STREAMING END-TO-END SPEECH RECOGNITION FOR MOBILE DEVICES

Yanzhang He[*]*, Tara N. Sainath*[*]*, Rohit Prabhavalkar, Ian McGraw, Raziel Alvarez, Ding Zhao, David Rybach, Anjuli Kannan, Yonghui Wu, Ruoming Pang, Qiao Liang, Deepti Bhatia, Yuan Shangguan, Bo Li, Golan Pundak, Khe Chai Sim, Tom Bagby, Shuo-yiin Chang, Kanishka Rao, Alexander Gruenstein*

Google, Inc., USA

{yanzhanghe, tsainath}@google.com

ABSTRACT

End-to-end (E2E) models, which directly predict output character sequences given input speech, are good candidates for on-device speech recognition. E2E models, however, present numerous challenges: In order to be truly useful, such models must decode speech utterances in a streaming fashion, in real time; they must be robust to the long tail of use cases; they must be able to leverage user-specific context (e.g., contact lists); and above all, they must be extremely accurate. In this work, we describe our efforts at building an E2E speech recognizer using a recurrent neural network transducer. In experimental evaluations, we find that the proposed approach can outperform a conventional CTC-based model in terms of both latency and accuracy in a number of evaluation categories.

1. INTRODUCTION

The last decade has seen tremendous advances in automatic speech recognition (ASR) technologies fueled by research in deep neural networks [1]. Coupled with the tremendous growth and adoption of smartphones, tablets and other consumer devices, these improvements have resulted in speech becoming one of the primary modes of interaction with such devices [2, 3]. The dominant paradigm for recognizing speech on mobile devices is to stream audio from the device to the server, while streaming decoded results back to the user. Replacing such a server-based system with one that can *run entirely on-device* has important implications from a reliability, latency, and privacy perspective, and has become an active area of research. Prominent examples include *wakeword detection* (i.e., recognizing specific words or phrases) [4, 5, 6, 7], as well as large vocabulary continuous speech recognition (LVCSR) [8, 9].

Previous attempts at building on-device LVCSR systems have typically consisted of shrinking traditional components of the overall system (acoustic (AM), pronunciation (PM), and language (LM) models) to satisfy computational and memory constraints. While this has enabled parity in accuracy for narrow domains such as voice commands and dictation [9], performance is significantly worse than a large server-based system on challenging tasks such as voice search.

In contrast to previous approaches, we instead focus on building a streaming system based on the recent advances in end-to-end (E2E) models [10, 11, 12, 13, 14]. Such models replace the traditional components of an ASR system with a single, end-to-end trained, all-neural model which directly predicts character sequences, thus greatly simplifying training and inference. E2E models are thus extremely attractive for on-device applications.

[*]Equal contribution

Early E2E work examined connectionist temporal classification (CTC) [15] with grapheme or word targets [16, 17, 18, 19]. More recent work has demonstrated that performance can be improved further using either the recurrent neural network transducer (RNN-T) model [12, 20, 21] or attention-based encoder-decoder models [10, 13, 14, 22]. When trained on sufficiently large amounts of acoustic training data ($10,000$+ hours), E2E models can outperform conventional hybrid RNN-HMM systems [21, 14]. Most E2E research has focused on systems which process the full input utterance before producing a hypothesis; models such as RNN-T [12, 20] or streaming attention-based models (e.g., MoChA [22]) are suitable if streaming recognition is desired. Therefore, in this work, we build a streaming E2E recognizer based on the RNN-T model.

Running an end-to-end model on device in a *production environment* presents a number of challenges: first, the model needs to be at least as accurate as a conventional system, without increasing latency (i.e., the delay between the user speaking and the text appearing on the screen), thus running at or faster than real-time on mobile devices; second, the model should be able to leverage on-device user context (e.g., lists of contacts, song names, etc.) to improve recognition accuracy [23]; finally, the system must be able to correctly recognize the 'long tail' of possible utterances, which is a challenge for an E2E system trained to produce text directly in the *written domain* (e.g., `call two double four triple six five` → `call 244-6665`).

In order to achieve these goals, we explore a number of improvements to the basic RNN-T model: using layer normalization [24] to stabilize training; using large batch size [25]; using word-piece targets [26]; using a time-reduction layer to speed up training and inference; and quantizing network parameters to reduce memory footprint and speed up computation [27]. In order to enable contextualized recognition, we use a shallow-fusion approach [28, 29] to bias towards user-specific context, which we find is on-par with conventional models [9, 23]. Finally, we characterize a fundamental limitation of vanilla E2E models: their inability to accurately model the normalization of *spoken* numeric sequences in the correct *written* form when exposed to unseen examples. We address this issue by training the model on synthetic data generated using a text-to-speech (TTS) system [30], which improves performance on numeric sets by 18–36% relative. When taken together, these innovations allow us to decode speech twice as fast as real time on a Google Pixel phone, which improves word error rate (WER) by more than 20% relative to a conventional CTC embedded model on voice search and dictation tasks.

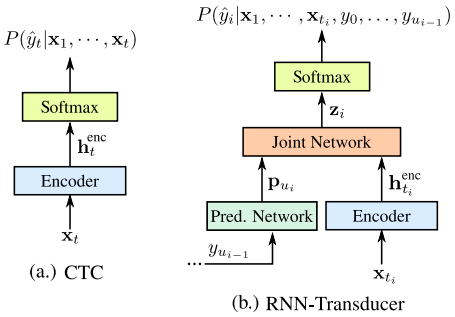

Fig. 1: A schematic representation of CTC and RNNT.

2. RECURRENT NEURAL NETWORK TRANSDUCER

Before describing the RNN-T model in detail, we begin by introducing our notation. We denote the parameterized input acoustic frames as $\mathbf{x} = (\mathbf{x}_1 \ldots \mathbf{x}_T)$, where $\mathbf{x}_t \in \mathbb{R}^d$ are 80-dimensional log-mel filterbank energies in this work ($d = 80$) and T denotes the number of frames in $\mathbf{x}$. We denote the ground-truth label sequence of length U as $\mathbf{y} = (y_1, \ldots, y_U)$, where $y_u \in \mathcal{Z}$ and where $\mathcal{Z}$ corresponds to context-independent (CI) phonemes, graphemes or word-pieces [26], in this work. We sometimes also use a special symbol, $y_0 = \langle \texttt{sos} \rangle$, which indicates the start of the sentence.

We describe the RNN-T [12, 20] model by contrasting it to a CTC [15] model. CTC computes the distribution of interest, $P(\mathbf{y}|\mathbf{x})$, by augmenting $\mathcal{Z}$ with an additional *blank* symbol, $\langle \texttt{b} \rangle$, and defining:

$$P(\mathbf{y}|\mathbf{x}) = \sum_{\hat{\mathbf{y}} \in \mathcal{A}_{\text{CTC}}(\mathbf{x}, \mathbf{y})} \prod_{t=1}^{T} P(\hat{y}_t|\mathbf{x}_1, \cdots, \mathbf{x}_t) \quad (1)$$

where $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_T) \in \mathcal{A}_{\text{CTC}}(\mathbf{x}, \mathbf{y}) \subset \{\mathcal{Z} \cup \langle \texttt{b} \rangle\}^T$ correspond to frame-level alignments of length T such that removing blanks and repeated symbols from $\hat{\mathbf{y}}$ yields $\mathbf{y}$. CTC makes a strong independence assumption that labels are conditionally independent of one another given acoustics. RNN-T removes this independence assumption by instead conditioning on the full history of previous non-blank labels:

$$P(\mathbf{y}|\mathbf{x}) = \sum_{\hat{y} \in \mathcal{A}_{\text{RNNT}}(\mathbf{x}, \mathbf{y})} \prod_{i=1}^{T+U} P(\hat{y}_i|\mathbf{x}_1, \cdots, \mathbf{x}_{t_i}, y_0, \ldots, y_{u_{i-1}}) \quad (2)$$

where $\hat{\mathbf{y}} = (\hat{y}_1, \ldots, \hat{y}_{T+U}) \in \mathcal{A}_{RNNT}(\mathbf{x}, \mathbf{y}) \subset \{\mathcal{Z} \cup \langle \texttt{b} \rangle\}^{T+U}$ are alignment sequences with T blanks and U labels such that removing the blanks in $\hat{\mathbf{y}}$ yields $\mathbf{y}$. Practically speaking this means that the probability of observing the ith label, $\hat{y}_i$, in an alignment, $\hat{\mathbf{y}}$, is conditioned on the history of non-blank labels, $y_1 \ldots y_{u_{i-1}}$, emitted thus far. Crucially, for both CTC and RNN-T we introduce one final conditional independence assumption: an alignment label $\hat{y}$ cannot depend on future acoustic frames. This enables us to build streaming systems that do not need to wait for the entire utterance to begin processing.

The conditional distributions for both models are parameterized by neural networks, as illustrated in Figure 1. Given the input features we stack unidirectional long short-term memory (LSTM) [31] layers to construct an *encoder*. For CTC the encoder is augmented with a final softmax layer that converts the encoder output into the relevent conditional probability distribution. The RNN-T, instead, employs a feed-forward joint network that accepts as input the results from both the encoder and a *prediction network* that depends only on label histories. The gradients required to train both models can be computed using the forward-backward algorithm [12, 15, 20].

3. REAL-TIME SPEECH RECOGNITION USING RNN-T

This section describes various architectural and optimization improvements that increase the RNN-T model accuracy and also allow us to run the model on device faster than real time.

3.1. Model Architecture

We make a number of architectural design choices for the encoder and prediction network in RNN-T in order to enable efficient processing on mobile devices. We employ an encoder network which consists of eight layers of uni-directional LSTM cells [31]. We add a projection layer [32] after each LSTM layer in the encoder, thus reducing the number of recurrent and output connections.

Motivated by [10, 33], we also add a time-reduction layer in the encoder to speed up training and inference. Specifically, if we denote the inputs to the time-reduction layer as $\mathbf{h}_1, \mathbf{h}_2, \cdots, \mathbf{h}_T$, then we concatenate together N adjacent input frames to produce $\lceil \frac{T}{N} \rceil$ output frames, where the $i+1$-th output frame is given by $[\mathbf{h}_{iN}; \mathbf{h}_{iN+1}; \cdots; \mathbf{h}_{(i+1)N-1}]$, thus effectively reducing the overall frame rate by a factor of N. The computational savings obtained using a time-reduction layer increase if it is inserted lower in the encoder LSTM stack. Applying the time-reduction layer to either model, which already has an input frame rate of 30ms, has different behaiviors. Specifically, we find that it can be inserted as low as after the second LSTM layer without any loss in accuracy for RNN-T, whereas adding it to the CTC phoneme models (with effective output frame rate $\geq$ 60ms) degrades accuracy.

3.2. Training Optimizations

In order to stabilize hidden state dynamics of the recurrent layers, we find it useful to apply layer normalization [24] to each LSTM layer in the encoder and the prediction network. Similar to [21], we train with word-piece subword units [26], which outperform graphemes in our experiments. We utilize an efficient forward-backward algorithm [25], which allows us to train RNN-T models on tensor processing units (TPUs) [34]. This allows us to train faster with much larger batch sizes than would be possible using GPUs, which improves accuracy.

3.3. Efficient Inference

Finally, we consider a number of runtime optimizations to enable efficient on-device inference. First, since the prediction network in RNN-T is analogous to an RNN language model, its computation is independent of the acoustics. We, therefore, apply the same state caching techniques used in RNN language models in order to avoid redundant computation for identical prediction histories. In our experiments, this results in saving 50–60% of the prediction network computations. In addition, we use different threads for the encoder and the prediction network to enable pipelining through asynchrony in order to save time. We further split the encoder execution over two threads corresponding to the components before and after the time-reduction layer, which balances the computation between the two encoder components and the prediction network. This results in a speed-up of 28% with respect to single-threaded execution.

3.4. Parameter Quantization

In order to reduce memory consumption, both on disk and at runtime, and to optimize the model's execution to meet real-time requirements, we quantize parameters from 32-bit floating-point precision into 8-bit fixed-point, as in our previous work [9]. In contrast to [9], we now use a simpler quantization approach that is linear (as before) but no

longer has an explicit "zero point" offset, thus assuming that values are distributed around floating point zero. More specifically we define the quantized vector, $\mathbf{x}_q$, to be the product of the original vector, $\mathbf{x}$, and a quantization factor, θ, where $\theta = \frac{127}{|\max(\mathbf{x}_{\min}, \mathbf{x}_{\max})|}$. The lack of zero point offset avoids having to apply it prior to performing operations, such as multiplication, in lower precision thus speeding-up execution. Note that we force the quantization to be in the range $\pm(2^7 - 1)$. Thus, for the typical multiply-accumulate operation, the sum of the products of 2 multiplies is always strictly smaller than 15-bits, which allows us to carry more than one operation into a 32-bit accumulator, further speeding up inference. We leverage TensorFlow Lite optimization tools and runtime to execute the model on both ARM and x86 mobile architectures [35]. On ARM architectures, this achieves a 3× speedup compared to floating point execution.

4. CONTEXTUAL BIASING

Contextual biasing is the problem of injecting prior knowledge into an ASR system during inference, for example a user's favorite songs, contacts, apps or location [23]. Conventional ASR systems perform contextual biasing by building an n-gram finite state transducer (FST) from a list of biasing phrases, which is composed on-the-fly with the decoder graph during decoding [36]. This helps to bias the recognition result towards the n-grams contained in the contextual FST, and thus improves accuracy in certain scenarios. In the E2E RNN-T model, we use a technique similar to [36], to compute biasing scores $P_C(\mathbf{y})$, which are interpolated with the base model $P(\mathbf{y}|\mathbf{x})$ using shallow-fusion [37] during beam search:

$$\mathbf{y}^* = \arg\max_{\mathbf{y}} \log P(\mathbf{y}|\mathbf{x}) + \lambda \log P_C(\mathbf{y}) \quad (3)$$

where, λ is a tunable hyperparameter controlling how much the contextual LM influences the overall model score during beam search.

To construct the contextual LM, we assume that a set of word-level biasing phrases are known ahead of time, and compile them into a weighted finite state transducer (WFST) [38]. This word-level WFST, G, is then left-composed with a "speller" FST, S, which transduces a sequence of graphemes or word-pieces into the corresponding word, to obtain the contextual LM: $C = \min(\det(S \circ G))$. In order to avoid artificially boosting prefixes which match early on but do not match the entire phrase, we add a special failure arc which *removes* the boosted score, as illustrated in Figure 2. Finally, in order to improve RNN-T performance on proper nouns, which is critical for biasing, we train with an additional 500M unsupervised voice search utterances (each training batch is filled with supervised data 80% of the time and unsupervised data 20% of the time) [39]. The unsupervised data is transcribed by our production-level recognizer [40] and filtered to contain high-confidence utterances with proper nouns only. Note that training with this data does not change results on our voice-search and dictation test sets, but only improves performance on the contextual biasing results described in Table 2.

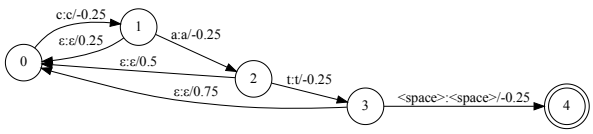

Fig. 2: Contextual FST for the word "cat", represented at the subword unit level with backoff arcs.

5. TEXT NORMALIZATION

Conventional models are trained in the *spoken* domain [41, 42], which allows them to convert unseen numeric sequences into the *written* domain during decoding (e.g., `navigate to two twenty one b baker street` → `navigate to 221b baker street`), which alleviates the data sparsity issue. This is done by training a class-based language model where classes such as ADDRESSNUM replace actual instances in the training data, and training grammar WFSTs that map these classes to all possible instances through hand-crafted rules. During decoding, the recognizer first outputs hypotheses in the spoken domain with the numeric part enclosed in the class tags (`<addressnum> two twenty one b </addressnum>`), which is then converted to written domain with a hand-crafted set of FST normalization rules.

For our purposes, it would be possible to train the E2E model to output hypotheses in the spoken domain, and then to use either a neural network [42] or an FST-based system [41] to convert the hypotheses into the written domain. To keep overall system size as small as possible, we instead train the E2E model to directly output hypotheses in the written domain (i.e., normalized into the output form). Since we do not observe a sufficiently large number of audio-text pairs containing numeric sequences in training, we generate a set of 5 million utterances containing numeric entities. We synthesize this data using a concatenative TTS approach with one voice [43] to create audio-text pairs, which we augment to our training data (each batch is filled with supervised data 90% of the time and synthetic data 10% of the time).

6. EXPERIMENTAL DETAILS

6.1. Data Sets

The training set used for experiments consists of 35 million English utterances ($\sim 27,500$ hours). The training utterances are anonymized and hand-transcribed, and are representative of Google's voice search and dictation traffic. This data set is created by artificially corrupting clean utterances using a room simulator, adding varying degrees of noise and reverberation such that the overall SNR is between 0dB and 30dB, with an average SNR of 12dB [44]. The noise sources are from YouTube and daily life noisy environmental recordings. The main test sets we report results on include 14.8K voice search (*VS*) utterances extracted from Google traffic, as well as 15.7K dictation utterances, which we refer to as the *IME* test set.

To evaluate the performance of contextual biasing, we report performance on 4 voice command test sets, namely *Songs* (requests to play media), *Contacts-Real*, *Contacts-TTS* (requests to call/text contacts), and *Apps* (requests to interact with an app). All sets except *Contacts-Real* are created by mining song, contact or app names from the web, and synthesizing TTS utterances in each of these categories. The *Contacts-Real* set contains anonymized and hand-transcribed utterances extracted from Google traffic. Only utterances with an intent to communicate with a contact are included in the test set. Noise is then artificially added to the TTS data, similar to the process described above [44].

To evaluate the performance of numerics, we report results on a real data numerics set (*Num-Real*), which contains anonymized and hand-transcribed utterances extracted from Google traffic. In addition, we include performance on a synthesized numerics set (*Num-TTS*), which uses Parallel Wavenet [30] with 1 voice. No utterance / transcript from the numerics test set appears in the TTS training set from Section 5.

6.2. Model Architecture Details

All experiments use 80-dimensional log-Mel features, computed with a 25ms window and shifted every 10ms. Similar to [40], at the current frame, t, these features are stacked with 3 frames to the left

and downsampled to a 30ms frame rate. The encoder network consists of 8 LSTM layers, where each layer has 2,048 hidden units followed by a 640-dimensional projection layer. For all models in this work, we insert a time-reduction layer with the reduction factor $N = 2$ after the second layer of encoder to achieve $1.7\times$ improvement in overall system speed without any accuracy loss. The prediction network is 2 LSTM layers with 2,048 hidden units and a 640-dimensional projection per layer. The encoder and prediction network are fed to a joint-network that has 640 hidden units. The joint network is fed to a softmax layer, with either 76 units (for graphemes) or 4,096 units (for word-pieces [45]). The total size of the RNN-T model is 117M parameters for graphemes and 120M parameters for word-pieces. For the WPM, after quantization, the total size is 120MB. All RNN-T models are trained in Tensorflow [46] on 8×8 Tensor Processing Units (TPU) slices with a global batch size of 4,096.

In this work, we compare the RNN-T model to a strong baseline conventional CTC embedded model, which is similar to [9] but much larger. The acoustic model consists of a CI-phone CTC model with 6 LSTM layers, where each layer has 1,200 hidden units followed by a 400-dimensional projection layer, and a 42-phoneme output softmax layer. The lexicon has 500K words in the vocabulary. We use a 5-gram first-pass language model and a small and efficient second-pass rescoring LSTM LM. Overall the size of the model after quantization is 130MB, which is of similar size to the RNN-T model.

7. RESULTS

7.1. Quality Improvements

Table 1 outlines various improvements to the quality of RNN-T models. First, $E1$ shows that layer norm [24] helps to stabilize training, resulting in a 6% relative improvement in WER for VS and IME. Next, by moving RNN-T training to TPUs [25] and having a larger batch size, we can get between a 1–4% relative improvement. Finally, changing units from graphemes to word-pieces [21] ($E3$) shows a 9% relative improvement. Overall, our algorithmic changes show 27% and 25% relative improvement on VS and IME respectively compared to the baseline conventional CTC embedded model ($B0$). All experiments going forward in the paper will report results using layer norm, word-pieces and TPU training ($E3$).

ID	Model	*VS WER*	*IME WER*
E0	RNN-T Grapheme	8.1%	4.9%
E1	+Layer Norm	7.6%	4.6%
E2	+Larger Batch	7.5%	4.4%
E3	+Word-piece	**6.8%**	**4.0%**
B0	CTC	9.3%	5.3%

Table 1: RNN-T model improvements. All models are *unquantized*.

7.2. Contextual Biasing

Table 2 shows results using the shallow-fusion biasing mechanism. We report biasing results with just supervised data ($E4$) and also including unsupervised data ($E6$). We also show biasing performance for the CTC conventional model in $B1$. The table indicates that E2E biasing outperforms or is on par with conventional-model biasing on all sets, except songs likely because the out-of-vocabulary rate in songs is 1.0%, which is higher than contacts (0.2%) or apps (0.5%).

7.3. Text normalization

Next, Table 3 indicates the performance of the baseline RNN-T ($E3$) word-piece model on two numeric sets. As can be seen in the table, the WER on the *Num-TTS* set is really high. A closer error analysis

ID	Model	*Songs*	*Contacts-Real*	*Contacts-TTS*	*Apps*
E3	RNN-T Word-piece	20.0%	15.9%	35.0%	13.1%
E4	+ Biasing	3.4%	6.4%	7.1%	1.9%
E5	E3 + Unsupervised	14.7%	15.4%	25.0%	9.6%
E6	+ Biasing	3.0%	**5.8%**	**5.4%**	**1.7%**
B1	CTC + Biasing	**2.4%**	6.8%	5.7%	2.4%

Table 2: WER on contextual biasing sets. All models *unquantized*.

reveals that these are due to the text normalization errors: e.g., if the user speaks `call two double three four ...`, the RNN-T model hypothesizes `2 double 3 4` rather than `2334`. To fix this, we train the RNN-T model with more numeric examples ($E7$), as described in Section 5, which mitigates this issue substantially, at the cost of a small degradation on *VS* and *IME*. However, we note that this still outperforms the baseline system with a separate FST-based normalizer [9] ($B0$) on all sets.

ID	Model	*VS*	*IME*	*Num-Real*	*Num-TTS*
E3	RNN-T Word-piece	6.8%	4.0%	6.7%	22.8%
E7	+ numerics TTS	**7.0%**	**4.1%**	**6.9%**	**4.3%**
B0	CTC	9.3%	5.3%	8.4%	6.8%

Table 3: WER on numeric sets. All models are *unquantized*.

7.4. Real Time Factor

In Table 4, we report WER and RT90, i.e. real time factor (processing time divided by audio duration) at 90 percentile, where lower values indicate faster processing and lower user-perceived latency. Comparing $E2$ and $E7$, we can see that the RNN-T word-piece model outperforms the grapheme model in both accuracy and speed.

Quantization speeds up inference further: asymmetric quantization ($E8$) improves RT90 by 28% compared to the float model ($E7$) with only a 0.1% absolute WER degradation; symmetric quantization ($E9$), which assumes that weights are centered around zero, only introduces additional small degradation on VS WER, but leads to a substantial reduction in RT90 (64% compared to the float model), which is twice as fast as real time. Moreover, quantization reduces model size by $4\times$. Our best model ($E9$) is also faster than the conventional CTC model $B2$, while still achieving accuracy improvements of more than 20%.

ID	Model	*VS*	*IME*	*RT90*
E2	RNN-T Grapheme (Float)	7.5%	4.4%	1.58
E7	RNN-T Word-piece (Float)	7.0%	4.1%	1.43
E8	+ Asymmetric Quantization	7.1%	4.2%	1.03
E9	+ Symmetric Quantization	**7.3%**	**4.2%**	**0.51**
B2	CTC + Symmetric Quantization	9.2%	5.4%	0.86

Table 4: Quantization results on WER and RT90.

8. CONCLUSIONS

We present the design of a compact E2E speech recognizer based on the RNN-T model which runs twice as fast as real-time on a Google Pixel phone, and improves WER by more than 20% over a strong embedded baseline system on both voice search and dictation tasks. This is achieved through a series of modifications to the RNN-T model architecture, quantized inference, and the use of TTS to synthesize training data for the E2E model. The proposed system shows that an end-to-end trained, all-neural model is very well suited for on-device applications for its ability to perform streaming, high-accuracy, low-latency, contextual speech recognition.

9. REFERENCES

[1] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, and B. Kingsbury, "Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 82–97, Nov 2012.

[2] J. Cohen, "Embedded speech recognition applications in mobile phones: Status, trends, and challenges," in *Proc. ICASSP*, 2008, pp. 5352–5355.

[3] J. Schalkwyk, D. Beeferman, F. Beaufays, B. Byrne, C. Chelba, M. Cohen, M. Kamvar, and B. Strope, *"Your Word is my Command": Google Search by Voice: A Case Study*, pp. 61–90, Springer US, 2010.

[4] T. N. Sainath and C. Parada, "Convolutional neural networks for small-footprint keyword spotting," in *Proc. Interspeech*, 2015.

[5] S. O. Arik, M. Kliegl, R. Child, J. Hestness, A. Gibiansky, C. Fougner, R. Prenger, and A. Coates, "Convolutional recurrent neural networks for small-footprint keyword spotting," in *Proc. Interspeech*, 2017.

[6] Y. He, R. Prabhavalkar, K. Rao, W. Li, A. Bakhtin, and I. McGraw, "Streaming small-footprint keyword spotting using sequence-to-sequence models," in *Proc. ASRU*, Dec 2017, pp. 474–481.

[7] R. Alvarez and H.J. Park, "End-to-End streaming keyword spotting," in *Proc. ICASSP*, 2019.

[8] A. Waibel et al., "Speechalator: Two-way speech-to-speech translation on a consumer PDA," in *Proc. Eurospeech*, 2003.

[9] I. McGraw, R. Prabhavalkar, R. Alvarez, M. G. Arenas, K. Rao, D. Rybach, O. Alsharif, H. Sak, A. Gruenstein, F. Beaufays, and C. Parada, "Personalized speech recognition on mobile devices," in *Proc. ICASSP*, 2016, pp. 5955–5959.

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

[11] D. Bahdanau, J. Chorowski, D. Serdyuk, P. Brakel, and Y. Bengio, "End-to-End Attention-based Large Vocabulary Speech Recognition," in *Proc. ICASSP*, 2016.

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

[13] 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.

[14] C. 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.

[15] A. Graves, S. Fernandez, F. Gomez, and J. Schmidhuber, "Connectionist Temporal Classification: Labeling Unsegmented Sequenece Data with Recurrent Neural Networks," in *Proc. ICML*, 2006.

[16] A. Graves and N. Jaitly, "Towards end-to-end speech recognition with recurrent neural networks," in *Proc. ICML*, 2014, pp. 1764–1772.

[17] A. Hannun, C. Case, J. Casper, B. Catanzaro, G. Diamos, E. Elsen, R. Prenger, S. Satheesh, S. Sengupta, A. Coates, and A. Y. Ng, "Deep speech: Scaling up end-to-end speech recognition," .

[18] Y. Miao, M. Gowayyed, and F. Metze, "Eesen: End-to-end speech recognition using deep rnn models and wfst-based decoding," in *Proc. ASRU*, 2015, pp. 167–174.

[19] H. Soltau, H. Liao, and H. Sak, "Neural speech recognizer: Acoustic-to-word lstm model for large vocabulary speech recognition," in *Proc. Interspeech*, 2017, pp. 3707–3711.

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

[21] 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.

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

[23] P. Aleksic, M. Ghodsi, A. Michaely, C. Allauzen, K. Hall, B. Roark, D. Rybach, and P. Moreno, "Bringing Contextual Information to Google Speech Recognition," in *in Proc. Interspeech*, 2015.

[24] J. L. Ba, R. Kiros, and G. E. Hinton, "Layer Normalization," *CoRR*, vol. abs/1607.06450, 2016.

[25] K. Sim, A. Narayanan, T. Bagby, T.N. Sainath, and M. Bacchiani, "Improving the Efficiency of Forward-Backward Algorithm using Batched Computation in TensorFlow," in *ASRU*, 2017.

[26] M. Schuster and K. Nakajima, "Japanese and korean voice search," in *Proc. ICASSP*, 2012, pp. 5149–5152.

[27] R. Alvarez, R. Prabhavalkar, and A. Bakhtin, "On the efficient representation and execution of deep acoustic models," in *Proc. Interspeech*, 2016.

[28] I. Williams, A. Kannan, P. Aleksic, D. Rybach, and T. N. Sainath, "Contextual speech recognition in end-to-end neural network systems using beam search," in *Proc. Interspeech*, 2018.

[29] G. Pundak, T. Sainath, R. Prabhavalkar, A. Kannan, and D. Zhao, "Deep Context: End-to-End Contextual Speech Recognition," in *Proc. SLT*, 2018.

[30] A. van den Oord, Y. Li, and I. Babuschkin et. al., "Parallel wavenet: Fast high-fidelity speech synthesis," Tech. Rep., Google Deepmind, 2017.

[31] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, Nov 1997.

[32] H. Sak, A. Senior, and F. Beaufays, "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling," in *Proc. Interspeech*, 2014.

[33] H. Soltau, H. Liao, and H. Sak, "Reducing the Computational Complexity for Whole Word Models," in *ASRU*, 2017.

[34] N. P. Jouppi et al., "In-datacenter performance analysis of a tensor processing unit," in *Proc. International Symposium on Computer Architecture (ISCA)*, 2017.

[35] R. Alvarez, R. Krishnamoorthi, S. Sivakumar, Y. Li, A. Chiao, P Warden, S. Shekhar, S. Sirajuddin, and Davis. T., "Introducing the Model Optimization Toolkit for TensorFlow," https://medium.com/tensorflow/introducing-the-model-optimization-toolkit-for-tensorflow-254aca1ba0a3, Accessed: 2018-10-22.

[36] K.B. Hall, E. Cho, C. Allauzen, F. Beaufays, N. Coccaro, K. Nakajima, M. Riley, B. Roark, D. Rybach, and L. Zhang, "Composition-based on-the-fly rescoring for salient n-gram biasing," in *Interspeech*, 2015.

[37] A. Kannan, Y. Wu, P. Nguyen, T. N. Sainath, Z. Chen, and R. Prabhavalkar, "An analysis of incorporating an external language model into a sequence-to-sequence model," in *Proc. ICASSP*, 2018.

[38] Mehryar Mohri, Fernando Pereira, and Michael Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1, pp. 69–88, 2002.

[39] B. Li, T.N. Sainath, R. Pang, and Z. Wu, "Semi-supervised Training for End-to-End models via Weak Distillation," in *Proc. ICASSP*, 2019.

[40] G. Pundak and T. N. Sainath, "Lower Frame Rate Neural Network Acoustic Models," in *Proc. Interspeech*, 2016.

[41] L. Vasserman, V. Schogol, and K.B. Hall, "Sequence-based class tagging for robust transcription in asr," in *Proc. Interspeech*, 2015.

[42] R. Sproat and N. Jaitly, "An RNN Model of Text Normalization," in *Proc. Interspeech*, 2017.

[43] X. Gonzalvo, S. Tazari, C. Chan, M. Becker, A. Gutkin, and H. Silen, "Recent Advances in Google Real-time HMM-driven Unit Selection Synthesizer," in *Interspeech*, 2016.

[44] C. Kim, A. Misra, K. Chin, T. Hughes, A. Narayanan, T. N. Sainath, and M. Bacchiani, "Generated of large-scale simulated utterances in virtual rooms to train deep-neural networks for far-field speech recognition in google home," in *Proc. Interspeech*, 2017.

[45] Mike Schuster and Kaisuke Nakajima, "Japanese and Korean voice search," *Proc. ICASSP*, 2012.

[46] M. Abadi et al., "TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems," Available online: http://download.tensorflow.org/paper/whitepaper2015.pdf, 2015.