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

Lookup-Table Recurrent Language Models for Long Tail Speech Recognition

W. Ronny Huang, Tara N. Sainath, Cal Peyser, Shankar Kumar, David Rybach, Trevor Strohman

Google Research, USA

{wrh, tsainath, cpeyser, shankarkumar, rybach, strohman}@google.com

Abstract

We introduce Lookup-Table Language Models (LookupLM), a method for scaling up the size of RNN language models with only a constant increase in the floating point operations, by increasing the expressivity of the embedding table. In particular, we instantiate an (additional) embedding table which embeds the previous n-gram token sequence, rather than a single token. This allows the embedding table to be scaled up arbitrarily—with a commensurate increase in performance—without changing the token vocabulary. Since embeddings are sparsely retrieved from the table via a lookup; increasing the size of the table adds neither extra operations to each forward pass nor extra parameters that need to be stored on limited GPU/TPU memory. We explore scaling n-gram embedding tables up to nearly a billion parameters. When trained on a 3-billion sentence corpus, we find that LookupLM improves long tail log perplexity by 2.44 and long tail WER by 23.4% on a downstream speech recognition task over a standard RNN language model baseline, an improvement comparable to a scaling up the baseline by 6.2x the number of floating point operations.

Index Terms: language modeling, speech recognition, embedding, n-gram, long tail

1. Introduction

Improving the decoding of rare words or sequences is an ongoing problem in natural language tasks. For example, speech recognition systems misrecognize many legitimate token sequences that appear with zero or low frequency in transcribed acoustic data. This is particularly a problem with end-to-end (E2E) models [1, 2], which are typically trained on a small-fraction of audio-text pairs compared to the amount of text-only data. In addition, E2E models tend to have narrower beam searches which also contributes to poor rare-word performance [3]. One solution to this problem is to integrate language models trained on text-only data which often contain rich long tail content [4]. For example, a language model trained on a corpus containing rare words absent in speech data can bias a speech recognition system toward correctly decoding those rare words via language model fusion [5, 6, 7, 8]. Note we use "rare word" and "long tail" interchangeably throughout this paper to mean rare sub-sequences which may or may not occupy an entire example.

Neural language models have shown in recent years that size matters. Given sufficient compute and data, our ability to model the vast complexity of language scales with the size of the model [9], with perplexities showing no sign of saturating even up to a trillion parameters [10, 11, 12, 13]. Unfortunately, this need for scaling up comes at the cost of computational parsimony, often a key factor in commercial systems.

One relatively unexplored path toward scaling up language models is via increasing the embedding vocabulary. Typically the *embedding* vocabulary (number of rows in the embedding table) is determined by the *token* vocabulary; i.e. there is one embedding for each unique token. Increasing the embedding vocabulary necessitates increasing the token vocabulary and, undesirably, the softmax parameters. Further, often a specific token vocabulary (e.g. a particular 4096-wordpiece vocabulary) is required in order to be compatible with downstream tasks (e.g. for shallow fusion with a speech model which has that vocabulary).

Yet, scaling up the embedding vocabulary has a key advantage: embedding lookups are sparse and lightweight. For instance in language models, only a single fixed-dimensional vector from the embedding table corresponding to the current token must be retrieved from memory at each step. In contrast to "sparse" parameters such as embeddings, other parameters are "dense": they are involved at each step (an exception is conditional computation models, see §1.1), and consequently must be stored on limited, high-bandwidth GPU/TPU memory. Sparse parameters can potentially be stored on off-chip memory, such as on CPU memory, disk, or even a remote parameter server [14], and retrieved in a lightweight fashion (typically a few kilobytes per embedding vector). Since the vocabulary size of the embedding table has no effect on the number of lookups or operations involved in each forward pass, it means that the embedding vocabulary can be scaled up arbitrarily with no increase in compute. Practically it is only limited by the device storage capacity, which is typically larger than GPU/TPU memory by a factor of 10 (CPU memory) to 100 (disk), and virtually unlimited for distributed clusters [14].

In this work, we present LookupLM, a simple method to scale up the embedding vocabulary in recurrent language models in such a way as to improve modeling quality, *without* changing the token vocabulary. At each step, LookupLM hashes the sequence of the previous n tokens, draws a vector from a large n-gram embedding lookup table, and concatenates it with the input to the RNN cell, as shown in Figure 1. Since the number of possible n-gram sequences grows exponentially with n, the n-gram embedding vocabulary size can grow arbitrarily large while commensurately improving its capacity to embed the previous sequence. Intuitively this should especially improve modeling of the long tail in subword models, since the n-gram embedding helps with short-range dependencies such as spelling out rare words.

LookupLM achieves a log perplexity decrease of 2.44 over a standard language model with minimal increase in dense model parameters or floating point operations, and it achieves a 23.4% relative improvement on long tail speech recognition WER when integrated via shallow fusion with an E2E speech model. We perform various studies to understand the factors that contribute to LookupLM's success.

1.1. Related Work

While to our knowledge no work has systematically experimented with scaling up the embedding vocabulary of language

Copyright © 2021 ISCA

http://dx.doi.org/10.21437/Interspeech.2021-340

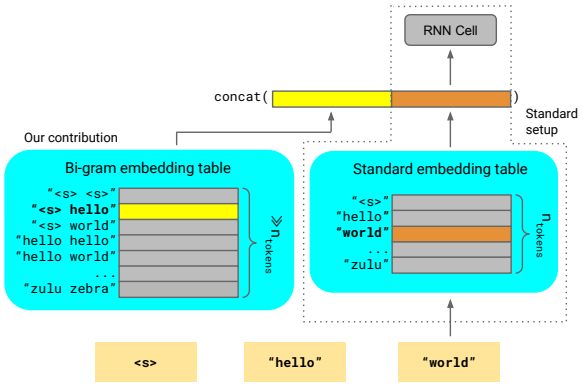


Figure 1: *We concatenate an n-gram embedding of the previous tokens (yellow vector) with the input to an RNN cell (red vector). The n-gram embedding table can be scaled to arbitrarily many rows since the number of unique n-grams is exponential.*

models to arbitrarily large sizes (for a fixed token vocabulary), our paper follows several themes of prior publications. In conditional computation [15], different blocks of parameters are turned on and off based on each example, but current implementations still place these sparse parameters on GPU/TPU memory [13]. The idea of integrating an off-chip datastore during inference to improve language modeling was pursued in kNN language models [16], where a nearest neighbor search is performed in embedding space against a large key-value datastore of (sentence prefix, embedding) pairs rather than an embedding lookup. Production systems with embedding lookup tables stored on SSD have been demonstrated for ad recommendation with a large number of entities (users, items, etc.) [14]; we experiment with large embedding vocabularies based on the composition of a *small* number of entities (wordpiece tokens). There have been a few variations of fast embedding compositions for language decoding. [17] and [18] generated word features by summing up a combination of the word's constituent unigrams, bigrams, and trigrams. An even more general transformation is to use an RNN to consume a token sequence to produce a fixed-length embedding [19], but this requires increasing dense parameters. Finally the use of n-gram embedding lookups have been demonstrated in RNN-T speech recognition decoders. Both [20] and [21] showed that feeding into the RNN-T joint network a simple bigram embedding of the previous two tokens, was sufficient to get similar WER as a recurrent prediction network.

2. Method

2.1. N-gram embedding

An embedding table is an $U \times E$ matrix where E is the embedding dimension and U, the number of embeddings, is typically equal to V, the number of unique tokens. We wish to disentangle U from V so that that it can be arbitrarily scaled up without affecting the model output space. Similar to n-gram language models, we make the assumption that the previous n-token[1] sequence can be used as a good signal for what the next token should be, and thus desire a way to embed that n-gram. It would be infeasible to instantiate a unique embedding for every n-gram, since the number of unique n-grams for a token vocabulary V scales exponentially as V^n. Instead, we predetermine an embedding vocabulary size U based on available storage, and assign each n-gram an embedding ID via a modular hash. Specifically, the token ID sequence $(t_0, \cdots, t_{n-1})$ is assigned an embedding ID as follows.

$$\texttt{ngram2id}(t_0, \cdots, t_{n-1}) = \left(\sum_{i=0}^{n-1} t_i V^i\right) \bmod U \quad (1)$$
$$\forall\, t_i \in [0, V)$$

Modular hashing necessitates collisions; arbitrarily different n-grams will be hashed to the same embedding. As we will show in §4.2, the performance improves as we reduce collisions by increasing U.

2.2. Injecting embeddings into the language model

Once an n-gram embedding vector is obtained, it is concatenated with the input vector before being fed into the RNN cell, as shown in Figure 1 for bi-grams. This causes the RNN to increase dense parameters in order to process the elongated input vector, but the scaling is linear rather than quadratic since the RNN output dimensionality remains fixed. We conjecture that the context information from the n-gram embedding would be useful not only at the input to the RNN stack, but at intermediate layers too, just as intermediate RNN cells receive context information via the hidden state specific to that layer. Therefore, we inject (concatenate) an n-gram embedding to the input activations of every layer, drawing each from an embedding table specific to that layer.

2.3. Integrating the LM into the speech model

For ASR evaluation, the E2E model used is a wordpiece Conformer-encoder [22], HAT-factorized [21] RNN-T decoder [1]. We integrate our language models with an E2E speech model in two steps: (1) obtain an effective E2E likelihood by separating out the speech model's log-posterior from its internal language model score via HAT factorization [21]:

$$\log p(x|y) \approx \log p(y|x) - \lambda_2 \log p_{ILM}(y),$$

and (2) add the LM log-posterior score:

$$y^* = \underset{y}{\operatorname{argmax}} \left[\log p(y|x) - \lambda_2 \log p_{ILM}(y) + \lambda_1 p_{LM}(y)\right].$$

The interpolation weights (λ_1, λ_2) are determined via a black-box optimization service [23] where the objective was the WER on a 50:50 mix of voice search queries and TTS-generated sentences with rare proper nouns from the Google Maps domain.

3. Setup

3.1. Model

Our baseline is a 4096-wordpiece RNN language model implemented in Lingvo on Tensorflow [24, 25]. Its architecture is a 2-layer, 512-width, layer-normalized LSTM with an input embedding (not to be confused with the n-gram embedding) dimensionality of 96. When not explicitly stated otherwise, our n-gram embeddings have a dimensionality of 2048, and n is 4.

3.2. Training

Our LM training dataset, identical to that of [4], is a sample of anonymized Google Maps queries, preprocessed with misspelling removal against a whitelist of 1 million common words,

[1] Note that we deviate slightly from standard naming convention; n-gram LMs use the previous $n-1$ tokens rather than n.

Table 1: *Main results (a) and ablation studies (b-e)*

	Language Model	# Params (M) Dense	# Params (M) Sparse	Log perplexity per word Head	Log perplexity per word RareASR	Log perplexity per word RareALL	WER (%) Head	WER (%) RareASR	WER (%) RareALL
(a)	B0: No LM	0.0	0.0	-	-	-	9.6	67.4	77.2
	B1: Base LM	5.5	0.4	7.09	17.22	22.62	7.1	56.8	80.3
	B2: Large LM	59.5	0.4	**6.70**	14.91	**20.16**	**6.2**	43.8	71.0
	E8: Base LM + Lookup Table	9.6	805.7	6.71	14.91	20.18	6.3	**43.5**	**69.1**
(b)	E0: Lookup-4k-2048	22.2	25.6	6.90	15.80	21.40	6.5	50.2	76.9
	E1: Lookup-8k-2048	22.2	50.7	6.89	15.69	21.27	6.6	48.3	75.6
	E2: Lookup-16k-2048	22.2	101.1	6.83	15.42	20.89	6.5	47.1	73.8
	E3: Lookup-33k-2048	22.2	201.7	6.79	15.20	20.56	6.4	45.4	72.6
	E4: Lookup-66k-2048	22.2	403.0	6.77	15.05	20.42	6.4	43.7	70.8
	E5: Lookup-131k-2048	22.2	805.7	**6.72**	**14.89**	**20.18**	**6.2**	**43.5**	**69.1**
(c)	E6: Lookup-131k-2048	22.2	805.7	6.72	14.89	20.18	6.2	**43.5**	**69.1**
	E7: Lookup-262k-1024	13.8	805.7	**6.70**	**14.84**	**20.05**	**6.0**	44.1	70.3
	E8: Lookup-524k-512	9.6	805.7	6.71	14.91	20.18	6.3	**43.5**	**69.1**
	E9: Lookup-1049k-256	7.5	805.7	6.71	14.99	20.30	6.2	44.4	69.9
	E10: Lookup-2097k-128	6.5	805.7	6.75	15.12	20.50	6.3	44.9	69.8
	E11: Lookup-4194k-64	6.0	805.7	6.88	15.57	21.16	6.5	46.5	72.4
	E12: Lookup-8389k-32	5.7	805.7	6.83	15.61	21.00	6.4	48.2	73.1
	E13: Lookup-16777k-16	5.6	805.7	6.89	15.95	21.32	6.7	48.9	74.8
(d)	E14: Lookup-33k-2048-2	22.2	201.7	**6.78**	15.23	20.65	**6.2**	46.6	74.1
	E15: Lookup-33k-2048-3	22.2	201.7	6.79	15.24	20.66	**6.2**	46.4	73.4
	E16: Lookup-33k-2048-4	22.2	201.7	6.79	**15.20**	**20.56**	6.4	**45.4**	**72.6**
	E17: Lookup-33k-2048-5	22.2	201.7	6.79	15.24	20.67	6.3	46.7	73.8
	E18: Lookup-33k-2048-6	22.2	201.7	6.80	15.22	20.66	6.4	45.5	72.8
(e)	E19: Lookup-33k-2048	22.2	201.7	6.79	15.20	20.56	6.4	45.4	72.6
	E20: + Incl. curr. token	22.2	201.7	6.78	15.16	20.58	6.3	45.6	72.6
	E21: + Layer0 injection only	9.6	67.5	6.84	15.40	20.95	6.5	46.3	74.3

and with $\log(n)$ scaling of sentence frequencies to ensure representation on the tail. We trained for 300k steps using the Adam optimizer with exponentially decaying learning rate schedule and a batch size of 16384 across 64 TPUv3 accelerator chips. For ASR evaluation, our RNN-T model was trained on 290M audio-text pairs spanning the domains of search, farfield, telephony, and YouTube [26]. All parameters, including the embedding tables, are initialized from scratch; we do not load pretrained embeddings.

3.3. Evaluation

We report language model log perplexity per word as well as speech recognition word error rate (WER) on 3 test sets of 10,000 sentences from the same domain as the training data but curated in different ways:

- Head: simply a held-out part of the LM training set, measuring overall accuracy.
- RareASR: held-out sentences containing a word that is "rare"—which we henceforth define as appearing *at most* 5 times—in the ASR data but not rare in the LM data. It measures the ability of LM integration to help ASR models learn words that were absent or infrequent in the speech data.
- RareALL: held-out sentences containing a word that is rare in both the ASR and LM data. It measures accuracy on the long tail.

For each sentence in RareASR and RareALL, we report LogPP and WER on only the rare word portion of the sentence—even though the entire sentence is passed to the model—in order to extract tail performance. For ASR evaluations, we synthesize TTS audio for each of the test sets. We have separately found that performance on TTS audio is correlated with recorded audio and thus believe that TTS audio is sufficient for measuring the relative performance gains of the LookupLM architecture.

Table 2: *Decoding samples picked from the top 5 wins or losses.*

LookupLM losses		LookupLM wins	
Base LM	LookupLM	Base LM	LookupLM
Funway	fun way	they would weigh	faywood Way
Foxboro	foxborough	Louisville Kentucky	Louisville Kentucky
yearly Ball	yearly	I'll go away	Algoa Bay
Des Moines	Baltimore		

4. Results

4.1. Comparison to baselines

In Table 1a, we evaluate language model log perplexity and downstream WER against the number of dense and sparse parameters it requires. Our best LookupLM model (E8: Base LM + Lookup Table) contains a embedding vocabulary of 524k and dimensionality of 512. We compare its results to 3 baselines:

- B0: No LM - ASR predictions without LM integration.
- B1: Base LM - The default LM described in §3.1 without n-gram embeddings. It has the same architecture as LookupLM minus the lookup table and is the main baseline.
- B2: Large LM - Baseline B1 but with 10x more parameters by having 2048 input/output activations instead of 512. It shows how much one would need to increase dense parameters in order to achieve similar results to LookupLM.

All models significantly outperform B0, indicating the ASR model is ripe for LM integration. LookupLM (E8) achieves noticeable gains compared to Base LM (B1). On the Head test set, the log perplexity decreases by 0.8 and the WER by 11.2% relative to the base. On the RareALL test set, log perplexity decreases by 2.44 and WER by 13.9% relative. Finally, on the RareASR test set the WER decreases by 23.4% relative, showing that LookupLM is an effective architecture for improving long tail speech recognition with sufficient text-only data. Table 2 shows some decoding samples.

We now compare with Large LM (B2) to see how much conventional model scaling is required to achieve the same gains as LookupLM. B2 has similar performance as LookupLM but requires 6.2 times more dense parameters and hence 6.2× the FLOPs, showing that using a large, off-chip lookup table can effectively trim on-chip model size and reduce computation.

Table 3: *Multidomain results.*

Language Model	# Params (M) # Dense	# Sparse	WER (%) Voice Search	Maps	News	Play	Search	YouTube
B3: Multidomain LM	126.7	0.4	6.1	19.8	9.0	45.3	32.5	28.0
E22: Multidomain LM + Lookup Table	135.1	67.5	6.1	**19.4**	**8.8**	**44.6**	**32.0**	**27.6**

4.2. Scaling of embedding vocabulary

We now study model performance scaling with embedding vocabulary size through an ablation study in Table 1b, which scales the vocabulary from a size of 4k (E0) to 131k (E5). While the embedding vocabulary can be much larger if stored off-TPU, for this demonstration we stored the embedding table on TPU and thus 131k was the limit of available memory at 2048-dimensional embeddings. We name each model according to the template, Lookup-$VOCAB-$DIM, where $VOCAB is the embedding vocabulary size and $DIM is the embedding dimensionality. The performance increases with embedding vocabulary, with no sign of saturating. The scaling of relative gain in this range roughly follows a power law with an exponent of $\sim$0.1 for linear perplexity and $\sim$0.5 for WER.

4.3. Vocabulary size vs. embedding dimension tradeoff

Given a fixed amount of sparse parameters, there is a tradeoff between the embedding vocabulary and embedding dimension. Table 1c shows various LookupLM models with a fixed number of sparse parameters but differing amounts of rows and columns. The results show a sweet spot around E6-E8, which have embedding dimensionalities above 256. Among these, we pick E8 as our best model since it requires the fewest dense parameters.

4.4. Effect of n-gram order

The n-gram order, or simply n, determines the context size for the embedding. A larger n-gram order entails a larger context, but also results in more collisions given that the number of possible n-grams increases with n. Increasing n-gram order also reduces the number of times any particular n-gram is updated. We aim to look for a sweet spot in Table 1d, where the naming template is Lookup-$VOCAB-$DIM-$ORDER. We find that there is only a slight variation in performance with n-gram order for values between 2 (E14) and 6 (E18), with 4 (E16) slightly being the best.

4.5. Design space variations

Finally, we consider two variations of our model architecture:

- E20: Instead of the n-gram embedding entailing the previous n tokens, we have it entail the current token plus the previous $n-1$ tokens.
- E21: Instead of injecting the n-gram embedding at every layer, we inject it only at the first layer.

Table 1e shows that shifting the context window of the n-gram embedding to include the current token (E20) makes no difference compared to not shifting (E19). It's advantageous to not include the current token because then the n-gram embedding can be pre-fetched before the current step, reducing latency. Removing the multi-layer injection (E21) reduces performance slightly, but also reduces the number of dense (and sparse) parameters significantly. Exploring the tradeoffs between single- vs. multi-layer injection may lead to a more optimal setup in terms of performance and parameter count, but we leave that to future work.

4.6. LookupLM maximizes resource utilization

We have separated *dense* and *sparse* parameters throughout this paper because they represent the more fundamental time and space complexities. In an LSTM, the dense parameters are each accessed once per step, so the number of dense parameters is directly correlated with the number of floating point operations (FLOPs), i.e. time complexity. Meanwhile, the storage requirement, i.e. space complexity, is dominated by the sparse parameters (increasing dense parameters also increases memory, but by a relatively much smaller amount). In typical networks, the time and space complexity are inextricably linked: improving performance requires increasing both parameter count *and* FLOPs. LookupLM provides a way of achieving performance by increasing memory without increasing FLOPs. This can allow for taking advantage of excess resources (typically RAM or disk memory). Table 4 shows the WER for a grid of independent memory and FLOPs configurations for LookupLM, which can be useful for determining the tradeoff between various resource allocations. Performance relative to Base LM (B1) improves by shown amounts when resources along either axis (memory or FLOPs) are increased.

Table 4: *Relative WER (%) on RareASR at various complexities.*

Mem. (GB)	# FLOPs (M) # Dense (M) # Sparse (M)	19.3 9.6	38.8 19.4	65.5 32.8	99.2 49.6	140.0 70.0
0.2	50.7	-15.9	-20.7	-21.2	-24.5	-23.4
0.4	101.1	-18.0	-20.3	-21.6	-25.1	-24.9
0.8	201.7	-19.7	-21.3	-24.2	-24.3	-24.2
1.6	403.0	-22.5	-23.9	-24.8	-23.9	-26.4
3.2	805.7	-24.0	-23.3	-25.2	-24.3	-26.5

4.7. Preliminary multidomain results

Table 3 shows preliminary results on a production-scale LSTM LM (4-layer, 2048-width) trained for 1M steps on $\sim$200B text examples from the domains of Voice Search, Maps, News, Play, Search, and YouTube. Besides Voice Search, all evaluation test sets in Table 3 are curated to measure long tail performance. With a modest Lookup Table size (67.5M parameters), we achieve modest WER gains across all the domains on the long tail, without harming Voice Search performance.

5. Conclusion

While our method here is already effective, it can still be improved. For example, we could assign the most common n-grams a unique embedding ID while modular hashing the rest. Further we could apply a per-embedding learning rate schedule which is based on the number of times it's been updated rather than on global step.

6. Acknowledgements

We thank Bo Li, Ruoming Pang, and Chen Zhu for helpful discussions and technical guidance.

7. References

[1] A. Graves, "Sequence transduction with recurrent neural networks," *arXiv preprint arXiv:1211.3711*, 2012.

[2] W. Chan, N. Jaitly, Q. V. Le, and O. Vinyals, "Listen, attend and spell," *arXiv preprint arXiv:1508.01211*, 2015.

[3] D. Zhao, T. N. Sainath, D. Rybach, P. Rondon, D. Bhatia, B. Li, and R. Pang, "Shallow-fusion end-to-end contextual biasing," *Proc. Interspeech 2019*, pp. 1418–1422, 2019.

[4] C. Peyser, S. Mavandadi, T. N. Sainath, J. Apfel, R. Pang, and S. Kumar, "Improving tail performance of a deliberation e2e asr model using a large text corpus," *Proc. Interspeech 2020*, pp. 4921–4925, 2020.

[5] 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 *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2018, pp. 1–5828.

[6] S. Toshniwal, A. Kannan, C.-C. Chiu, Y. Wu, T. N. Sainath, and K. Livescu, "A comparison of techniques for language model integration in encoder-decoder speech recognition," in *2018 IEEE spoken language technology workshop (SLT)*. IEEE, 2018, pp. 369–375.

[7] A. Sriram, H. Jun, S. Satheesh, and A. Coates, "Cold fusion: Training seq2seq models together with language models," *arXiv preprint arXiv:1708.06426*, 2017.

[8] C. Weng, C. Yu, J. Cui, C. Zhang, and D. Yu, "Minimum bayes risk training of rnn-transducer for end-to-end speech recognition," *Proc. Interspeech 2020*, pp. 966–970, 2020.

[9] J. Kaplan, S. McCandlish, T. Henighan, T. B. Brown, B. Chess, R. Child, S. Gray, A. Radford, J. Wu, and D. Amodei, "Scaling laws for neural language models," *arXiv preprint arXiv:2001.08361*, 2020.

[10] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.

[11] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[12] D. Lepikhin, H. Lee, Y. Xu, D. Chen, O. Firat, Y. Huang, M. Krikun, N. Shazeer, and Z. Chen, "Gshard: Scaling giant models with conditional computation and automatic sharding," *arXiv preprint arXiv:2006.16668*, 2020.

[13] W. Fedus, B. Zoph, and N. Shazeer, "Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity," *arXiv preprint arXiv:2101.03961*, 2021.

[14] W. Zhao, D. Xie, R. Jia, Y. Qian, R. Ding, M. Sun, and P. Li, "Distributed hierarchical gpu parameter server for massive scale deep learning ads systems," *arXiv preprint arXiv:2003.05622*, 2020.

[15] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017.

[16] U. Khandelwal, O. Levy, D. Jurafsky, L. Zettlemoyer, and M. Lewis, "Generalization through memorization: Nearest neighbor language models," in *International Conference on Learning Representations*, 2019.

[17] J. A. Botha, E. Pitler, J. Ma, A. Bakalov, A. Salcianu, D. Weiss, R. McDonald, and S. Petrov, "Natural language processing with small feed-forward networks," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, 2017, pp. 2879–2885.

[18] H. Zhang, R. Sproat, A. H. Ng, F. Stahlberg, X. Peng, K. Gorman, and B. Roark, "Neural models of text normalization for speech applications," *Computational Linguistics*, vol. 45, no. 2, pp. 293–337, 2019.

[19] W. Ling, I. Trancoso, C. Dyer, and A. W. Black, "Character-based neural machine translation," *arXiv preprint arXiv:1511.04586*, 2015.

[20] M. Ghodsi, X. Liu, J. Apfel, R. Cabrera, and E. Weinstein, "Rnn-transducer with stateless prediction network," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 7049–7053.

[21] E. Variani, D. Rybach, C. Allauzen, and M. Riley, "Hybrid autoregressive transducer (hat)," in *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2020, pp. 6139–6143.

[22] A. Gulati, J. Qin, C.-C. Chiu, N. Parmar, Y. Zhang, J. Yu, W. Han, S. Wang, Z. Zhang, Y. Wu *et al.*, "Conformer: Convolution-augmented transformer for speech recognition," *Proc. Interspeech 2020*, pp. 5036–5040, 2020.

[23] D. Golovin, B. Solnik, S. Moitra, G. Kochanski, J. Karro, and D. Sculley, "Google vizier: A service for black-box optimization," in *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, 2017, pp. 1487–1495.

[24] J. Shen, P. Nguyen, Y. Wu, Z. Chen, M. X. Chen, Y. Jia, A. Kannan, T. Sainath, Y. Cao, C.-C. Chiu *et al.*, "Lingvo: a modular and scalable framework for sequence-to-sequence modeling," *arXiv preprint arXiv:1902.08295*, 2019.

[25] Specifically, our implementation is based off of the Lingvo model spec `RnnLm` at https://github.com/tensorflow/lingvo/blob/master/lingvo/tasks/lm/layers.py.

[26] A. Narayanan, R. Prabhavalkar, C.-C. Chiu, D. Rybach, T. N. Sainath, and T. Strohman, "Recognizing long-form speech using streaming end-to-end models," in *2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*. IEEE, 2019, pp. 920–927.