INTERSPEECH 2016
September 8–12, 2016, San Francisco, USA

Memory-efficient modeling and search techniques for hardware ASR decoders

Michael Price[1,2], *Anantha Chandrakasan*[2], *James Glass*[1]

[1]Computer Science and Artificial Intelligence Laboratory
[2]Microsystems Technology Laboratory
MIT, Cambridge, MA, USA

pricem@mit.edu, anantha@mtl.mit.edu, glass@mit.edu

Abstract

This paper gives an overview of acoustic modeling and search techniques for low-power embedded ASR decoders. Our design decisions prioritize memory bandwidth, which is the main driver in system power consumption. We evaluate three acoustic modeling approaches–Gaussian mixture model (GMM), subspace GMM (SGMM) and deep neural network (DNN)–and identify tradeoffs between memory bandwidth and recognition accuracy. We also present an HMM search scheme with WFST compression and caching, predictive beam width control, and a word lattice. Our results apply to embedded system implementations using microcontrollers, DSPs, FPGAs, or ASICs.

Index Terms: speech recognition, neural networks, fixed-point arithmetic, embedded systems

1. Introduction

Bringing ASR capabilities to small or portable devices requires attention to computational efficiency. Using a more efficient processor, whether a low-voltage DSP or a specialized ASIC, significantly reduces the energy used by arithmetic operations on the chip. There is some memory (i.e. caches) integrated on these chips, but it is not enough to store large ASR models–typically 100+ MB for an acoustic model and search graph. However, accessing an external memory such as DRAM or flash remains comparatively expensive [1]. Any attempt to reduce system power must reduce the amount of data exchanged with this external memory.

This paper explains the most helpful techniques we identified to perform speech recognition in these constrained environments. Acoustic model comparisons (Section 2) show that DNNs provide good accuracy even when the number and precision of parameters are limited. Search techniques that accommodate memory limitations are presented in Section 3.

2. Acoustic model

We are operating within the conventional HMM framework for ASR, with a WFST decoding graph [2]. The acoustic model has to evaluate the likelihood of input features $\mathbf{y}_t$ with respect to a set of distributions $p(\mathbf{y}|i)$, where i is the index of an acoustic state or senone. The accuracy of likelihoods directly impacts the search workload and word accuracy of a recognizer.

The general trend in ASR research has been to leverage growing compute resources and datasets to train increasingly elaborate models. Implementers of low-power ASR systems cannot blindly follow this trend. Instead of exclusively pursuing accuracy, we ask a slightly different question.

2.1. What is the most accurate acoustic model with 1 MB of parameters?

Until 2010, most ASR systems used Gaussian mixture models (GMMs) to specify each senone distribution. The community has since shifted to deep neural networks (DNNs), including convolutional and recurrent networks [3]. Povey et al. also developed a subspace GMM (SGMM) framework that improves accuracy over GMMs [4]. It was unclear whether the advantages of DNNs would be maintained in memory-limited implementations, so we characterized the tradeoff between WER and memory bandwidth for GMM, SGMM, and DNN models.

We used Kaldi [5] to train recognizers for several ASR tasks. To model the accuracy and bandwidth of our hardware-oriented algorithm changes, we constructed a separate ASR decoder in C++ and performed comparisons with a speaker-independent recognizer on the WSJ [6] dev93 task. The recognizer's pruned trigram LM (`bd_tgpr` in the Kaldi recipe) has a vocabulary of 145k words and 1.47M N-grams; the resulting WFSTs have 9.2–9.8M states and 22.2–23.6M arcs, depending on the number of acoustic states.

2.1.1. Gaussian mixture model (GMM)

The GMM models each distribution as a weighted sum of multivariate normal distributions over $\mathbf{y}$. With single-precision floating point values, a typical GMM acoustic model occupies 50 MB; naive evaluation at 100 fps would require 5 GB/s of memory bandwidth. This can be reduced by compressing the GMM, or by not evaluating the entire model on every frame.

One way to shrink a GMM is to use fixed-point mean and variance coefficients with reduced precision. With a nonlinear quantizer, acceptable accuracy can be obtained with as few as 5 bits for the mean and 3 bits for the variance [7].[1] Some hardware architectures can also evaluate the distributions for multiple frames in parallel [8], in exchange for longer latency: the first frame is not evaluated until features for a batch of frames have been loaded. The local memory required by a GMM is then dominated by the storage of likelihood results. Assuming a fixed memory size (in this case 1 Mb) and precision (32 bits), there is a tradeoff between the number of distributions and the number of frames that can be evaluated in parallel. Taking this into account, the performance of several hyperparameter combinations is shown in Figure 1. There is a steep tradeoff between memory bandwidth and recognition accuracy; this is generally true for all three modeling frameworks.

[1]Unless the feature space has been whitened, separate quantizers must be estimated for each feature dimension.

http://dx.doi.org/10.21437/Interspeech.2016-287

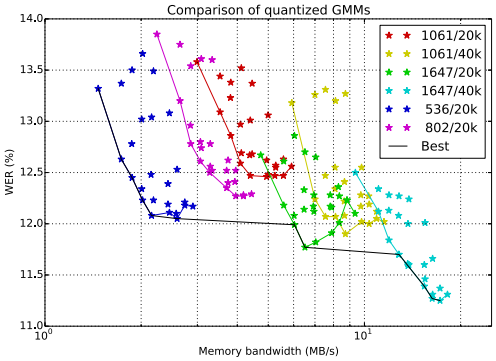

Figure 1: Bandwidth/accuracy tradeoff for GMM acoustic models of different model size and quantization depth.

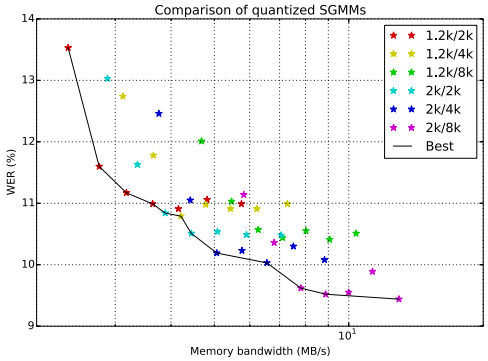

Figure 2: Bandwidth/accuracy tradeoff for SGMM.

2.1.2. *Subspace Gaussian mixture model (SGMM)*

The SGMM is a collection of GMMs whose parameters vary in a low-rank vector space relative to a universal background model, or UBM [4]. Each distribution is modeled as follows:

$$P(\mathbf{y}_t|i) = \sum_{m=0}^{M_i-1} k_{im} \sum_{c=0}^{C-1} w_{cim}\mathcal{N}(\mathbf{y};\mu_{cim},\mathbf{\Sigma}_c)$$

$$\mu_{cim} = \mathbf{M}_c\mathbf{v}_{im}$$

$$w_{cim} = \frac{\exp \mathbf{w}_c^T\mathbf{v}_{im}}{\sum_{c'=0}^{C-1}\exp \mathbf{w}_{c'}^T\mathbf{v}_{im}}$$

where the vectors $\mathbf{v}_{im}$ describe the location of parameters for distribution i in a low-rank subspace, and $\mathbf{M}_c$ and $\mathbf{w}_c$ define the subspace for UBM component c. The second level of mixing, performed by the weights k_{im}, accounts for different ways that a single phonetic unit can be realized (i.e. depending on context). We exploit two opportunities for bandwidth reduction:

1. It is possible to store global parameters (i.e. UBM) locally and fetch a subset of the state-specific parameters for each evaluation.
2. "Gaussian selection" can be performed to prune the set of mixture components for each frame.

The quantization and parallelization techniques that we used for GMMs also apply to SGMMs. For a fair comparison, the 1 Mb local memory is split between likelihoods and intermediate results, reducing the number of frames that can be evaluated in parallel. Figure 2 shows the bandwidth/accuracy tradeoff for different model dimensions. All models used a 400-component UBM and selected the top 3 components per frame.

2.1.3. *Neural network (NN)*

A neural network is a sequence of layers that operate on real-valued vectors. Each layer l performs an affine transformation of its input followed by an element-wise nonlinearity:

$$\mathbf{x}_{l+1} = g(\mathbf{W}_l\mathbf{x}_l + \mathbf{b}_l)$$

We use a feed-forward, fully-connected NN that operates on stacked MFCC feature vectors. Most of the time and bandwidth is spent multiplying the layers' weight matrices by their input vectors. To save bandwidth, the weights can be quantized and multiple frames can be evaluated in parallel, as with GMMs. We also train sparse matrices and store only the nonzero weights [9]; similar gains could be made with low-rank approximations [10]. Our model performs a piecewise Chebyshev polynomial approximation to the sigmoid function.

NN evaluation requires working memory proportional to the width of the largest layer. This memory is used most efficiently if all of the layers, including the output layer, have approximately the same number of outputs. This is in contrast to most software implementations where the output layer is large (2k–16k nodes) relative to the hidden layers (512–2k nodes).

Figure 3 shows results for a variety of NN sizes and sparsities. For this task, small NNs perform almost as well as large NNs. A quantized sparse NN with 512 nodes per layer requires about 4 MB/s of memory bandwidth to evaluate, with 10% relative WER degradation compared to the best model tested.

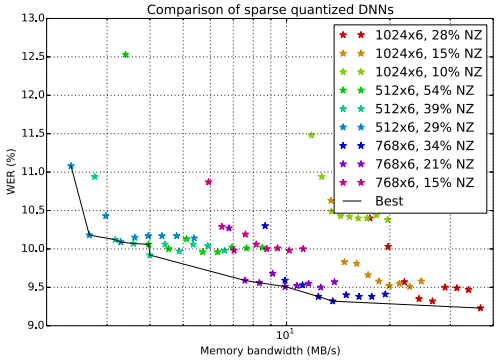

Figure 3: Bandwidth/accuracy tradeoff for NN acoustic model.

2.2. Model comparison and discussion

Figure 4 shows the Pareto optimal subset of results from each model framework. This allows us to see the tradeoff between WER and memory bandwidth, assuming that hyperparameters such as quantizer depth and model dimensions have been optimized. Only the acoustic model's contribution to bandwidth is included.

We selected the DNN framework for our implementation because it offers the best accuracy, even when memory bandwidth is strictly limited. We hope that further improvements developed by the ASR research community can be ported to hardware with incremental changes in architecture.

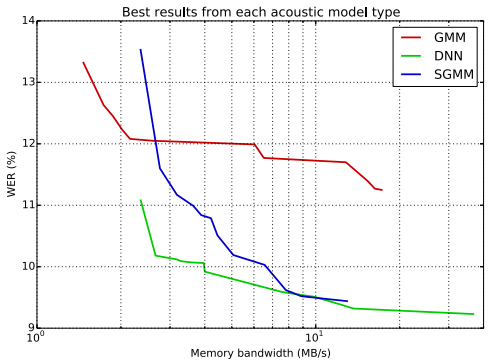

Figure 4: Bandwidth/accuracy comparison of AM frameworks.

3. Search

Our recognizers use Kaldi's $HCLG$ WFST construction. The search subsystem initializes a set of hypotheses for the HMM's hidden state and propagates them forward in time. Our starting point is the architecture presented in [11], which moves hypotheses between two "active state lists" in local memory.

The forward pass of Viterbi search is divided into two phases (based on the input label of WFST arcs): the "non-ϵ" phase and the "ϵ" phase. To avoid recursion in the ϵ phase, we preprocess the WFST so that all paths of only ϵ arcs are bypassed by a single ϵ arc having the same total weight [12]. Some paths may traverse multiple arcs with non-ϵ output labels, so we create multi-word output symbols as necessary to bypass these paths; see Figure 5.

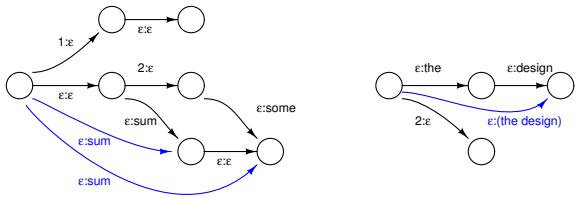

Figure 5: WFST preprocessing with multi-word output symbols. Left: Adding ϵ arcs to bypass arbitrary length chains of only ϵ arcs [12]. Right: Compound word labels are created for arcs that bypass multiple non-ϵ output labels.

The following sections describe our efforts to implement a WFST search supporting large models on a processor with limited local memory and off-chip memory bandwidth.

3.1. WFST

Search initialization, forward passes, final weight updates, and backtraces all require access to a WFST. Arcs expanded in the forward pass account for significant memory bandwidth (10–100 MB/s) in a typical decoder. We apply both compression and caching to reduce WFST memory bandwidth. These work hand in hand: compression makes the cache look larger, relative to the information it is storing. Figure 6 shows the cache hit rate and WFST-related memory bandwidth for decoding a typical utterance with different beam widths.

Our WFST encoding is based on [7]. It preserves graph structure but quantizes weights (arc weights and state final

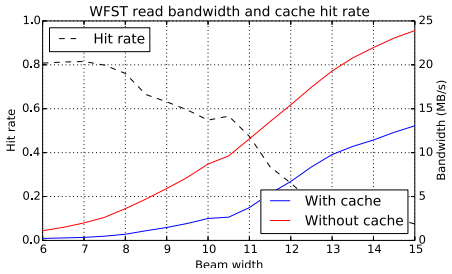

Figure 6: WFST cache performance with varying beam width.

weights) because high accuracy is unnecessary. Each state is stored as a state header followed by a sequence of arcs. The data structure is illustrated in Figure 7.

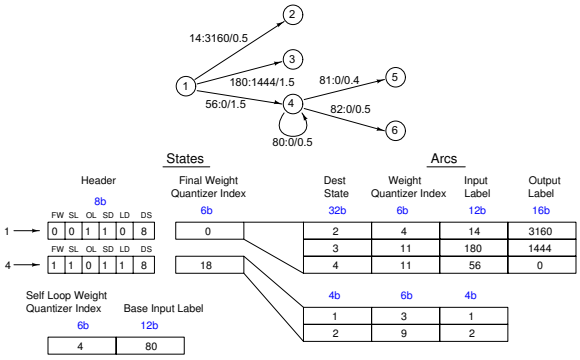

Figure 7: Example WFST and compressed format.

States are packed byte-aligned so that the byte address of a state can be used as a unique ID. This compression scheme results in compression ratios of 0.279–0.366 for the five recognizers we studied, which is similar to `gzip` (0.308–0.378).

3.2. Beam width control

ASR systems keep runtime in check by using a "beam search" variant of the Viterbi algorithm, pruning unlikely hypotheses during each forward pass [13]. The pruning can be relative (based on likelihood) or absolute (based on a maximum number of hypotheses). The desired number of hypotheses varies naturally over time; for example, there is a high branching factor near the beginning of a word. Figure 8 demonstrates this effect when decoding with relative pruning.

Hardware implementations need a combination of relative and absolute pruning to operate efficiently with a fixed local memory. We have a configurable hard limit for the number of active states; when this limit is reached during search, we reduce the beam width and prune the hypotheses in-place. This introduces search errors: it changes the beam width within a single frame, and the beam width is reduced in fixed (relatively large) increments to avoid repeating the process many times.

Performance can be improved over this crude approach by adding a "soft" cutoff inspired by Kaldi [14]. When the soft cutoff is exceeded, we preemptively reduce the beam width. This is done by storing a histogram of relative log-likelihoods, as in [15], and estimating (via linear interpolation of the cumulative sum) the beam width needed to hit a target state count. This is illustrated in Figure 9. We use the histogram to compute a

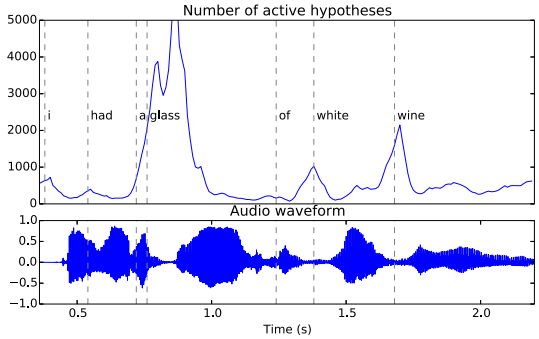

Figure 8: The changing level of ambiguity in the speech signal causes search workload to vary over time.

pruning threshold for the next frame, rather than retroactively pruning the current hypotheses. This feature is only engaged if the soft cutoff is lower than the hard cutoff.

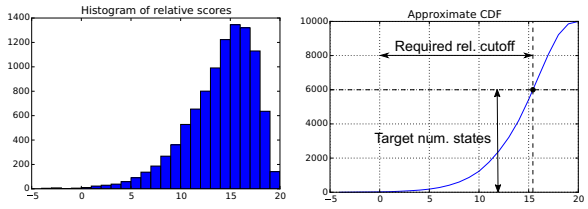

Figure 9: CDF-based beam width estimation using a soft cutoff.

The performance impact of these pruning mechanisms is visible in Figure 10. These results were obtained on the WSJ dev93 task (configured as in section 2) using a sparse 6-layer DNN acoustic model with 1024 nodes per hidden layer. Generally, lowering the cutoffs speeds up decoding at higher beam widths. Using a soft cutoff at 2k states (compared to the default of 5k) saves almost as much time as a hard cutoff, but doesn't degrade accuracy.

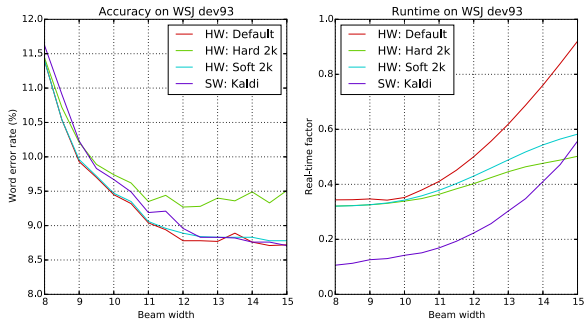

Figure 10: WER (left) and RTF (right) for search using varying soft and hard cutoffs, compared to Kaldi.

3.3. Word lattice

Viterbi search generates the equivalent of a state lattice, but the information needed to reconstruct word hypotheses can be stored in a smaller word lattice. As Figure 11 shows, most of the arcs in the state lattice have ϵ output labels. To reduce memory bandwidth (both reads and writes), we maintain a word lattice in local memory and save it to the external memory when there is an overflow. The lattice is pruned after every frame.

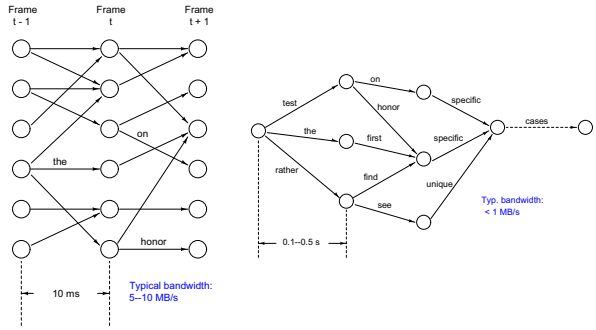

Figure 11: State lattice (left) and word lattice (right) structures.

The WFST state ID is not sufficient to uniquely identify a word lattice (WL) state because WFST states can be visited multiple times (for example, when a phrase matching a language model N-gram appears twice in an utterance). Loops can be prevented by using a tuple of (WFST state ID, number of words) as the WL state ID, but we use (WFST state ID, frame index) to keep scores consistent for pruning. This is similar to the word trace mentioned in [16].

The word lattice must be coordinated with the state list. We opted to store the WL state ID associated with every hypothesis in the state list. This requires more local memory, but the state list only includes hypotheses for 2 frames, whereas the word lattice covers the entire utterance.

Figure 12 shows the effect of the word lattice structure on external memory writes. Heavy workloads force more frequent word lattice snapshots, but there is still a 7–8x bandwidth savings because not all arcs have word labels.

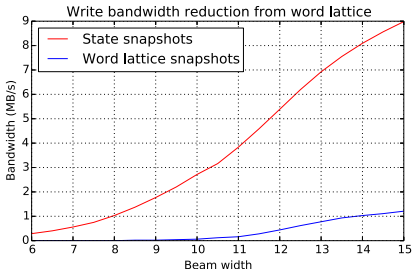

Figure 12: Write bandwidth comparison of state lattice and word lattice approaches, averaged over a single utterance.

4. Conclusions

This paper has laid the groundwork for memory-efficient ASR decoders with a bandwidth requirement on the order of 10 MB/s. We will provide more details about our hardware architecture and implementation in a future publication. Future work will also consider recent developments such as CTC-trained LSTM models [17] and split-VQ compression [18].

5. Acknowledgements

This work was funded by Quanta Computer via the Qmulus Project.

6. References

[1] M. Horowitz, "Computing's energy problem (and what we can do about it)," in *Solid-State Circuits Conference Digest of Technical Papers (ISSCC), 2014 IEEE International*, Feb 2014, pp. 10–14.

[2] M. Mohri, F. Pereira, and M. Riley, "Speech recognition with weighted finite-state transducers," in *Springer Handbook on Speech Processing and Speech Communication*, 2008.

[3] T. Sainath, O. Vinyals, A. Senior, and H. Sak, "Convolutional, Long Short-Term Memory, fully connected Deep Neural Networks," in *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, April 2015, pp. 4580–4584.

[4] D. Povey, L. Burget, M. Agarwal, P. Akyazi, K. Feng, A. Ghoshal, O. Glembek, N. Goel, M. Karafiat, A. Rastrow, R. Rose, P. Schwarz, and S. Thomas, "Subspace Gaussian Mixture Models for speech recognition," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, March 2010, pp. 4330–4333.

[5] D. Povey, A. Ghoshal, G. Boulianne, L. Burget, O. Glembek, N. Goel, M. Hannemann, P. Motlicek, Y. Qian, P. Schwarz, J. Silovsky, G. Stemmer, and K. Vesely, "The Kaldi Speech Recognition Toolkit," in *IEEE 2011 Workshop on Automatic Speech Recognition and Understanding*. IEEE Signal Processing Society, Dec. 2011, iEEE Catalog No.: CFP11SRW-USB.

[6] D. B. Paul and J. M. Baker, "The design for the Wall Street Journal-based CSR corpus," in *Proceedings of the workshop on Speech and Natural Language*, ser. HLT '91. Stroudsburg, PA, USA: Association for Computational Linguistics, 1992, pp. 357–362. [Online]. Available: http://dx.doi.org/10.3115/1075527.1075614

[7] I. L. Hetherington, "PocketSUMMIT: Small-Footprint Continuous Speech Recognition," in *INTERSPEECH-2007*, 2007, pp. 1465–1468.

[8] G. He, T. Sugahara, Y. Miyamoto, T. Fujinaga, H. Noguchi, S. Izumi, H. Kawaguchi, and M. Yoshimoto, "A 40 nm 144 mW VLSI Processor for Real-Time 60-kWord Continuous Speech Recognition," *Circuits and Systems I: Regular Papers, IEEE Transactions on*, vol. 59, no. 8, pp. 1656–1666, 2012.

[9] D. Yu, F. Seide, G. Li, and L. Deng, "Exploiting sparseness in deep neural networks for large vocabulary speech recognition," in *Acoustics, Speech and Signal Processing (ICASSP), 2012 IEEE International Conference on*, March 2012, pp. 4409–4412.

[10] P. Nakkiran, R. Alvarez, R. Prabhavalkar, and C. Parada, "Compressing deep neural networks using a rank-constrained topology," in *Proceedings of Annual Conference of the International Speech Communication Association (Interspeech)*, 2015, pp. 1473–1477.

[11] M. Price, J. Glass, and A. Chandrakasan, "A 6 mW, 5,000-Word Real-Time Speech Recognizer Using WFST Models," *Solid-State Circuits, IEEE Journal of*, vol. 50, no. 1, pp. 102–112, Jan 2015.

[12] J. Choi, K. You, and W. Sung, "An FPGA implementation of speech recognition with weighted finite state transducers," in *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, march 2010, pp. 1602 –1605.

[13] F. Jelinek, *Statistical methods for speech recognition*. MIT Press, 1997.

[14] D. Povey et al., "Decoders used in the Kaldi toolkit (Kaldi documentation)," available online: http://kaldi-asr.org/doc/decoders.html.

[15] V. Steinbiss, B.-H. Tran, and H. Ney, "Improvements in beam search." in *ICSLP*, vol. 94, no. 4, 1994, pp. 2143–2146.

[16] G. Saon, D. Povey, and G. Zweig, "Anatomy of an extremely fast LVCSR decoder." in *INTERSPEECH*, 2005, pp. 549–552.

[17] I. McGraw, R. Prabhavalkar, R. Alvarez, M. G. Arenas, K. Rao, D. Rybach, O. Alsharif, A. Gruenstein, C. Parada *et al.*, "Personalized speech recognition on mobile devices," in *2016 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2016, pp. 5955–5959.

[18] Y. Wang, J. Li, and Y. Gong, "Small-footprint high-performance deep neural network-based speech recognition using split-vq," in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2015, pp. 4984–4988.