INTERSPEECH 2023
20-24 August 2023, Dublin, Ireland

Streaming Parrotron for on-device speech-to-speech conversion

Oleg Rybakov, Fadi Biadsy, Xia Zhang, Liyang Jiang, Phoenix Meadowlark, Shivani Agrawal

Google Research

{rybakov,biadsy,xiaz,jiangliyang,meadowlark,shivaniagrawal}@google.com

Abstract

We present a fully on-device streaming Speech2Speech conversion model that normalizes a given input speech directly to synthesized output speech. Deploying such a model on mobile devices pose significant challenges in terms of memory footprint and computation requirements. We present a streaming-based approach to produce an acceptable delay, with minimal loss in speech conversion quality, when compared to a reference state of the art non-streaming approach. Our method consists of first streaming the encoder in real time while the speaker is speaking. Then, as soon as the speaker stops speaking, we run the spectrogram decoder in streaming mode along the side of a streaming vocoder to generate output speech. To achieve an acceptable delay-quality trade-off, we propose a novel hybrid approach for look-ahead in the encoder which combines a look-ahead feature stacker with a look-ahead self-attention. We show that our streaming approach is ≈2x faster than real time on the Pixel4 CPU.

Index Terms: speech to speech, parrotron

1. Introduction

With the increase in computational resources of mobile devices and advances in speech modeling in recent years, on-device inference has become an important research topic. Automatic Speech Recognition (ASR), Text to Speech (TTS), and machine translation models, for example, have been ported to run locally on mobile devices with considerable success [1, 2, 3, 4]. In this work, we tackle the difficulties of running end-to-end attention-based Speech-To-Speech (STS) models, which directly map input spectrograms to output spectrograms, on mobile devices. STS models have several applications including STS translation [5], speech conversion and normalization [6, 7].

Our work focuses on Parrotron [6], a STS conversion model that has shown to successfully convert atypical speech from speakers with speech impairments (due to, for example, deafness, Parkinsons, ALS, Stroke, etc.) *directly* to fluent typical speech [8]. To achieve optimal results, Parrotron is adapted for each dysarthric speaker, producing a Submodel for each speaker. [8, 9] Running such a personalized model locally on a device has substantial practical advantages over server-based models, including: scalability, inference without internet connectivity, reduced latency, and privacy.

Unlike TTS and ASR, *both* Parrotron's input and output are acoustic frames – i.e., significantly longer input-output sequences and the model requires more parameters. Due to these extra challenges, running inference on-device, the model size has to be sufficiently smaller to fit in RAM, and the model must run with limited compute resources. We have shown that running all the components of the model in non-streaming mode substantially increases the delay between the time the speaker stops speaking and the time the user obtains the converted speech. Controlling the latency of this model is crucial for natural human-to-human communication, when the system is the only mode of communication for speakers with speech impairments, for example. Parrotron is an encoder-decoder model. There are several approaches for streaming such a model. One is to stream the encoder and decoder simultaneously so that the model generates new speech while the user is speaking [10], or generates translated text while the user is speaking [11]. Alternatively, for speech conversion, we propose to stream the encoder while the user is speaking, and then stream the decoder to generate new speech afterwards, similar to the way the decoder runs in this TTS [4]. We suspect that this option is well suited for a face-to-face dialog, especially for dysarthric input speech conversion [7]. The reader is encouraged to view demo [9]. We present the following contributions:

- A real time on-device streaming STS conversion model where the encoder runs in real-time while the speaker is speaking, and as soon as the user stops speaking, the decoder starts generating output audio.
- A novel approach for streaming hybrid lookahead Conformer encoder which outperforms conventional techniques in terms of delay and accuracy, using a combination of lookahead local self-attention and stacker lookahead.
- A comprehensive analysis and comparison of several streaming approaches: streaming decoder, streaming causal and non-causal encoders, streaming vocoders and quantization in terms of latency, delay, accuracy, model size and memory footprint.

2. Model architectures

2.1. Baseline Parrotron model

Baseline Parrotron model (we call it *Base* model) [7] is composed of a Mel frontend, Conformer encoder, and an auto-regressive LSTM decoder with cross attention, followed by a vocoder to synthesize a time-domain waveform, as shown in Figure 1. *Base* model [7] takes the whole input speech and processes it end to end. The entire input spectrogram is first encoded and stored in the 'Encoded sequence' block on Figure 1. Then, at every decoder step this sequence is processed by a cross 'Location-sensitive attention' block [12]. The prediction from previous time steps are processed by a PreNet (two fully connected layers, 256 units each). The PreNet's output is concatenated with the cross-attention context and passed to two unidirectional LSTM layers (1024 units). The LSTM output and the attention context are concatenated and passed to 'Linear projection', which predicts two frames, each of which is 12.5ms, per call. These frames are processed by a PostNet block (5 convolution layers) and added as residual connection to the output of 'Linear projection'. The final decoder's out-

10.21437/Interspeech.2023-160

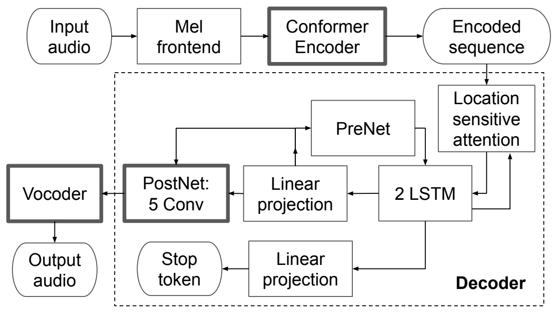

Figure 1: *Conformer Parrotron model.*

put is two linear spectrogram frames. Finally, the output frames are passed through a 'Vocoder'(in *Base* [7] model they use non streaming Griffin Lim) block to generate a time-domain waveform of the converted speech.

Conformer encoder in *Base* model is composed of a sequence of layers: 2 conformer blocks; 1 causal Stacker block; 2 conformer blocks; 1 causal Stacker block followed by 13 conformer blocks. In this paper all models have 17 Conformer blocks with total number of parameters of a model 168M as in *Base* [7].

Causal Stacker block diagram is shown on Figure 2 (a). Input encoded frames (with 512 states), labeled by different colors (on Figure 2 (a)), are processed by several steps: at time *t*, it stacks one past frame from time *t-1*, then pass them through fully connected layer which project them back to original 512 dimensions, then apply 2x subsampling in time dimension to reduce further computations.

One conformer block [13] is a sequence of fully connected, multi-head full self attention, causal convolution (32x1 kernel size), fully connected and normalization layers. Every layer has a residual connection (except layer normalization). Full self attention layer (with 512 states, 8 attention heads) at time *t* computes self attention over all *T* input frames as shown on Figure 3 (c). That is why this model can not run in real time streaming mode: it needs to wait until all frames are available for computation. As a result it has the worst *total delay*. *Total delay* is a difference between the time when a user stops talking and time when output audio (generated by the model) is played back to the user. It is a user-facing metric, so one of the goals of this paper is to find a model with accuracy similar to the *Base* model but with much less *total delay*.

In section 3.2 we show that *Base* model has 7 seconds *total delay* of processing 10 seconds of input audio on Pixel4. It makes it unusable for real time user experience. That is why in the next section 2.2 we present the streaming *Causal Base* Parrotron model which addresses this issue. In sections 2.3 and 2.4 we propose further improvements of *Causal Base* model with better *total delay* accuracy trade-off.

2.2. Streaming *Causal Base* model

Streaming *Causal Base* model has the similar topology with non streaming *Base* model (shown on Figure 1), with several key differences in blocks highlighted by thick line: Conformer Encoder, PostNet, and Vocoder blocks (these blocks are running in streaming mode). The streaming Parrotron model runs encoder in real time streaming mode while the speaker is speaking. As soon as the speaker stops speaking we have the whole encoded speech available in the block "Encoded sequence" as shown on Figure 1. Then we run the spectrogram decoder in streaming mode along the side of a streaming vocoder to generate output

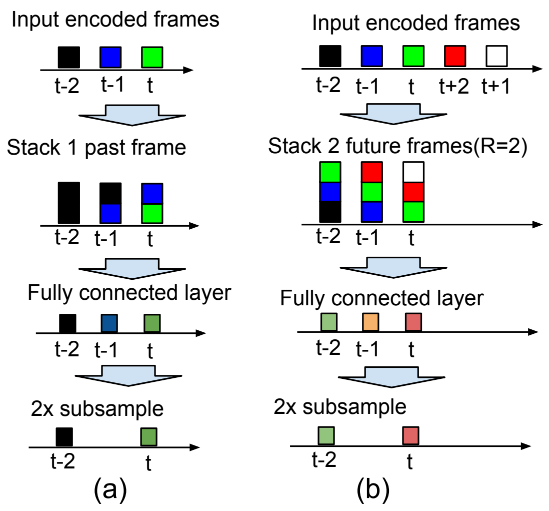

Figure 2: *Causal (a) and lookahead (b) Stackers.*

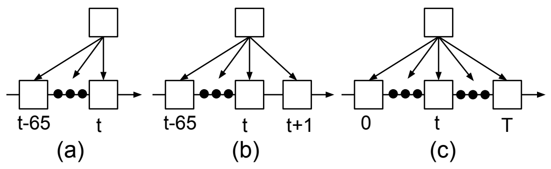

Figure 3: *Self attention diagrams: Local causal (a) local lookahead with right context R=1 (b) and full self attention (c).*

speech in real time.

The standard approach of converting a model to streaming mode is to make all modules causal. So in the Decoder block (on Figure 1) we use causal convolutions in PostNet and call this decoder streaming decoder *sDec*. Output of streaming decoder is processed by streaming vocoder (described in [14]) which generates output audio as shown on Figure 1.

We also replace the Conformer encoder (on Figure 1) by the Causal conformer encoder. It has the same topology with the Conformer Encoder of the *Base* model (described in section 2.1), but full self attention of the Conformer block is replaced by local causal self attention. Local causal self attention at time t computes local self attention over the previous 65 frames, as shown on Figure 3. Causal Conformer encoder has zero delay, so by the time the user stops talking we have all encoded frames stored in "Encoded sequence" block (on Figure 1) and decoder can start streaming output audio. We hypothesize that such an approach has significant quality implications. That is why in the next sections we propose a new streaming Conformer encoders which can run in streaming mode at real time and at the same time have minimum quality impact.

2.3. Streaming lookahead self attention model(*LSA*)

To improve accuracy of the *Causal Base* model we propose to replace "Conformer Encoder" by streaming "lookahead Conformer encoder" shown on Figure 4. We call this model *LSA*(lookahead self attention). Lookahead conformer block has the same topology as the standard Conformer block, but full self attention is replaced with lookahead local self attention. It is a popular approach for modeling lookahead, described in [15, 16], which include future/right hidden-states to improve quality over a fully casual model. For example on Figure 3 (b) lookahead local self attention at time t computes self attention over the last 65 frames and future R frames (R = 1). So it is a

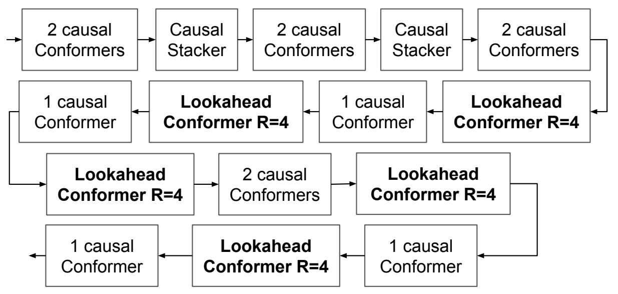

Figure 4: *Streaming lookahead Conformer encoder.*

trade-off between a fully causal (optimal delay) and non-causal encoder (optimal quality). Causal Conformer block is described in section 2.2.

2.4. Streaming lookahead self-attention with a lookahead stacker model(*LSA_LS*)

In this section we hypothesize that with the same *total delay* budget it is possible to further improve accuracy of *LSA* model. So in the model described in section 2.3 we propose to replace streaming lookahead Conformer encoder (shown on Figure 4) by streaming hybrid lookahead Conformer encoder shown on Figure 5. The key difference with the "lookahead Conformer encoder" is that causal Stacker is replaced by lookahead Stacker. For example, a diagram of lookahead Stacker with R=2 is shown on Figure 2 (b). Input encoded frames (with 512 states) are processed by several steps: at time t, it stacks two future frames from time t+1 and t+2, then pass them through fully connected layer which project them back to original 512 dimensions, then apply 2x subsampling in time dimension to reduce further computations. Since each hidden-state now includes information from the future, we hypothesize that causal conformer blocks is more likely to make use of this information in combination with a lookahead conformer blocks as opposed to having them modeled by the lookahead conformer blocks independently, risking low attention weights(as in *LSA* model). The new streaming hybrid lookahead Conformer encoder is shown on Figure 5. It is built using the following stack of layers: 2 causal Conformer blocks; lookahead Stacker block(with right context R=7); 3 causal conformer blocks; lookahead Stacker block(with right context R=6); 3 causal conformer blocks; lookahead Stacker block(with right context R=4); 5 causal Conformer blocks; 1 lookahead Conformer block (with right context R=4); 1 causal Conformer block; 1 lookahead Conformer block (with right context R=4) followed by 1 causal Conformer block. Note that this encoder uses three Stacking layers, thus it reduces the sampling rate by 8×. We call such a model *LSA_LS* (lookahead self-attention with a lookahead stacker).

3. Experimental results and discussion

3.1. Datasets and key metrics

All our models are trained on a parallel corpus, where the input utterances are from the entire set of the Librispeech [17] (960 hours of training data composed of "train-clean-100", "train-clean-360", "train-other-500") and the corresponding target utterances are the synthesis of their manual transcripts, using Google's Parallel WaveNet-based single-voice TTS [18]. We use training recipe described in [7], model training on 16 V3 TPUs takes 4 days. Parrotron effectively builds a many-to-one voice conversion system that normalizes speech from arbitrary speakers to a single canonical voice.

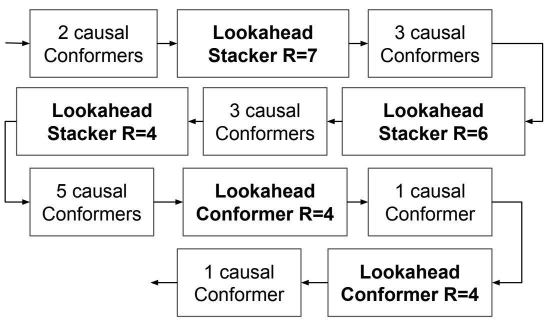

Figure 5: *Streaming hybrid lookahead Conformer encoder.*

Table 1: *10s benchmark of non-streaming Parrotron on Pixel4*

Model part	Latency[sec] float32	Latency[sec] int8
Encoder	2.8	2.6
Decoder	2.7	2.4
Vocoder sGL	2.4	
Vocoder nGL	7	

For WER evaluation, we use the Librispeech test-clean data set as in [7]. To evaluate the quality of our speech conversion systems, we use Google's state-of-the-art ASR to automatically transcribe the converted speech and then calculate and compare Word Error Rates (WERs) across systems, as in [6, 7]. As in [6] we use WER as measure of speech intelligibility. Parrotron allows users to communicate not only with humans but rather with any speech-enabled devices, such as Google Home, Amazon Alexa, Siri. Thus ASR WER is a crucial metric to evaluate given the application.

Another important user facing metric is a *total delay* described in section 2.1. All benchmarks are done with TFLite [19] single-threaded on a Pixel4 CPU, using [20] to build our models.

3.2. Non streaming *Base* model vs streaming models

Due to out-of-memory, we are unable to benchmark *Base* model [7] end-to-end on Pixel4 [21]. Therefore, we divide the model into three parts: encoder, decoder and vocoder and benchmark them separately. We process 10 seconds of audio which generates 6 seconds of synthesized audio. We apply int8 post training quantization and show both float32 and int8 latency in Table 1. In addition, we benchmark the non-streaming Griffin Lim (*nGL* used by Parrotron model in [7]) and the streaming Griffin Lim (*GL* used by Parrotron model in [14]). As shown in Table 1, we observe that running the model components on-device takes more than several seconds each, both when using float32 and int8 tensor operations. It makes it unusable for real STS applications(according to our user study acceptable latency has to be less around $300ms$).

Non streaming *Base* model has the best WER as shown on Table 2. It is a bit higher than WER reported in [7] since we do not do extra model tuning applied in [7]. We use a standard approach for all our experiments: train a model on training data, use validation data to select the best model and then test it on test data (no additional model tuning). Even though *Base* model has the best WER, it has the worst *total delay* of 7.9 seconds (sum of Encoder, Decoder and *GL* on Table 1) to process 10 seconds of input audio. For such a model *total delay* goes up

Table 2: *Base model vs streaming model accuracy and total delay of processing 10 seconds audio*

Model	**WER[%]**	**total delay float [ms]**
Base [7]*	14.7	7900
Streaming LSA_LS	15.3	400
Streaming LSA	16.4	440
Streaming Causal Base	19.2	0

with the increase of the length of the input audio, but for below streaming models *total delay* does not depend on input audio length.

We benchmarked streaming aware models presented in sections 2.2, 2.3 and 2.4 (with streaming Griffin-Lim vocoder [14]). As expected, the streaming *Causal Base* model has the best *total delay* equal 0ms and the worst WER equal 19.2%. Hybrid lookahead model *LSA_LS*(WER=15.3%) outperforms both lookahead *LSA*(WER=16.4%) and Causal models. It proves that combination of lookahead Stacker with lookahead local self attention (*LSA_LS* model) is superior when compared to lookahead local self attention (*LSA* model) in WER metric with the same *total delay* budget. Audio samples generated with these models can be found here [22].

With the best streaming model *LSA_LS* the difference with non-streaming *Base* model becomes 0.6% absolute. This is while substantially reducing the total delay (down to 400ms) when compared to the *Base* model 7400ms. Using this encoder with our streaming decoder and vocoder seems to build our optimal streaming-aware speech conversion system.

3.3. Streaming decoder benchmarks

Streaming decoder *sDec* takes data from "Encoded sequence" (as shown on Figure 1) then decodes it and passes it through the streaming vocoder. To stream the vocoder, we experiment with two streaming versions of vocoders: Streaming Griffin-Lim (denoted *GL*) and Streaming MelGAN (denoted *MG*), described in [14]. In all our experiments, both of these vocoders look ahead into 1 hop size, hence they have a delay of 12.5ms. To optimize the latency of *MG*, we process two frames at once, employing streaming with external states as discussed in [23].

In Table 3, we benchmark the latency, Real Time Factor (RTF is the ratio of the input duration to processing time), the TFLite model size and memory footprint when using the streaming decoder for both vocoders. To reduce the model size, we apply int8 post training quantization and report the impact on size and latency. In all these experiments, the streaming decoder generates two frames per call and the vocoder converts them into 25ms of audio.

We observe that the model size of *GL* is 0.1MB, which is a key advantage over *MG* (25MB). While *MG* requires a larger memory footprint, it is fully convolutional, allowing us to process multiple samples per streaming inference step in parallel – a major latency advantage over sequential vocoders, such as *GL* and WaveRNN [24]. We also find that the use of int8 version of the decoder exert substantial improvements for all metrics over float32 (as shown on Table 3).

3.4. Quantization of streaming model with *LSA_LS* encoder

We now explore the use of weight and activation quantization to reduce the memory footprint and latency of our best streaming model with *LSA_LS* encoder, shown on Figure 5. We apply int8 post training quantization [25]. To further reduce the encoder size, we perform a hybrid approach of int4 weight and int8 activation quantization aware training as in [26] for our encoder.

Table 3: *25ms benchmark of streaming decoder sDec with vocoders (GL, MG) on Pixel4*

	float32		int8	
	sDec+GL	*sDec+MG*	*sDec+GL*	*sDec+MG*
Latency[ms]	16.0	13.4	13.6	11.0
RTF	1.6x	1.9x	1.8x	2.3x
Size [MB]	122	147	30	55
Memory[MB]	139	166	44	64

Table 4: *80ms benchmark of streaming encoder on Pixel4*

	float32	int8 (int4*)
Latency[ms]	40	32
RTF	2x	2.5x
Model size [MB]	436	111 (70* with int4)
Memory size [MB]	905	186

We first benchmark our most accurate streaming-aware encoder *LSA_LS* on Pixel4 alone. As shown in Table 4, the latency of processing 80ms of audio (our streaming chunk) is only 40ms using float32 and 32ms using int8 quantization. This also leads to an encoder size of about 70MB (int4). We observe that float and quantized encoders can run in real time: RTF $\geq$ 2x.

We now evaluate the impact of quantization on accuracy of our best streaming model with *LSA_LS* encoder. We report the WER for both int4 and int8 quantized *LSA_LS* encoder, while always using our int8 streaming decoder (*sDec*), with streaming vocoders *GL* and *MG*. The results are shown in Table 5. Quantization helped to reduce total delay from 400ms(on Table 2) to 320ms as shown on Table 5. We observe that the use of *GL* always exerts significantly better quality than *MG*. We only lose 0.2% in WER if we use int4 quantization for our encoder. We find that quantization of both encoder and decoder does not substantially affect quality.

Table 5: *WERs of the STS conversion system with streaming quantized encoder LSA_LS decoder and vocoders.*

	GL	*MG*	total delay int8 [ms]
LSA_LS int8, sDec int8	15.4	15.9	320
LSA_LS int4, sDec int8	15.6	15.8	320

4. Conclusion

We have presented an on-device streaming aware STS conversion model. Our proposed system consists of a streaming-aware encoder, a streaming decoder and a streaming vocoder. We have explored a novel Conformer encoder(*LSA_LS*) based on look-ahead self-attention and look-ahead stacker. It improves quality compared to a fully casual model and lookahead self-attention(*LSA*), while also substantially minimizes the *total delay* when compared to a full context non-streaming *Base* model. Studying a variety of quantization configurations, we have discussed a hybrid approach which quantizes the model weights using int4 and int8 to enable us to deploy this model locally on-device. Our best configuration runs with a Real-Time-Factor of 2$\times$, with a delay of 320ms, to start emitting audio as soon as the speaker stops speaking, whereas the non-streaming model requires a 7-second delay. We also obtain a significant error reduction in comparison to a fully casual model, while only 0.7% error degradation when compared to a full context non-streaming *Base* model. We suspect that this is an acceptable trade-off for deploying such a streaming STS conversion system locally on mobile device.

5. References

[1] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang, Q. Liang, D. Bhatia, Y. Shangguan, B. Li, G. Pundak, K. C. Sim, T. Bagby, S.-y. Chang, K. Rao, and A. Gruenstein, "Streaming end-to-end speech recognition for mobile devices," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6381–6385.

[2] K. Kim, S. Jung, J. Lee, M. Han, C. Kim, K. Lee, D. Gowda, J. Park, S. Kim, S. Jin, Y. Lee, J. Yeo, and D. Kim, "Attention based on-device streaming speech recognition with large speech corpus," in *IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2019, Singapore, December 14-18, 2019*. IEEE, 2019, pp. 956–963.

[3] Z. Huang, H. Li, and M. Lei, "Devicetts: A small-footprint, fast, stable network for on-device text-to-speech," *CoRR*, vol. abs/2010.15311, 2020.

[4] S. Achanta, A. Antony, L. Golipour, J. Li, T. Raitio, R. Rasipuram, F. Rossi, J. Shi, J. Upadhyay, D. Winarsky, and H. Zhang, "On-device neural speech synthesis," in *IEEE Automatic Speech Recognition and Understanding Workshop, ASRU 2021, Cartagena, Colombia, December 13-17, 2021*. IEEE, 2021, pp. 1155–1161.

[5] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015.

[6] F. Biadsy, R. J. Weiss, P. J. Moreno, D. Kanvesky, and Y. Jia, "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*, G. Kubin and Z. Kacic, Eds. ISCA, 2019, pp. 4115–4119.

[7] Z. Chen, B. Ramabhadran, F. Biadsy, X. Zhang, Y. Chen, L. Jiang, F. Chu, R. Doshi, and P. J. Moreno, "Conformer parrotron: A faster and stronger end-to-end speech conversion and recognition model for atypical speech," in *Interspeech 2021, 22nd Annual Conference of the International Speech Communication Association, Brno, Czechia, 30 August - 3 September 2021*, H. Hermansky, H. Cernocký, L. Burget, L. Lamel, O. Scharenborg, and P. Motlícek, Eds. ISCA, 2021, pp. 4828–4832.

[8] F. Biadsy, Y. Chen, X. Zhang, O. Rybakov, A. Rosenberg, and P. J. Moreno, "A scalable model specialization framework for training and inference using submodels and its application to speech model personalization," in *Interspeech 2022, 23rd Annual Conference of the International Speech Communication Association, Incheon, Korea, 18-22 September 2022*, H. Ko and J. H. L. Hansen, Eds. ISCA, 2022, pp. 5125–5129.

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

[10] K. Sudoh, T. Kano, S. Novitasari, T. Yanagita, S. Sakti, and S. Nakamura, "Simultaneous speech-to-speech translation system with neural incremental asr, mt, and tts," 2020.

[11] X. Ma, J. M. Pino, and P. Koehn, "Simulmt to simulst: Adapting simultaneous text translation to end-to-end simultaneous speech translation," in *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing, AACL/IJCNLP 2020, Suzhou, China, December 4-7, 2020*, K. Wong, K. Knight, and H. Wu, Eds. Association for Computational Linguistics, 2020, pp. 582–587.

[12] J. Shen, R. Pang, R. J. Weiss, M. Schuster, N. Jaitly, Z. Yang, Z. Chen, Y. Zhang, Y. Wang, R. Ryan, R. A. Saurous, Y. Agiomyrgiannakis, and Y. Wu, "Natural TTS synthesis by conditioning wavenet on MEL spectrogram predictions," in *2018 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2018, Calgary, AB, Canada, April 15-20, 2018*. IEEE, 2018, pp. 4779–4783.

[13] A. Gulati, J. Qin, C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu, and R. Pang, "Conformer: Convolution-augmented transformer for speech recognition," in *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, H. Meng, B. Xu, and T. F. Zheng, Eds. ISCA, 2020, pp. 5036–5040.

[14] O. Rybakov, M. Tagliasacchi, Y. Li, L. Jiang, X. Zhang, and F. Biadsy, "Real time spectrogram inversion on mobile phone," *CoRR*, vol. abs/2203.00756, 2022.

[15] Q. Zhang, H. Lu, H. Sak, A. Tripathi, E. McDermott, S. Koo, and S. Kumar, "Transformer transducer: A streamable speech recognition model with transformer encoders and RNN-T loss," in *2020 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2020, Barcelona, Spain, May 4-8, 2020*. IEEE, 2020, pp. 7829–7833.

[16] A. Tripathi, J. Kim, Q. Zhang, H. Lu, and H. Sak, "Transformer transducer: One model unifying streaming and non-streaming speech recognition," *CoRR*, vol. abs/2010.03192, 2020.

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

[18] 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, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 3915–3923.

[19] "Tflite," https://www.tensorflow.org/lite.

[20] J. Shen, P. Nguyen, Y. Wu, Z. Chen *et al.*, "Lingvo: a modular and scalable framework for sequence-to-sequence modeling," 2019.

[21] "Pixel4," https://en.wikipedia.org/wiki/Pixel_4.

[22] "Audio examples," https://github.com/google-research/google-research/tree/master/stream_s2s.

[23] O. Rybakov, N. Kononenko, N. Subrahmanya, M. Visontai, and S. Laurenzo, "Streaming keyword spotting on mobile devices," in *Interspeech 2020, 21st Annual Conference of the International Speech Communication Association, Virtual Event, Shanghai, China, 25-29 October 2020*, H. Meng, B. Xu, and T. F. Zheng, Eds. ISCA, 2020, pp. 2277–2281.

[24] N. Kalchbrenner, E. Elsen, K. Simonyan, S. Noury, N. Casagrande, E. Lockhart, F. Stimberg, A. van den Oord, S. Dieleman, and K. Kavukcuoglu, "Efficient neural audio synthesis," in *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018*, ser. Proceedings of Machine Learning Research, J. G. Dy and A. Krause, Eds., vol. 80. PMLR, 2018, pp. 2415–2424.

[25] "Tflite quantization," https://www.tensorflow.org/lite/performance/post_training_quantization.

[26] S. Ding, P. Meadowlark, Y. He, L. Lew, S. Agrawal, and O. Rybakov, "4-bit conformer with native quantization aware training for speech recognition," in *Interspeech 2022, 23rd Annual Conference of the International Speech Communication Association, Incheon, Korea, 18-22 September 2022*, H. Ko and J. H. L. Hansen, Eds. ISCA, 2022, pp. 1711–1715.