Pal, "Zoneout: Regularizing rnns by randomly preserving hidden activations," in *Proc. 5th International Conference on Learning Representations (ICLR)*, 2017.

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *Proc. 3rd International Conference on Learning Representations (ICLR)*, 2015.

[27] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2019, pp. 8024–8035.

[28] T. Hayashi, R. Yamamoto, K. Inoue, T. Yoshimura, S. Watanabe, T. Toda, K. Takeda, Y. Zhang, and X. Tan, "Espnet-tts: Unified, reproducible, and integratable open source end-to-end text-to-speech toolkit," in *Proc. International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7654–7658.