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

# Low Latency Speech Recognition using End-to-End Prefetching

*Shuo-Yiin Chang, Bo Li, David Rybach, Yanzhang He, Wei Li, Tara Sainath, Trevor Strohman*

Google Inc., U.S.A

{shuoyiin, boboli, rybach, yanzhanghe, mweili, tsainath, strohman}@google.com

## Abstract

Latency is a crucial metric for streaming speech recognition systems. In this paper, we reduce latency by fetching responses early based on the *partial* recognition results and refer to it as *prefetching*. Specifically, prefetching works by submitting partial recognition results for subsequent processing such as obtaining assistant server responses or second-pass rescoring *before* the recognition result is finalized. If the partial result matches the final recognition result, the early fetched response can be delivered to the user instantly. This effectively speeds up the system by saving the execution latency that typically happens after recognition is completed.

Prefetching can be triggered multiple times for a single query, but this leads to multiple rounds of downstream processing and increases the computation costs. It is hence desirable to fetch the result sooner but meanwhile limiting the number of prefetches. To achieve the best trade-off between latency and computation cost, we investigated a series of prefetching decision models including decoder silence based prefetching, acoustic silence based prefetching and end-to-end prefetching.

In this paper, we demonstrate the proposed prefetching mechanism reduced latency by ∼200 ms for a system that consists of a streaming first pass model using recurrent neural network transducer and a non-streaming second pass rescoring model using Listen, Attend and Spell. We observe that the end-to-end prefetching provides the best trade-off between cost and latency and is 120 ms faster compared to silence based prefetching at a fixed prefetch rate.

## 1. Introduction

In a typical streaming speech recognition system, the trade-off between word error rate (WER) and latency is controlled by the *endpointer* [1, 2]. Here we term it as a *microphone closer* to avoid ambiguity. Microphone closer is responsible for determining when the user has finished speaking to ensure a natural and fast voice interaction. The system makes a series of binary decisions: to wait further for more speech, or to stop listening and submit the audio for subsequent processing. Each microphone closing decision is irrevocable, therefore the errors brought by the microphone closer have a dramatic impact on speech recognition accuracy or user experience as it can either cut off users or introduce latency.

To achieve a certain WER, we normally need to sacrifice latency. One way to improve latency is to develop models that have better latency and WER trade-offs [1, 3]. An alternative way is to fetch results "before" the microphone is closed and the final recognition result is ready. We refer to this mechanism as *prefetching* and the corresponding component as a *prefetcher*. Prefetching works by triggering downstream processes (e.g. web search or assistant server) based on partial recognition results - before having the finalized recognition hypothesis. If the partial result matches the final recognition result after the microphone is closed, i.e. a correct prefetching, we could have the final response ready by the time the system finishes the recognition. Hence, a correct prefetching could save the execution latency while the system is still listening. This can potentially bring in substantial latency savings.

Unlike a microphone closer, a system could generate multiple prefetches in a single query. An early prefetch can potentially bring in large latency gains. But it also has the risk of triggering the downstream tasks with a wrong recognition result, which renders the prefetch a waste of computation. If generated a lot, such premature prefetches can dramatically increase backend server load. The trade-off for a prefetcher is between the average number of prefetches per utterance, referred to as the prefetch rate, and the latency improvements. An aggressive prefetching decision provides faster response at the expense of higher prefetch rate while late prefetches have limited improvements on the total latency.

End-to-end (E2E) models [4, 5, 6, 7, 8, 9, 10, 11, 12] have gained large popularity for ASR over the last few years. These models replace components of a conventional ASR system, namely an acoustic, pronunciation, language models and microphone closer with a single neural network. They have already been shown to yield better quality and latency than conventional ASR systems [4]. This paper presents an efficient prefetching system that has a better trade-off between the prefetch rate and latency. Most importantly, with such a prefetcher, we can further improve the latency of E2E models [4, 3].

## 2. Prefetching

### 2.1. Prefetching for Voice Search/Assistants

Prefetching is a common technique used for web search applications [13, 14] to speed up interactions. It allows applications to fetch the necessary resources that a user might access in the near future. Figure 1 depicts a prefetching application in a voice search/assistant system where two partial recognition results are submitted to the server. The backend server then computes the latest response and fetches the necessary resources. In Figure 1, the first prefetch is a premature one and the computation of search/assistant server is wasted as the response is discarded. The second prefetch is based on a hypothesis that matches the final recognition result. When the mic-closing decision is emitted and the recognition is completed, there is no need to submit the final recognition result to the server. The response received from the second prefetch is directly presented to the user once done. Hence, the total latency, $T_{\texttt{total}}$, for a voice query with $\texttt{N}$ prefetches ($\texttt{PF}_1, \ldots, \texttt{PF}_\texttt{N}$) can be calculated as:

$$T_{\texttt{total}} = \min(T_{\texttt{PF}_1}, \ldots, T_{\texttt{PF}_\texttt{N}}, T_{\texttt{mic\_closer}}) + T_{\texttt{server}} \quad (1)$$



http://dx.doi.org/10.21437/Interspeech.2020-1898

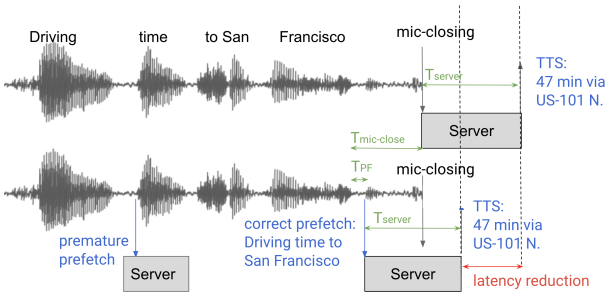


Figure 1: *Prefetching saves the total latency by hiding part of the server execution latency.*

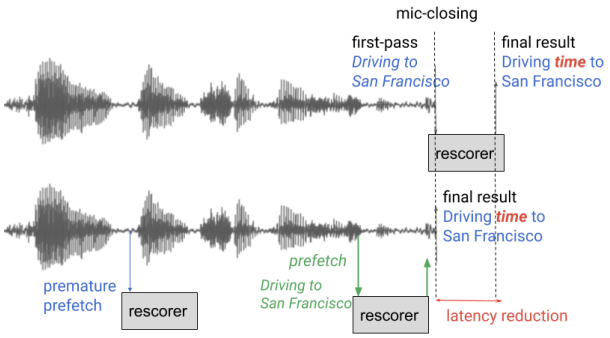


Figure 2: *Prefetching saves total latency for a two-pass E2E model by hiding the second pass rescoring latency.*

where

$$T_{\texttt{PF}_\texttt{i}} = \begin{cases} t_{\texttt{PF}_\texttt{i}} - t_{\texttt{<\/s>}}, & \text{if } \texttt{PF}_\texttt{i} \text{ is the correct prefetch} \\ \infty, & \text{otherwise} \end{cases}, \quad (2)$$

$$T_{\texttt{mic\_closer}} = t_{\texttt{mic\_closer}} - t_{\texttt{<\/s>}}. \quad (3)$$

The $i$-th prefetch latency, $T_{\texttt{PF}_\texttt{i}}$ for $\texttt{i} \in 1, \ldots, \texttt{N}$, is measured by the time difference between when the user finishes speaking, $t_{\texttt{<\/s>}}$, and the time when the prefetch is triggered, $t_{\texttt{PF}_\texttt{i}}$, if it is the correct prefetch. Otherwise, the prefetch is simply discarded and $T_{\texttt{PF}_\texttt{i}}$ is set to $\infty$. A prefetch is correct only if the partial recognition result it used matches the final recognition result. If a correct prefetch is missing i.e. none of the partial results used for prefetch matches the final result, the latency then falls back to the microphone closing latency, $T_{\texttt{mic\_closer}}$, which is the time difference between $t_{\texttt{<\/s>}}$ and the microphone closing time $t_{\texttt{mic\_closer}}$. For either case, we also need to include the server execution latency $T_{\texttt{server}}$ which accounts for the time used to compute and fetch responses from the server. Besides the latency, another important metric is the prefetch rate which is the number of prefetches for each query. In the above example, the prefetch rate is N.

### 2.2. Prefetching for Second-Pass Rescoring

Motivated by section 2.1, we propose adopting the prefetching mechanism to reduce latency for two-pass ASR models, e.g. [4, 3, 15]. In the two-pass E2E model proposed in [15], a recurrent neural network transducer (RNN-T) model is used as the streaming first pass model to generate initial hypotheses and a Listen, Attend and Spell (LAS) model is used as the non-streaming second pass model to rescore those hypotheses. This model was shown to abide by user interaction constraints, and offer better performance than a conventional model [4]. Due to the second pass rescoring, additional latency was brought to the system to achieve the recognition performance gains. Specifically, the second pass LAS rescoring brings 10% relative WER improvements at the expense of roughly 200 ms latency increases.

To reduce the latency increase caused by the second-pass rescoring, we propose using prefetching to trigger early rescoring as shown in Figure 2. Similar to prefetching in voice search/assistant servers described in section 2.1, partial result is submitted to the LAS rescorer before having the finalized first-pass result. The rescorer once triggered computes the LAS outputs based on the current lattice generated by the first pass RNN-T model and the audio received so far. In Figure 2, the lattice used by the second prefetch differs from the final one only by silence suffices, therefore we could approximate the first-pass final lattice by using the one at prefetch. At the mic-closing time, this rescored top hypothesis is returned to the user immediately when it is ready without conducting another LAS rescoring with the final lattice.

For simplification, we consider the partial lattice as a correct prefetch and skip rescoring at final if the corresponding top hypothesis matches final hypothesis. Hence, the proposed scenario has exactly the same mechanism as the system in Fig. 1 except that the downstream processor is changed from a server to a rescorer. Therefore, total latency can be calculated as Eq. (1) to (3) while $T_{\texttt{server}}$ is replaced by $T_{\texttt{rescorer}}$ representing computation latency of the second-pass rescoring.

## 3. Models

This section describes different models for doing prefetching.

### 3.1. Decoder Silence based Prefetching

A simple approach to do prefetching is to submit the current partial results after the decoder observing a fixed interval of silence, e.g. 200 ms of silence. Silence interval is computed as the duration since the last word. Based on the current state of the decoder we select a minimum wait-time and if the decoder reports a longer-than-that silence on the best path, prefetching decision is declared. The decision of prefetching is calculated at every frame while redundant prefetch is disabled if the partial result is unchanged compared to the most recent prefetch.

### 3.2. Acoustic Silence based Prefetching

A voice activity detector (VAD) [16, 17, 18, 19] is an alternative way to make silence based prefetching decisions. It is directly trained to classify each input frame to be either silence or not. The model is usually optimized using the cross entropy loss function with a frame-level speech/non-speech forced alignment as the ground truth target. A VAD is typically more accurate to align speech and silence compared to the silence interval observed by E2E ASR models.

However, a VAD cannot discriminate between a pause within an utterance and the sentence end silence. Partial results before the middle pauses do not generate correct prefetching. Therefore, we expanded the output vocabulary of the basic VAD to classify four targets, namely speech, initial silence, intermediate silence and final silence [2, 20] as Fig. 3(b). At a given time frame, a recurrent neural network model makes its predictions and prefetch decisions are made by thresholding the poste-

riors of final silence. The recurrent model's sequential modeling capability and the fine-grained silence targets bring the capability to capture the potential acoustic cues such as filler sounds and speaking rate. Such information is most useful for discriminating initial, intermediate and final silence which would be hard for the basic VAD models.

### 3.3. End-to-end Prefetching

For better microphone closing decisions, [1] extends the RNN-T's output vocabulary with a special token </s> indicating the end of the utterance as part of the output label sequence. The RNN-T model directly makes microphone closing decision by emitting </s> .

In this work, we leverage on the </s> prediction of the RNN-T microphone closer for prefetching decisions and refer to it as the end-to-end prefetching. Unlike microphone closing decision, we want to predict </s> for every partial result but avoid changing decoding. To achieve that, for each frame before </s> shows on the top beam, we artificially add </s> token at the end of the current hypothesis to compute the probability of </s> given the observations and the best path so far. Unlike using </s> for microphone closer, this computation allows us to predict how likely the query is completed at each time frame and the corresponding partial result while decoding following frames. If the probability is greater than a predefined threshold $\theta$, the system declares a prefetching decision as:

$$p(\texttt{</s>}|\mathbf{x}_1, \ldots, \mathbf{x}_t, y_0^{\text{RNN-T}}, \ldots, y_{t-1}^{\text{RNN-T}}) \geq \theta, \quad (4)$$

where we denoted input frames as $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ and the top hypothesis as $\{y_1, \ldots, y_U\}$. </s> is only for computing the prefetching decision and is not actually added to the hypothesis.

For models directly outputting </s> , a premature </s> prediction can result in deletion errors, while late predictions of </s> can increase latency as </s> is used to inform the system when the speech ends. [3] addresses these issues by applying additional early and late penalties on the </s> token (Equation (5)). During training for every input frame in $\{\mathbf{x}_1, \ldots, \mathbf{x}_T\}$ and every label $\{y_1, \ldots, y_U\}$, RNN-T computes a $U \times T$ matrix $\mathbf{P}_{\text{RNN-T}}(\mathbf{y}|\mathbf{x})$, which is used in the training loss computation. For the RNN-T microphone closer, the last label $y_U$ is always </s> . We denote $t_{\texttt{</s>}}$ as the frame index after the last non-silence phoneme, which is obtained from the forced alignment of the audio with a conventional model. The RNN-T log-probability $\log \mathbf{P}_{\text{RNN-T}}(y_U|\mathbf{x})$ is modified to include a penalty at each time step $t$ for predicting </s> too early or too late. $t_{\text{buffer}}$ gives a grace period after the reference $t_{\texttt{</s>}}$ before the late penalty is applied. $\alpha_{\text{early}}$ and $\alpha_{\text{late}}$ are scales on the early and late penalties respectively. All hyper parameters are tuned experimentally. This model is adopted for generating prefetching decisions.

$$\log \mathbf{P}_{\text{RNN-T}}(y_U|\mathbf{x}_t) \; -= \big( \max(0,\; \alpha_{\text{early}}(t_{\texttt{</s>}} - t)) + \max(0,\; \alpha_{\text{late}}(t - t_{\texttt{</s>}} - t_{\text{buffer}})) \big) \quad (5)$$

## 4. Experimental Setup

### 4.1. Data and Model Configurations

Our experiments are based on E2E ASR system that consists of a streaming first pass model using RNN-T and a non-streaming second pass rescoring model LAS [4]. All models are trained using a 128-dimensions log-mel feature frontend [4]. The features are computed using 32 msec windows with a 10 msec hop.

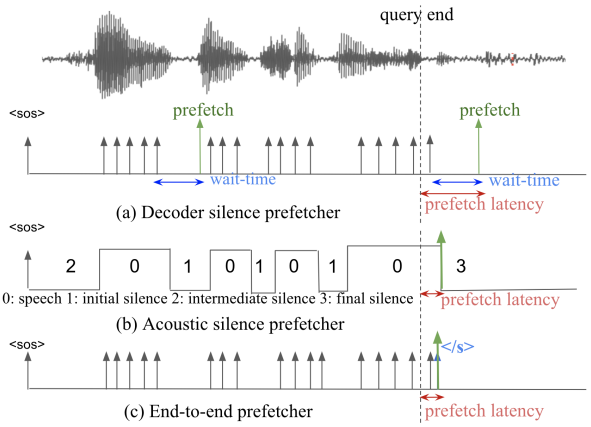


Figure 3: *Prefetchers.*

Features from 4 contiguous frames are stacked to form a 512 dimensional input representation, which is further subsampled by a factor of 3 and passed to the model. Following [4], all LSTM layers in the model are unidirectional, with 2,048 units and a projection layer with 640 units. The shared encoder consists of 8 LSTM layers, with a time-reduction layer after the 2nd-layer. The RNN-T decoder consists of a prediction network with 2 LSTM layers, and a joint network with a single feed-forward layer with 640 units. The additional LAS-specific encoder consists of 2 LSTM layers. The LAS decoder consists of multi-head attention [21] with 4 attention heads, which is fed into 2 LSTM layers. Both decoders are trained to predict 4,096 word pieces [22]. The RNN-T model has 120M parameters. The additional encoder and the LAS decoder have 57M parameters. All parameters are quantized to 8-bit fixed-point [5]. The total model size in memory/disk is 177MB. All models are trained in Tensorflow [23] using the Lingvo [24] toolkit on $8 \times 8$ Tensor Processing Units (TPU) slices with a global batch size of 4,096.

Models are trained on a diverse training set [4] with SpecAugment [25] and multi-condition training (MTR) [26, 27] and random data down-sampling to 8kHz [28] is also used to further increase data diversity. Noisy data is generated at signal-noise-ratio (SNR) from 0 to 30 dB, with an average SNR of 12 dB, and with T60 times ranging from 0 to 900 msec, averaging 500 msec. Noise segments are sampled from YouTube and daily life noisy environmental recordings. Both 8 kHz and 16 kHz versions of the data are generated, each with equal probability, to make the model robust to varying sample rates. The test set includes 14K Voice-search utterances extracted from Google traffic which are anonymized and hand-transcribed.

## 5. Results

In this section, we first present a series of experimental results comparing different prefetchers discussed in Section 3. We then apply the best prefetcher to reduce latency of a two-pass E2E system that consists of a streaming first pass model using RNN-T and a non-streaming second pass rescoring model LAS.

### 5.1. Metrics

In practice, we would like to have a fast prefetching for lower latency but also want to avoid a high prefetch rate (PFR) i.e. average number of prefetches per utterance. An aggressive decision provides faster responds at the expense of computation

Table 1: *Prefetch latency and coverage rate of the three different prefetch models at a prefetch rate of 1.25.*

| **Exp.** | **PF50** (ms) | **PF90** (ms) | **coverage** |
|---|---|---|---|
| Decoder silence prefetch<br>(rel. imp. to mic-closing) | 430<br>(10) | 740<br>(20) | 11% |
| Acoustic silence prefetch<br>(rel. imp. to mic-closing) | 330<br>(120) | 610<br>(150) | 84% |
| E2E prefetch<br>(rel. imp. to mic-closing) | **320**<br>**(130)** | **490**<br>**(270)** | **94%** |

cost while a lower prefetching rate could limit the latency gains brought by prefetching. To understand this trade-off, we depict the interactions between prefecth latency and prefetch rate by ROC curves.

Prefetch latency is measured by the minimum latency of a correct prefetch relative to the end of the speech. We consider the prefetch as a correct prefetch if the partial result used matches the final result. If a correct prefetch is missing, the latency falls backs to the microphone closing latency as Eq. (1). We determine the "true" end of speech by running a forced alignment. We also evaluate prefetch coverage representing the percentage of the test data actually receiving a correct prefetch.

### 5.2. Prefetch rate vs latency

Figure 4 demonstrates the ROC curves of median and 90th percentile prefetch latency against the prefetch rate respectively. The three prefetching models discussed in previous sections are presented based on an E2E RNN-T system [4] with a WER of 6.4%. Results show that the E2E prefetching and the acoustic silence prefetching have significantly better efficiency than decoder silence prefetching. The prefetches generated based on decoder silence threshold are mostly premature prefetches as could be observed by the low coverage rate shown in Table 1.

For median prefetch latency shown in Figure 4, the E2E prefetching and the acoustic silence based prefetching have similar ROC curves while the E2E prefetching is 10∼30 ms faster than the acoustic one with a prefetch rate below 1.5. In terms of 90 percentile prefetch latency, Figure 4 shows that the E2E prefetching is the most efficient approach of doing prefetch. It can provide over 100 ms latency improvement over the acoustic silence prefetching with a prefetch rate below 1.5. In Table 1, we report prefetching metrics for the three approaches at a PFR of 1.25. Table 1 also shows that the latency improvement relative to microphone closing which is the total latency reduction brought by prefetching. The E2E prefetching has the best latency, which is 120 ms faster than the acoustic silence prefetching and provides a 270 ms latency saving relative to microphone closing. This means the system can save 270 ms for downstream processing by using the E2E prefetching. Table 1 also shows that 94% of the utterances have a correct prefetch when using the E2E prefetching.

### 5.3. Latency of LAS Rescoring

In this section, we report the quality and latency of a two pass system that uses RNN-T as the first-pass model and LAS as the second-pass rescorer. The RNN-T first-pass model is frozen during the training of the LAS rescorer using a cross entropy loss. During inference, the LAS rescoring is done on hypotheses generated by the first-pass RNN-T model. In order to evaluate the total latency of the two-pass model, we benchmark the wall time when we run the recognition system with around 100 search utterances on a Google Pixel-4 phone. Inference is done on the device's CPU.

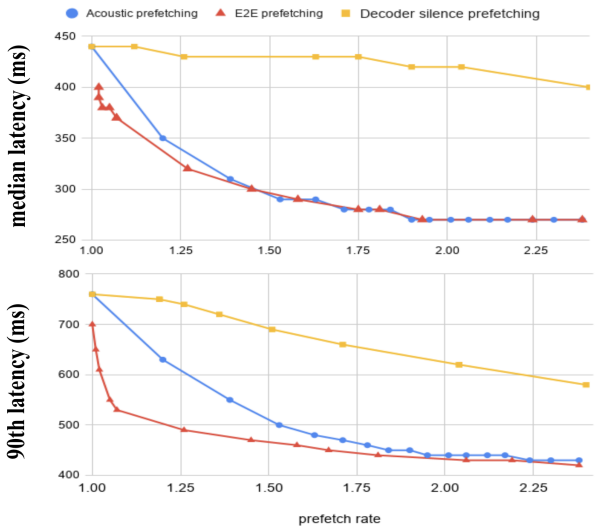


Figure 4: *Median (above) and 90th percentile (below) prefetch latency vs prefetch rate for different systems.*

Table 2: *ASR quality and total latency of a two-pass model.*

| **Exp.** | **WER** (%) | **Total Latency** (ms) | |
|---|---|---|---|
| | | median | 90th percentile |
| RNN-T | 6.4 | **586** | **1060** |
| RNN-T + LAS<br>No prefetching | **6.0** | 670 | 1286 |
| RNN-T + LAS<br>E2E prefetching | **6.0** | 612 | 1073 |

Table 2 shows that LAS rescoring reduces the WER from 6.4% to 6.0%. However, we also observe that the second-pass rescoring increase roughly 100 ms of median latency and 200 ms of 90th percentile latency due to computation cost of LAS in Table 2. To reduce the computation latency of the second-pass rescoring, we use the prefetching mechanism as described in Section 2.2 where the prefetch decision is made by the E2E prefetching with a PFR of 1.25. Table 2 demonstrates the latency increase could be almost completely saved by using the E2E prefetching. In the final two-pass RNN-T + LAS system, we can reduce WER with similar total latency as RNN-T alone.

## 6. Conclusions

In this paper, we demonstrate that the end-to-end prefetching reduces 200 ms for a system that consists of a streaming first pass model using RNN-T and a non-streaming second pass rescoring model using LAS. By comparing to other approaches, the end-to-end prefetching provides the best trade-off between the prefetch rate and the latency, showing a 120 ms improvement over the acoustic silence prefetching and a 270 ms of 90th percentile latency saving relative to microphone closing.

# 7. References


[1] S.-Y. Chang, R. Prabhavalkar, Y. He, T. N. Sainath, and G. Simko, "Joint endpointing and decoding with end-to-end models," in *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2019, pp. 5626–5630.

[2] S. Chang, B. Li, G. Simko, T. N. Sainath, and C. Parada, "End-point detection using grid long short-term memory networks for streaming speech recognition," in *Interspeech*, 2017.

[3] B. Li, S.-y. Chang, T. N. Sainath, R. Pang, Y. He, T. Strohman, and Y. Wu, "Towards fast and accurate streaming end-to-end asr," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6069–6073.

[4] T. N. Sainath, Y. He, B. Li, A. Narayanan, R. Pang, A. Bruguier, S.-y. Chang, W. Li, R. Alvarez, Z. Chen *et al.*, "A streaming on-device end-to-end model surpassing server-side conventional model quality and latency," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6059–6063.

[5] 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. Sim, T. Bagby, S. Chang, K. Rao, and A. Gruenstein, "Streaming End-to-end Speech Recognition For Mobile Devices," in *Proc. ICASSP*, 2019.

[6] C. Chiu, T. N. Sainath, Y. Wu, R. Prabhavalkar, P. Nguyen, Z. Chen, A. Kannan, R. J. Weiss, K. Rao, N. Jaitly, B. Li, and J. Chorowski, "State-of-the-art speech recognition with sequence-to-sequence models," in *Proc. ICASSP*, 2018.

[7] A. Graves, "Sequence transduction with recurrent neural networks," *CoRR*, vol. abs/1211.3711, 2012.

[8] A. Graves, A.-R. Mohamed, and G. Hinton, "Speech recognition with deep neural networks," in *Proc. ICASSP*, 2012.

[9] K. Rao, H. Sak, and R. Prabhavalkar, "Exploring architectures, data and units for streaming end-to-end speech recognition with rnn-transducer," in *Proc. ASRU*, 2017, pp. 193–199.

[10] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell," *CoRR*, vol. abs/1508.01211, 2015.

[11] S. Kim, T. Hori, and S. Watanabe, "Joint CTC-attention based end-to-end speech recognition using multi-task learning," in *Proc. ICASSP*, 2017, pp. 4835–4839.

[12] C.-C. Chiu and C. Raffel, "Monotonic chunkwise alignments," in *Proc. ICLR*, 2017.

[13] L. A. Barroso, J. Dean, and U. Hölzle, "Web search for a planet: The google cluster architecture," *IEEE Micro*, vol. 23, pp. 22–28, 2003.

[14] S. Jonassen, B. B. Cambazoglu, and F. Silvestri, "Prefetching query results and its impact on search engines," in *Proceedings of the 35th international ACM SIGIR conference on Research and development in information retrieval*, 2012, pp. 631–640.

[15] T. Sainath, R. Pang, D. Rybach, Y. He, R. Prabhavalkar, W. Li, M. Visontai, Q. Liang, T. Strohman, Y. Wu, I. McGraw, and C. Chiu, "Two-Pass End-to-End Speech Recognition," in *Proc. Interspeech*, 2019.

[16] S. Chang, B. Li, G. Simko, T. N. Sainath, A. Tripathi, A. Oord, and O. Vinyals, "Temporal modeling using dilated convolution and gating for voice-activity-detection," in *Interspeech*, 2017.

[17] S. Thomas, G. Saon, M. V. Segbroeck, and S. Naranyanan, "Improvements to the IBM speech activity detection system for the darpa rats program," in *Proc. ICASSP*, 2015.

[18] M. G. et al., "All for one: feature combination for highly channel-degraded speech activity detection," in *Proc. Interspeech*, 2013.

[19] F. Eyben, F. Weninger, S. Squartini, and B. Schuller, "Real-life voice activity detection with LSTM recurrent neural networks and an application to hollywood movies," in *Proc. ICASSP*, 2013.

[20] G. Simko, M. ahannon, S. Chang, and C. Parada, "Improved end-of-query detection for streaming speech recognition," in *Interspeech*, 2017.

[21] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention Is All You Need," *CoRR*, vol. abs/1706.03762, 2017. [Online]. Available: http://arxiv.org/abs/1706.03762

[22] M. Schuster and K. Nakajima, "Japanese and Korean voice search," in *Proc. ICASSP*, 2012.

[23] M. Abadi et al., "Tensorflow: Large-scale machine learning on heterogeneous distributed systems," Available online: http://download.tensorflow.org/paper/whitepaper2015.pdf, 2015.

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

[25] D. S. Park, Y. Zhang, C.-C. Chiu, Y. Chen, B. Li, W. Chan, Q. V. Le, and Y. Wu, "Specaugment on large scale datasets," in *in Proc. ICASSP*, 2020.

[26] V. Peddinti, V. Manohar, Y. Wang, D. Povey, and S. Khudanpur, "Far-Field ASR Without Parallel Data." in *Proc. of Interspeech*, 2016.

[27] C. Kim, A. Misra, K. Chin, T. Hughes, A. Narayanan, T. N. Sainath, and M. Bacchiani, "Generation of Large-Scale Simulated Utterances in Virtual Rooms to Train Deep-Neural Networks for Far-Field Speech Recognition in Google Home," in *Proc. of Interspeech*, 2017.

[28] J. Li, D. Yu, J. Huang, and Y. Gong, "Improving Wideband Speech Rcognition using Mixed-bandwidth Training Data in CD-DNN-HMM," in *Proc. SLT*, 2012.