INTERSPEECH 2021
30 August – 3 September, 2021, Brno, Czechia

LinearSpeech: Parallel Text-to-Speech with Linear Complexity

Haozhe Zhang[1,2], *Zhihua Huang*[2,3], *Zengqiang Shang*[1,2], *Pengyuan Zhang*[1,2]*, *Yonghong Yan*[1,2]

[1]Key Laboratory of Speech Acoustics & Content Understanding, Institute of Acoustics, CAS, China
[2]University of Chinese Academy of Sciences, Beijing, China
[3]The Xinjiang Technical Institute of Physics & Chemistry, CAS, Urumqi, China

zhanghaozhe@hccl.ioa.ac.cn

Abstract

Non-autoregressive text to speech models such as FastSpeech can synthesize speech significantly faster than previous autoregressive models with comparable quality. However, the memory and time complexity $\mathcal{O}\left(N^2\right)$ of self-attention hinders FastSpeech from generating long sequences, where N is the length of mel-spectrograms. In this work, we propose LinearSpeech, an efficient parallel text-to-speech model with memory and computational complexity $\mathcal{O}\left(N\right)$. Firstly, we replace standard attention modules in decoder of the model with linear attention modules to reduce the time and memory cost. Secondly, we add a novel positional encoding to standard and linear attention modules, which enable the model to learn the order of input sequence and synthesizing long mel-spectrograms. Furthermore, we use reversible residual layers instead of the standard residuals, which reduce the memory consumption in training stage. In our experiments, LinearSpeech can be trained with doubled batch size than FastSpeech with similar number of parameters. At inference, LinearSpeech achieves more than 2.0x inference speedup on CPU when synthesizing mel-spectrograms longer than 3,500. And our model can synthesize 5.5x longer mel-spectrograms than FastSpeech when running out of 12GB GPU memory. Our subjective listening test also shows that the speech quality of LinearSpeech is comparable to FastSpeech.^

Index Terms: text-to-speech, acoustic model, attention

1. Introduction

In recent years, the Text-to-Speech(TTS) field has seen remarkable progresses. With the development of neural network techniques, neural TTS can synthesize more natural and expressive speech than traditional hybrid TTS systems [1]. Neural TTS is usually composed of acoustic models and vocoders. First, acoustic models, such as Tacotron[2], Tacotron2[3], Transformer-TTS[4] and FastSpeech[5], generate intermediate acoustic feature sequence (mel-spectrogram) from input lexical unit sequence. Then vocoders such as WaveNet[6], WaveRNN[7] and MelGAN[8], synthesize speech from intermediate acoustic feature. FastSpeech[5] is one of the most popular parallel TTS acoustic models with fast and high-quality speech synthesis, which can synthesize mel-spectrograms orders of magnitude faster than autoregressive models[5][9].

However, the high memory and time cost of self-attention modules obstructs long sequence generation with FastSpeech. Self-attention network[10] is the only component in FastSpeech that acts across sequences. With self-attention, FastSpeech can extract text and audio features by considering long term dependecies, but the benefit from self-attention comes with a high cost. Self-attention modules process input sequences of length N with a memory and computational complexity $\mathcal{O}\left(N^2\right)$. In our experiment, FastSpeech needs more than 12 GB GPU memory when generating mel-spectrograms longer than 8,000.

Except the cost of self-attention, positional encoding also hinders long sequence synthesis based on FastSpeech. In order to make use of the order of sequence, sinusoidal positional encoding is added to input embeddings at the bottoms of the encoder and decoder stacks[9]. Although it may allow the model to extrapolate to sequence lengths longer than ones encountered during training[10], attention failure occurs when the length of synthesized sequence is much longer than training sequences.

Finally, the memory cost of self-attention module is times by the number of self-attention layers in training stage. FastSpeech consists of two stacks of feed-forward Transformer blocks, and each block includes a self-attention layer. In training stage, due to the fact that activations need to be stored for back-propagation, the memory cost of the model with M feed-forward Transformer blocks is M-times than a single-layer model, which accounts for a large fraction of memory use.

Fortunately, there has been several efficient Transformer-based model variants proposed recently. Child[11] introduced Sparse Transformer, which uses sparse factorizations of the attention matrix to reduce the self-attention complexity to $\mathcal{O}\left(N\sqrt{N}\right)$. Sparse Transformer employs fixed attention patterns which are defined by strides and local neighbourhoods, which need specific block-sparse variant of matrix-matrix-multiplication and can not be easily implemented. Reformer[12] is another efficient Transformer variant based on locality sensitive hashing (LSH), which reduces the self-attention complexity to $\mathcal{O}\left(N \log N\right)$. Linear Transformer[13] improves the complexity of self-attention from quadratic to linear by using a kernel-based formulation of self-attention and the associative property of matrix products.

In this paper, we propose LinearSpeech, where some of self-attention modules of the model are replaced with linear attention modules, and reversible residual networks are introduced, which reduce the memory and computational cost. We also use Rotary Position Embedding[14] to standard self-attention and linear attention to enable the model to synthesize very long speech, while the speech quality is not degraded compared to FastSpeech. We found that LinearSpeech can be trained on approximately half memory usage compared to FastSpeech of similar parameter size and same batch size. In our experiments, LinearSpeech can generate much longer sequences than FastSpeech with much a lower memory and computational cost. Finally, LinearSpeech can be implemented with normal operators, which makes the model easy to be implemented in industry.

* corresponding author
^ Audio demos are provided at: https://zhangfanruo.github.io/linearspeech/

http://dx.doi.org/10.21437/Interspeech.2021-1192

Table 1: *Memory and time complexity of attention variants. We write l for length, b for batch size, n_h for the number of heads.*

Attention Type	Complexity
Scaled Dot-Product	$\max(bn_h ld_k, bn_h l^2)$
Linear Attention	$\max(bn_h ld_k, \frac{bd_k^2}{n_h})$

2. Background

2.1. Fastspeech

FastSpeech is one of the most popular parallel TTS models, and we adopt it as the model backbone. FastSpeech is composed mainly of a length regulator, an encoder and a decoder. The duration prediction model of the length regulator learns to predict the length of each input lexical unit from a teacher model, such as Transformer-TTS and MFA. Then, the length regulator expands the encoder output sequence according to the predicted length to fix the length mismatch between text and mel-spectrogram sequence.

The encoder and decoder in FastSpeech are two stacks of feed-forward Transformer blocks, which include a pair of self-attention network and position-wise feed-forward network. Self-attention network consists of a multi-head attention to extract the cross-position information. To learn the order of sequences, sinusoidal version positional encoding[10] is added to embedding sequences at the bottoms of the encoder and decoder stacks.

2.2. Linear Attention

Linear attention module is proposed as an alternative to standard attention module with linear complexity. An attention function can be described as mapping a query and a set of key-value pairs to an output[10]. The standard attention in the Transformer is scaled dot-product attention. The global receptive field of scaled dot-product attention brings a very high cost. Attention on sequences of length N is $\mathcal{O}\left(N^2\right)$ in both computational and memory complexity, which becomes the bottleneck in transformer-based models.

The self-attention funtion $\mathrm{Attention}(\cdot)$ compute a weighted average of whole feature vector sequence for all positions. Let $x \in \mathbb{R}^{N\times d_x}$ denote a sequence of N feature vectors of dimensions d_x.

$$\begin{aligned} Q &= xW_Q, \\ K &= xW_K, \\ V &= xW_V, \end{aligned} \tag{1}$$
$$\mathrm{Attention}(x) = V' = \mathrm{softmax}\left(\frac{QK^T}{\sqrt{D}}\right)V.$$

W_Q,W_K,$W_V \in \mathbb{R}^{d_x\times d_k}$ are parameter matrices. These parameter matrices are unique per layer.

We can subscript a matrix with i returns the i-th row as a vector and substitute the similarity function with $\mathrm{sim}\left(q,k\right) = \exp\left(\frac{q^T k}{\sqrt{D}}\right)$ to get a generalized attention equation as follows,

$$V_i' = \frac{\sum_{j=1}^{N}\mathrm{sim}\left(Q_i,K_j\right)V_j}{\sum_{j=1}^{N}\mathrm{sim}\left(Q_i,K_j\right)}. \tag{2}$$

To define an attention function, $\mathrm{sim}\left(\cdot\right)$ is imposed to be non-negative. This includes all kernels $k(x,y):\mathbb{R}^{2\times d_k}\rightarrow\mathbb{R}_+$.

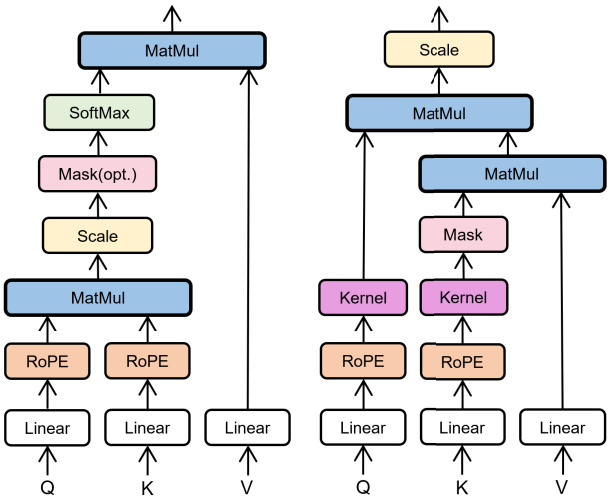

Figure 1: *Architecture of standard self-attention and Linear Attention with RoPE.*

Given such a kernel with a feature representation $\phi\left(x\right)$, we can rewrite equation 2 as follows,

$$V_i' = \frac{\sum_{j=1}^{N}\phi\left(Q_i\right)^T\phi\left(K_j\right)V_j}{\sum_{j=1}^{N}\phi\left(Q_i\right)^T\phi\left(K_j\right)}, \tag{3}$$

and then further simplify it by making use of the associative property of matrix multiplication to

$$V_i' = \frac{\phi\left(Q_i\right)^T\sum_{j=1}^{N}\phi\left(K_j\right)V_j^T}{\phi\left(Q_i\right)^T\sum_{j=1}^{N}\phi\left(K_j\right)}. \tag{4}$$

The above equation is simpler to follow when the numerator is written in vectorized form as follows,

$$\left(\phi\left(Q\right)\phi\left(K\right)^T\right)V = \phi\left(Q\right)\left(\phi\left(K\right)^T V\right). \tag{5}$$

In contrast, linear attention has time and memory complexity $\mathcal{O}\left(N\right)$ because we can compute $\sum_{j=1}^{N}\phi\left(K_j\right)V_j^T$ and $\sum_{j=1}^{N}\phi\left(K_j\right)$ once and reuse them for every query.

2.3. Rotary Position Embedding

Sinusoidal positional encoding hinders FastSpeech from long sequence synthesis. There has been several relative positional encoding variants for Transformer proposed recently[15][16][17], and all these methods are added to attention scores matrix of attention module. However, by making use of the associative property of matrix multiplication, the attention scores matrix disappears in linear attention, which made none of methods mentioned above can be applied to linear attention.

We use a novel positional encoding approach called Rotary Position Embedding (RoPE)[14], extending the self-attention mechanism to efficiently consider representations of the relative positions, which can also be applied to linear attention.

Let us assume that q,k are two matrix with the shape $[N,2]$. A matrix with n returns the n-th row as a vector and a matrix with n,m returns the element in n row, m column.

$$q = \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_N \end{pmatrix} = \begin{pmatrix} q_{1,1}, q_{1,2} \\ q_{2,1}, q_{2,2} \\ \vdots \\ q_{N,1}, q_{N,2} \end{pmatrix}$$

Function $f(x, y) : \mathbb{R}^{1\times 2} \to \mathbb{C}$ converts vector to complex number, which defined as follows: $f(x, y) = x + iy$. Obviously,

$$\begin{aligned} q_m k_n^T &= q_{m,1}k_{n,1} + q_{m,2}k_{n,2} \\ &= \mathrm{Re}((q_{m,1} + iq_{m,2})(k_{n,1} - ik_{n,2})) \\ &= \mathrm{Re}(f(q_m)f(k_n)^*) \end{aligned} \tag{6}$$

Function $\mathrm{Re}(\cdot)$ gets real part of complex number. $*$ means complex conjugate of a complex number. Then, if we multiply the vector at row n by $e^{in\theta}$ to represent the absolute position n of the vector in matrix,

$$\begin{aligned} &(f^{-1}(f(q_m)e^{im\theta}))(f^{-1}(f(k_n)e^{in\theta}))^T = \\ &\mathrm{Re}((f(q_m)e^{im\theta})(f(k_n)e^{in\theta})^*) = \\ &\mathrm{Re}(f(q_m)f(k_n)^* e^{i(m-n)\theta}) \end{aligned} \tag{7}$$

We can found that the result is only directly related to the relative position $m - n$, which represents the order of row m and row n. So we can process every row in q, k as follows:

$$\begin{aligned} q_m &= f^{-1}(f(q_m)e^{im\theta}) \\ &= \begin{pmatrix} q_{m,1}\cos m\theta - q_{m,2}\sin m\theta \\ q_{m,1}\sin m\theta + q_{m,2}\cos m\theta \end{pmatrix}^T \\ &= \begin{pmatrix} q_{m,1} \\ q_{m,2} \end{pmatrix}^T \cos m\theta + \begin{pmatrix} -q_{m,2} \\ q_{m,1} \end{pmatrix}^T \sin m\theta \end{aligned} \tag{8}$$

The shape of q, k is $[N, 2]$, if we want add positional information to Q,K $\in$ $\mathbb{R}^{N\times d_k}$, we can pack every two adjacent columns as q, k corresponding to different θ. So we can process Q,K as follow, $((Q_{col:1}, Q_{col:2}), (Q_{col:3}, Q_{col:4}), ..., (Q_{col:dk-1}, Q_{col:dk}))$ $((K_{col:1}, K_{col:2}), (K_{col:3}, K_{col:4}), ..., (K_{col:dk-1}, K_{col:dk}))$ Then we learn $(\theta_1, \theta_2, \theta_3, ..., \theta_{\frac{dk}{2}})$ to process every row of $(Q_{col:2k+1}, Q_{col:2k+2})$,$(K_{col:2k+1}, K_{col:2k+2})$ as eq.8, we can add position information to self-attention layer, which makes the attention mechanism learn relative positions of input sequences and not limited by the length of training data.

2.4. RevNets

The main idea of reversible residual networks is to allow the activations at any given layer to be recovered from the activations at the following layer. In back-propagation, input of layers can be reversed from their output one by one, rather than saving intermediate values of each layers, which saves memory in training.

In Reformer[12], this idea is applied to Transformer by combining the attention and feed-forward layers inside the revnet block. Layer Normalization[18] is moved inside the residual blocks.

$$\begin{aligned} Y_1 &= X_1 + \mathrm{Attention}(X_2) \\ Y_2 &= X_2 + \mathrm{FeedForward}(Y_1) \end{aligned} \tag{9}$$

A Transformer block can be reversed by subtracting (rather than adding) the residuals:

$$\begin{aligned} X_2 &= Y_2 - \mathrm{FeedForward}(Y_1) \\ X_1 &= Y_1 - \mathrm{Attention}(X_2) \end{aligned} \tag{10}$$

3. The proposed method

We propose LinearSpeech, a memory-efficient non-autoregressive end-to-end speech synthesis system with linear complexities. By using linear attention and Revnet, LinearSpeech gets higher inference speed and less memory redundancy. In this section, we will describe the structure of LinearSpeech and the function of each part.

Our model is based on FastSpeech[5] architecture and consists of an encoder, a decoder and a length regulator. The encoder converts phoneme embedding sequence into phoneme hidden sequence, then the duration predictor predicts duration of each phoneme and the length regulator expands hidden states of phoneme sequences according to duration. The total length of the extended hidden states equals the length of the mel-spectrograms.

3.1. Linear Attention and Rotary Position Embedding

To keep the computational cost low when the length of mel-spectrogram sequence is short, we use a kernel function as defined below[13],

$$\phi(x) = \mathrm{elu}(x) + 1, \tag{11}$$

elu$(\cdot)$ denotes the exponential linear unit[13] activation function, which can keep a positive similarity score in attention and avoid setting the gradients to 0 when x is negative[13]. Our experiments show that the kernel function of eq. 11 and the standard self-attention behave equally, while the linear attention can reduce computational and memory cost significantly. As shown in Table 1, the cost of linear attention doesn't increase with the number of heads, but more number of heads doesn't come with better speech quality in our experiments.

Different from FastSpeech, we add Rotary Position Embedding in every self-attention layer in each FFT block. As mentioned in section 2.3, we add a trainable angle embedding and a linear layer using the tanh$(\cdot)$ function in every self-attention layer to learn θ_i. To keep a positive similarity score in linear attention, RoPE is added before the kernel function in linear attention as Figure 1.

Rotary Position Embedding enable LinearSpeech model to make use of the order of the sequence and learn to synthesize much longer mel-spectrograms than training sequences.

3.2. Encoder

We use feed-forward Transformer blocks as the basic structure for the encoder of LinearSpeech.

The difference between the encoder of LinearSpeech and FastSpeech is the way to add positional information. We add RoPE to every self-attention layer in every FFT block of encoder, to enable the model to process long phoneme sequences. The length of phoneme sequences are much shorter than length of mel-spectrograms, so performance gap of FFT block and linear FFT block in encoder is small. Reversible residual blocks sometimes cause higher voice speed at end of very long synthesized speech, so we use standard self-attention modules in encoder without the reversible residual block.

Therefore, the encoder is composed of four FFT blocks, each one including a standard self-attention layer with RoPE and a feed-forward network. Each hidden layer dimension is 256, and the output is sent to length regulator.

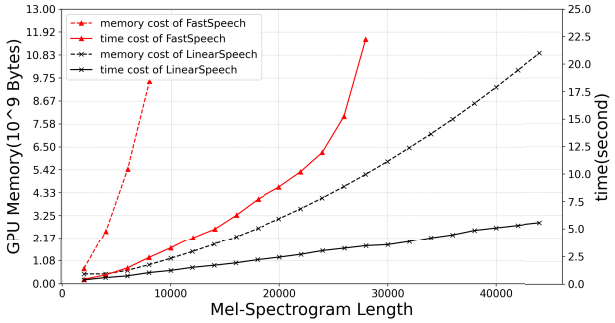

Figure 2: *Comparison of the memory and computational cost for FastSpeech and LinearSpeech.*

3.3. Decoder

There are four linear FFT blocks in decoder, and each one includes a linear attention layer with RoPE and a feed-forward network. To save memory in training stage, we combine the linear attention layer and the feed-forward-layer inside the reversible residual blocks. Layer Normalization is also moved inside the residual blocks. Each hidden layer dimension is 256.

At inference, because of the $\mathcal{O}(N)$ time and memory complexity, linear attention has obvious advantage in performance over standard self-attention when synthesizing long mel-spectrograms.

4. Experiments

4.1. Experimental setup

In training stage, we use a publicly-available Chinese corpus named Chinese Standard Mandarin Speech Corpus[2], which contains about 12 hours of speech from a female speaker, recorded in a professional studio. The training and testing sets are composed of 9900 and 100 utterances respectively. Firstly, the Montreal Force Aligner[19] is used to get the time-point of the initials and the finals. Then, we normalized audio loudness and extracted 80-dimension mel-spectrograms at 16kHz using a 50 ms frame length, 12.5 ms frame hop, with Hann window function. We trained the model with ADAM optimizer with $\beta_1 = 0.9, \beta_2 = 0.98$. We used HIFIGAN as our vocoder for converting mel-spectrograms to waveforms. For 16k HIFI-GAN, we use upsample rates: $[5, 5, 4, 2]$ and upsample kernel sizes:$[10, 10, 8, 4]$, the segment size is 6400, the frame hop is 200 and the frame length is 800.

Parameters of the model and baseline model follow FastSpeech2[9] , but we did not use energy and pitch predictor. Each FFT block is composed of a 1-D conv with kernel size 9 and 256/1024 input/output channels, ReLU activation, a 1-D conv with kernel size 1 and 1024/256 input/output filters, followed by Dropout and Layer Norm.

4.2. Evaluation

We measured the memory cost needed to train FastSpeech and LinearSpeech. Memory costs of the models given number of parameters, batch size are shown in Table 2. The results show that, in training stage, LinearSpeech is more efficient in terms of memory. Linear attention and reversible residual block enable LinearSpeech to be trained with doubled batch size than FastSpeech.

[2]https://www.data-baker.com/open_source.html

Table 2: *Comparison of cached memory consuming(Bytes) on the FastSpeech and LinearSpeech.*

Model	**Batch size**	**Number of parameters**	**Memory cached**
FastSpeech	64	28.45×10^6	8.61×10^9
LinearSpeech	64	28.71×10^6	3.86×10^9
LinearSpeech	128	28.71×10^6	6.84×10^9

Table 3: *CMOS test on FastSpeech and LinearSpeech.*

Model	**CMOS**
FastSpeech	0.04
LinearSpeech	0

We also compared FastSpeech and LinearSpeech with respect to their computational and memory requirements at inference. Sinusoidal positional encoding in FastSpeech was replaced with RoPE to synthesize long sequences. We recorded the peak allocated GPU memory and required time when models synthesizing mel-spectrograms with varying lengths $N \in (2000, 4000, 6000, ..., 44000)$. For memory benchmark we use an NVIDIA Tesla P100 with 12GB of memory. This results in a maximum sequence length of 8,000 elements for FastSpeech and 44,000 for LinearSpeech. The memory costs were evaluated up to the maximum sequence length that fits the GPU memory. The required time was measured on CPU because FastSpeech run out of GPU memory when mel-spectrogram is longer than 8000. As seen in figure 2, the time and memory cost of LinearSpeech scales linearly with respect to sequence length. As expected, our method is faster and requires less memory than the baseline for every configuration.

For the perceptual subjective listening test, we used the Comparative Mean Opinion Score (CMOS) to compare the naturalness of the synthesized speech from FastSpeech and LinearSpeech. We selected 50 unseen sentences for evaluating the models. 10 Chinese native speakers participated in the test. The listeners rated each sample in five categories: A is very bad, bad, normal, good, and very good compared to B and each scored from 1 to 5 points to calculate the CMOS score. We selected 50 unseen sentences for evaluating the models. As shown in Table 3, the speech sample generated from the LinearSpeech shows the comparable quality to FastSpeech.

5. Conclusions

We present LinearSpeech, a parallel text-to-speech model with linear time and memory complexity. Experiments demonstrate that Revnet, linear attention and RoPE are very efficient and would greatly reduce the memory and computational cost in training and inference stage. LinearSpeech can be trained with doubled batch size than FastSpeech with almost the same number of parameters. At inference, LinearSpeech achieves more than 2.0x inference speedup on CPU when synthesizing mel-spectrograms longer than 3,500. And our model can synthesize 5.5x longer mel-spectrograms than FastSpeech with Tesla P100 when running out of 12GB GPU memory. Furthermore, LinearSpeech can be implemented with normal operators, which make the model easy to be deployed.

6. References

[1] K. Tokuda, T. Yoshimura, T. Masuko, T. Kobayashi, and T. Kitamura, "Speech parameter generation algorithms for hmm-based speech synthesis," in *2000 IEEE International Conference on Acoustics, Speech, and Signal Processing. Proceedings (Cat. No. 00CH37100)*, vol. 3. IEEE, 2000, pp. 1315–1318.

[2] 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," *arXiv preprint arXiv:1703.10135*, 2017.

[3] 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 *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 4779–4783.

[4] N. Li, S. Liu, Y. Liu, S. Zhao, and M. Liu, "Neural speech synthesis with transformer network," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, no. 01, 2019, pp. 6706–6713.

[5] Y. Ren, Y. Ruan, X. Tan, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "Fastspeech: Fast, robust and controllable text to speech," *arXiv preprint arXiv:1905.09263*, 2019.

[6] 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 preprint arXiv:1609.03499*, 2016.

[7] 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 *International Conference on Machine Learning*. PMLR, 2018, pp. 2410–2419.

[8] K. Kumar, R. Kumar, T. de Boissiere, L. Gestin, W. Z. Teoh, J. Sotelo, A. de Brébisson, Y. Bengio, and A. Courville, "Melgan: Generative adversarial networks for conditional waveform synthesis," *arXiv preprint arXiv:1910.06711*, 2019.

[9] Y. Ren, C. Hu, T. Qin, S. Zhao, Z. Zhao, and T.-Y. Liu, "Fastspeech 2: Fast and high-quality end-to-end text-to-speech," *arXiv preprint arXiv:2006.04558*, 2020.

[10] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[11] R. Child, S. Gray, A. Radford, and I. Sutskever, "Generating long sequences with sparse transformers," *arXiv preprint arXiv:1904.10509*, 2019.

[12] N. Kitaev, Ł. Kaiser, and A. Levskaya, "Reformer: The efficient transformer," *arXiv preprint arXiv:2001.04451*, 2020.

[13] A. Katharopoulos, A. Vyas, N. Pappas, and F. Fleuret, "Transformers are rnns: Fast autoregressive transformers with linear attention," in *International Conference on Machine Learning*. PMLR, 2020, pp. 5156–5165.

[14] J. Su, Y. Lu, S. Pan, B. Wen, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *arXiv preprint arXiv:2104.09864*, 2021.

[15] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018.

[16] Z. Dai, Z. Yang, Y. Yang, J. Carbonell, Q. V. Le, and R. Salakhutdinov, "Transformer-xl: Attentive language models beyond a fixed-length context," *arXiv preprint arXiv:1901.02860*, 2019.

[17] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

[18] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[19] M. McAuliffe, M. Socolof, S. Mihuc, M. Wagner, and M. Sonderegger, "Montreal forced aligner: Trainable text-speech alignment using kaldi." in *Interspeech*, vol. 2017, 2017, pp. 498–502.