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

Hierarchical Multi-Stage Word-to-Grapheme Named Entity Corrector for Automatic Speech Recognition

Abhinav Garg, Ashutosh Gupta, Dhananjaya Gowda, Shatrughan Singh, Chanwoo Kim

Samsung Research Korea

{abhinav.garg, g.ashu, d.gowda, shatrughan.s, chanw.com}@samsung.com

Abstract

In this paper, we propose a hierarchical multi-stage word-to-grapheme Named Entity Correction (NEC) algorithm. Conventional NEC algorithms use a single-stage grapheme or phoneme level edit distance to search and replace Named Entities (NEs) misrecognized by a speech recognizer. However, longer named entities like song titles cannot be easily handled by such a single stage correction. We propose a three-stage NEC, starting with a word-level matching, followed by a phonetic double metaphone based matching, and a final grapheme level candidate selection. We also propose a novel NE Rejection mechanism which is important to ensure that the NEC does not replace correctly recognized NEs with unintended but similar named entities. We evaluate our solution on two different test sets from the *call* and *music* domains, for both server as well as on-device speech recognition configurations. For the on-device model, our NEC outperforms an n-gram fusion when employed standalone. Our NEC reduces the word error rate by 14% and 63% relatively for *music* and *call*, respectively, when used after an n-gram based biasing language model. The average latency of our NEC is under 3 *ms* per input sentence while using only $\sim$1 MB for an input NE list of 20,000 entries.

Index Terms: Named Entity Corrector, Spell Correction, Automatic Speech Recognition, Domain biasing, NE Rejection

1. Introduction

End-to-end Automatic Speech Recognition (ASR) models encompass the separate components such as acoustic, lexicon, and language model of a conventional ASR model into a single network [1, 2, 3, 4, 5]. It enables training and optimizing the different components together, thereby simplifying the development of ASR systems. Listen, Attend and Spell (LAS) [6], and Recurrent Neural Network Transducer (RNN-T) [7, 8] are examples of end-to-end architectures, which have seen wider adoption for both server-based and on-device deployment owing to their competitive performance compared to conventional models for a variety of tasks [9, 10, 5, 11].

End-to-end models are much simpler to train and can achieve a competitive word error rate (WER) compared to conventional ASR systems [12]. However, the misrecognition of rarely occurring words such as named entities (NEs) is a well-known shortcoming of end-to-end models [13]. This is because end-to-end models are trained using a far smaller number of audio-text pairs compared to the large text-only data used to train language models in the conventional ASR systems. An ASR is the first step of any spoken dialog system, and errors in recognition of hypothesis especially named entities cascades in subsequent natural language understanding (NLU) module. Hence a variety of research [14, 15, 16, 17, 18, 19] is being done by researchers in both NLU and ASR community to address this problem.

The problem of NE misrecognition is particularly severe for domains like music, contacts, location, and so on having a large number of infrequent proper nouns. One approach to incorporate domain information into ASR is using on-the-fly rescoring [20, 21] with a domain-specific n-gram language model (LM). Another similar approach is to do a shallow fusion with a domain biased Weighted Finite-State Transducer (WFST) [22, 23, 24]. Some previous works have also explored incorporating the domain-specific LMs into the end-to-end network using different fusion techniques such as shallow,deep [25, 26] and cold [27] fusion. [28, 29] used deep learning approaches for named entity correction whereas [23, 30] also explored using deep neural architectures for contextualizing ASR output where they jointly optimize the ASR along with the contextual embeddings.

While the above methods significantly improve the performance of domain ASR, each of them has some shortcomings. They might require large amounts of domain-specific training data, additional training time; moreover, they might add high latency, or increase the model size making the overall ASR model bulky and slow. This restricts their usage especially for on-device applications with limited memory and latency constraints. Hence, simple edit distance based algorithmic spell correction methods have been proposed [16, 31, 32, 33]. While these are fast and light-weight, they might also suffer from a severe problem of replacing a correct ASR output with its close and confusing match from the corresponding input NE list.

In this paper, we propose a hierarchical multi-stage word-to-grapheme Named Entity Correction (NEC) algorithm. Our NEC uses weighted edit distance and has a NE Rejection module based on ASR beam search outputs to ensure that NEC does not replace any correctly recognized named entity. We follow a multi-stage hierarchical matching approach starting from word to grapheme. At each stage, we select only a sub-set of candidate NEs based on the normalized edit distance which constrains the size of the NE list to be processed in the subsequent stage. In the first two stages, we select candidates based on word level and phoneme level information. In the third stage, we calculate the weighted edit distance of the remaining candidates and choose a candidate with minimum distance. Finally, we verify if this candidate is indeed a better replacement than the ASR output itself. We focus on building light-weight and fast solution that can be used as a standalone domain biasing solution in constrained environments or it can be applied as an additional step along with other domain biasing solutions such as those based on WFST, n-gram, and so on.

The proposed algorithm is evaluated on three test sets - *Call*, *Music*, and *Open Domain* with three different model configurations - the on-device model, on-device model + n-gram, the server model + WFST. For the on-device model, proposed NEC outperforms n-gram fusion when used standalone and gives 14% and 63% relative WER improvement for *music* and

http://dx.doi.org/10.21437/Interspeech.2020-3174

call test sets respectively when used with an n-gram LM.

2. ASR Architecture

For our baseline ASR we use an Attention based Encoder-Decoder (AED) architecture, composed of an encoder, a decoder, and an attention block [6]. Our AED system is constructed using a `Tensorflow` 2.0 Keras model. This model is trained using an *in-house* trainer built using `Tensorflow` 2.0-Keras APIs and the `tf.data` pipeline. An AED model takes speech x as an input and outputs a beam of possible hypotheses, $y^0, \ldots, y^{b-1}$ along with their probability scores (beam scores) $p^0, \ldots, p^{b-1}$, where b is the beam size, $p^i = P(y^i|x)$, and $p^0 > p^1 \cdots > p^{b-1}$. In this work, we use server and on-device AED models from our previous works [34, 21] for evaluating our NEC solution.

After obtaining the hypothesis from the AED model, we used a simple keyword-based domain classifier for deciding the domain of the AED output. Based on this domain, we apply an optional domain-specific biasing method as explained in Section 4. Finally, we extract the NE, n^0 from the top output y^0 of the AED model using regular expression (regex) matching with handcrafted rules. If y^0 contains multiple NEs, we process each of them separately with our NEC algorithm. We refrain from using neural domain classifiers [35] and NE extractors [36] to keep our post-processing lightweight and fast. Neural domain classifiers and NE extractors can be further used to improve the overall performance.

3. Named Entity Corrector

A block schematic of the proposed multi-stage Named Entity Corrector (NEC) is shown in Fig. 1. Given ASR outputs $(y^0, \ldots, y^{b-1})$, $(p^0, \ldots p^{b-1})$ and an NE list L_0 containing possible alternatives for NE in ASR output, the purpose of an NEC is to replace the NE in the ASR output with a better alternative from the input NE list L_0 if available.

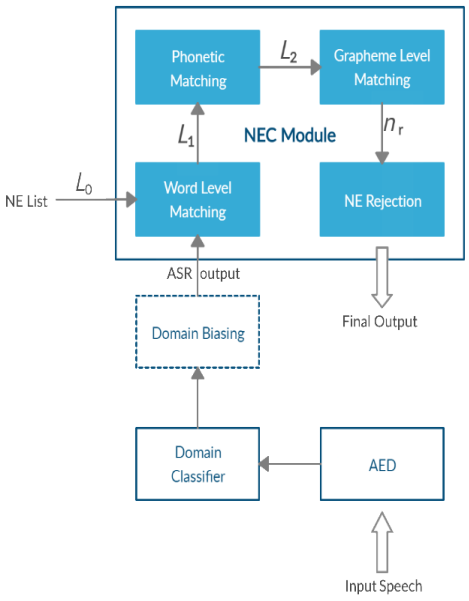

Figure 1: *Block schematic of the proposed Named Entity Corrector (NEC)*

We follow a hierarchical multi-stage, multi-granular approach starting from word to grapheme. At each stage, we select only a subset of candidate NEs based on the normalized edit distance which constrains the size of the NE list to be processed in the subsequent stage. In the first stage, a word level edit distance is used to shortlist NE candidates, which are refined at the second stage using a double meta-phone based edit distance. In the third stage, we calculate the weighted edit distance for all remaining candidates and choose a candidate with minimum distance. Finally, we check if the obtained candidate is indeed a good replacement for the input NE using its beam information. Now, we explain each of these stages in details.

3.1. Word level matching

At the first stage, we select candidate NEs from the NE list L_0 based on their normalized word-level edit distance with NE n^0 corresponding to the top ASR hypothesis y^0. In particular, given an input NE list L_0 and NE n^0, L_1 is the list of all NEs $n_i \in L_0$ such that:

$$d_i^1 = \mathit{WordEd}(n_i, n^0) \tag{1}$$

$$L_1 = \{n_i | d_i^1 < \epsilon_1, \forall n_i \in L_0\} \tag{2}$$

where $\mathit{WordEd}(n_i, n^0)$ is a word level edit-distance between n_i and n^0 normalized by number of words in n^0. ϵ_1 is the threshold we use for selecting potential replacement candidates from the input NE list L_0.

3.2. Phonetic matching

In the next stage we calculate phonetic similarity between NEs in L_1 and the input NE n^0. For doing this, we use Double MetaPhone (DMP) algorithm [37]. DMP maps an input token to its **approximate** phonetic representation using predefined rules and heuristics. It was primarily developed for English but was later extended to other languages as well.

We compute the DMP code for a given multi-word NE by obtaining the DMP code for each word in the NE and later concatenating them with a space. Similar to stage one, we calculate normalized edit distance between DMP code for input NE n^0 and each NEs in L_1. Based on this we create a next list of NE candidates $L_2 \subset L_1$, such that :

$$d_i^2 = ED(\mathit{DMP}\ (n_i), \mathit{DMP}(n^0)) \tag{3}$$

$$L_2 = \{n_i | d_i^2 < \epsilon_2, \forall n_i \in L_1\} \tag{4}$$

where $\mathit{DMP}(n)$ is DMP code for n computed as described above. $ED(a, b)$ is the grapheme level edit distance between a and b normalized by the length of b. ϵ_2 is the filtering parameter used to reduce the size of L_2.

3.3. Grapheme level matching

At the third stage, we calculate grapheme based edit distance d_i^3 as $d_i^3 = ED(n_i, n^0)$ for each candidate NE $n_i \in L_2$. For choosing our final replacement NE, n_r, we calculate weighted edit distance for each candidate NE in L_2 as :

$$d_i^f = w_1 * d_i^1 + w_2 * d_i^2 + w_3 * d_i^3 \tag{5}$$

where w_1, w_2, w_3 are the weights given to word level, phonetic level and grapheme level edit distance respectively and $w_1 + w_2 + w_3 = 1$. $n_r \in L_2$ is the final suggestion NE with weighted edit distance d_r^f s.t. $d_r^f = \min(d_i^f) \forall n_i \in L_2$. If $d_r^f < \epsilon_3$, our NEC passes n_r to the next stage for final verification else our NEC terminates with no suggestion.

3.4. NE Rejection

A simple edit distance based NEC has a shortcoming of replacing NEs which are correctly predicted by the ASR. This problem is more profound for domains like call, music, location. For such domains either it is difficult to get an exhaustive NE list, or the NE list might be too large resulting in resource-intensive and slow NEC. In such cases instead of the correct NE, the NE lists might contain similar but incorrect replacements which can degrade the performance of the ASR. Hence, apart from improving domain-specific performance, our main focus was to ensure that our NEC solution does not cause any degradation to ASR output irrespective of the domain or the amount of NEs used. We use the beam information of the ASR to accomplish this. In this module, we verify whether n_r is indeed a suitable replacement for n^0.

For this stage first we extract $n^1, \ldots, n^{b-1}$, where n^i is the NE present in top i^{th} hypothesis of the ASR, y^i. For extracting n^i we use the following approaches. First we try to extract n^i using the same regex rule as n^0. As y^i might be erroneous, we use a fall back NE extractor, which aligns y^0 and y^i using minimum edit distance alignment and extracts NE n^i from y^i corresponding to n^0 in y^0.

If n_r matches with any of the beam NEs, $n^0 .... n^{b-1}$ then we straightaway accept n_r as a replacement for n^0. Otherwise for both n^0 and n_r we calculate rejection score, r_n as :

$$r_n = \sum_{i=0}^{b-1} p^i * (w_1 * WordEd(n^i, n) + w_2 * ED(DMP(n^i), DMP(n)) + w_3 * ED(n^i, n)) \quad (6)$$

Where n can be either n^0 or n_r, p^i represents probability for ASR output y^i. Finally, if $r_{n^0} > r_{n_r}$, we accept n_r as a suitable replacement for n^0 otherwise we reject n_r and output no suggestion for n^0.

4. Experimental Setup

We use two AED systems to evaluate our NEC algorithm. The first model is similar to [34], which is trained with anonymized open domain corpus consisting of around 10K hours of transcribed speech. The second model is similar to [21], which is essentially the first model compressed and quantized for on-device applications. Both models use subword BPE units [38] as their output and have an output vocabulary size of 10k. We call the first model as the server model and the second model as the on-device model. All the models in this paper are trained using an *in-house* trainer built using the Tensorflow 2 Keras APIs.

The on-device model has 20 times fewer parameters than the server model as it is SVD [39, 9] compressed and quantized. We use beam search with a beam size of 8 for decoding of server model and beam size of 4 for the on-device model. Both these models use MoChA attention [21, 40] and are streaming with real-time decoding. We apply NEC only after the complete ASR output is available.

We use ϵ_1 and ϵ_2 as 0.5 and ϵ_3 as 0.25. We give highest weightage to grapheme level distance with $w_3 = 0.6$ while w_2 and w_1 are set to 0.25 and 0.15 respectively. These parameters can be further fine tuned with grid search and optimized for each domain. However, for consistency, we use the same set of parameters for all our experiments across various domains.

We perform experiments for 2 domains namely *Call* and *Music*. For *call* NE list, we collected around 20k most common English names. For *music* NE list, we used titles of about 400k most popular songs and names of 100k most popular artists. We experiment with 3 test sets - *Call*, *Music*, and *Open domain*. The *Call* test set contains about 5k utterances of call requests (e.g. `call becker mathewson`). The *Music* test set contains 20k utterances of requests to play music (e.g. `play ariana grande`, `play not afraid by eminem`). *Open domain* contains 1.5k utterances out of which about 4% are from music or call domain. For *Open domain* test set depending upon classified domain we use either *Call* post-processing (biasing + NEC) or *Music* post processing or none.

We also optionally use WFST rescoring with the server model and n-gram rescoring with the on-device model. Both these domain biasing solutions are built from the text corpus obtained by expanding NE matching regex rules with all possible NEs in the NE list. We use fusion weights of 0.2 for n-gram LM and 0.25 for WFST fusion.

For edit distance calculations, we use Levenshtein distance [41] as our edit distance matrix. To perform ED calculations, we apply symmetric delete spelling correction algorithm, which is about six orders of magnitude faster than Norvig's [31] spelling corrector.

5. Results

In this section, we discuss the results obtained with our NEC algorithm. Table 1 contains the results and contributions for each module of our algorithm in terms of Word Error Rate (WER) and Sentence Error Rate (SER). We start with only grapheme level matching. After that, we add phonetic level matching using DMP, Word level matching and finally, we add NE Rejection. For all the results in Table 1 we used the default configuration of NEC parameters mentioned in Section 4.

We evaluate our algorithm on three test sets - *Call, Music* and *Open Domain*. For The *Call* test set, over 90% of the ground truth NEs are already present in the input NE list. The *Music* test set has about 50% of the ground truth NEs present in the input NE list. Finally, The *Open Domain* test set has only a few (about 4%) music or call domain sentences.

For each of these test sets, we evaluate 3 different scenarios. First, when NEC is used as the only domain-specific post-processing module for the on-device model. Secondly, we explore the use of NEC along with an n-gram domain LM. Finally, we explore the use of NEC along with a much larger server-side model having a WFST for domain biasing. Please note our NEC solution is the same for all these cases; however, the base model varies in terms of parameters and beam size.

For the *Call* test set, we observe that NE Rejection marginally degrades the performance. This degradation is expected because over 90% of the ground truth NEs are already present in the NE list. Therefore any rejection is presumably a bad rejection. Hence it becomes difficult for NE Rejection to make a decision. We obtain a minimum of 53% and a maximum of 69.4% relative WER improvement for the *call* test set.

The *Music* test set is the best application scenario for our NEC algorithm where only about 50% of the ground truth NEs are present in the NE list. However, the NE list is huge and may contain confusing incorrect replacements for other 50% of the ground truth NEs. As can be seen, NE Rejection consistently improves performance for this test set. For the server-side model, we obtain about 7.3% relative WER improvement without NE Rejection and about 8.6% with NE Rejection.

To further evaluate the usefulness of NE Rejection, we compute results for the *music* test set with only 2, 5, 10, 20% of the NE list selected randomly from the full NE list of 500k

Table 1: *Performance (in %) of different stages of NEC for various models on music, call and open domain testsets*

Testset	Model	Biasing	Baseline		+Grapheme NEC		+Phonetic NEC		+Word NEC		+NE Rejection	
			WER	SER	WER	SER	WER	SER	WER	SER	WER	SER
Music	On-device	None	12.20	46.85	10.12	35.27	9.58	**33.10**	9.68	33.60	**9.60**	33.15
		N-gram	10.26	37.97	9.36	32.17	8.96	30.61	8.90	30.46	**8.80**	**29.9**
	Server	WFST	6.12	24.6	6.05	23.61	5.81	22.41	5.67	21.80	**5.59**	**21.68**
Call	on-device	None	19.15	46.01	7.22	13.26	5.55	9.78	**5.48**	**9.50**	5.85	10.24
		N-gram	12.80	31.95	5.96	11.10	4.91	8.90	**4.69**	**8.60**	4.80	8.74
	Server	WFST	10.30	24.50	5.12	9.89	**4.64**	**8.51**	4.73	8.68	4.79	8.72
Open Domain	On-device	None	14.61	29.21	14.67	29.33	14.64	29.21	14.61	29.15	**14.59**	**29.15**
		N-gram	14.52	28.89	14.58	29.27	14.54	28.89	14.52	28.83	**14.49**	**28.77**
	Server	WFST	9.03	22.02	9.20	22.52	9.09	22.27	9.05	22.08	**8.90**	**21.57**

song titles and artist names (400k +100k). We used our server model, and the results were averaged over 5 runs, as shown in Table 2. As the size of the NE list increases the probability of ground truth NE being present in the list increases, however, the probability of close matching incorrect NE being present in the list also increases. For NEC without NE Rejection, the results improve initially with 2% of the NE list as there are not many confusing candidates. However, they degrade after that for 5% and 10% as the confusing candidates start to appear. Finally, the results start improving once enough number of ground truth NEs are present in the NE list. In contrast to this, the results for NEC with NE Rejection are consistently better than the baseline.

For *Open domain*, NEC without NE Rejection degrades the performance marginally. This is majorly due to NEs in sentences like: "`call previous number`", "`play some good song`" getting replaced with wrong NEs from the NE list. Such cases can be probably handled via a better domain classifier, but for us, NE Rejection ignores such misclassifications and improves the performance by 1.4% relative WER for server model.

As can be seen from Table 1, our algorithm achieves better performance than n-gram fusion for the on-device model for both *call* and *music* test sets. Also, when combined with other biasing techniques like n-gram or WFST fusion, it significantly improves the performance. Hence, our NEC can either be used as a replacement for n-gram in an on-device model. Or it can also be used as an additional step in the domain-specific post-processing pipeline of any ASR system.

The average latency of our algorithm is about **2-3 ms** per utterance averaged across all domains on a CPU with a clock rate of 2.6 Ghz and ∼9 Gb memory. The memory consumption is around 1 MB for *call* and around 40 MB for *music* which is 3.5 $\sim$ 4 times the size of NE list itself. This memory is required to store the data structure for the fast computation of edit distances. This data structure can either reside in memory or reside in the secondary storage and parts of it can be loaded on an on-demand basis. All our experiments use an in-memory data-structure. In terms of number of candidates, average length of L_1 is about 0.7% of L_0 and average length of L_2 is about 50% of L_1.

To evaluate the effectiveness of our fallback NE extractor, which extracts the NE using minimum edit distance alignment, we also computed the percentage of times fallback NE extractor was used. For server model with WFST, fallback NE extractor was used <5% as the beams were well-formed. However, for the on-device model, it was used about 8% for the n-gram case and about 11% for the no-bias case.

Table 2: *Performance (in %) of NEC experiments for music test set with various percentages of the full NE list averaged over 5 runs*

Percentage of NE list used	NEC w/o NE Rejection		NEC with NE Rejection	
	WER	SER	WER	SER
0	6.12	24.6	6.12	24.6
2	6.04 ↓	24.14	5.96 ↓	23.74
5	*6.43* ↑	*26.13*	5.99 ↓	23.88
10	*6.13* ↑	*24.63*	5.96 ↓	23.73
20	5.91 ↓	23.5	5.84 ↓	22.6
100	**5.67** ↓	**21.80**	**5.59** ↓	**21.68**

6. Conclusion

In this paper, we presented a hierarchical multi-stage word-to-grapheme Named Entity Correction (NEC) algorithm. We start from word-level matching, followed by a phonetic matching and finally a grapheme level candidate selection. We also presented a novel NE candidate rejection mechanism to prevent NEC from replacing a correctly recognized NE. We evaluate improvement obtained by each stage. We also perform experiments with various NE list sizes, where NEC without NE Rejection degrades the performance for a few list sizes; however, with NE Rejection, it always performs better than the baseline. The proposed NEC has a latency of 2-3ms and a memory footprint of about 3.5-4 times the size of the NE list. For a resource constraint on-device model, we show that our NEC outperforms n-gram fusion when used standalone and provides significant improvement when used along with an n-gram. Hence, the NEC can be used as a standalone domain biasing solution for resource constraint environment, or it can be applied as an additional step along with other domain biasing solutions such based on WFST, n-gram, and so on.

7. References

[1] D. S. Park, W. Chan, Y. Zhang, C.-C. Chiu, B. Zoph, E. D. Cubuk, and Q. V. Le, "SpecAugment: A Simple Data Augmen-

tation Method for Automatic Speech Recognition," in *Proc. Interspeech*, 2019.

[2] A. Garg, D. Gowda, A. Kumar, K. Kim, M. Kumar, and C. Kim, "Improved Multi-Stage Training of Online Attention-based Encoder-Decoder Models," in *Proc. ASRU*, 2019.

[3] D. Gowda, A. Garg, K. Kim, M. Kumar, and C. Kim, "Multi-task multi-resolution char-to-bpe cross-attention decoder for end-toend speech recognition," in *Proc. Interspeech*, 2019.

[4] C. Kim, M. Shin, A. Garg, and D. Gowda, "Improved vocal tract length perturbation for a state-of-the-art end-to-end speech recognition system," *Proc. Interspeech 2019*, pp. 739–743, 2019.

[5] D. Gowda, A. Kumar, K. Kim, H. Yang, A. Garg, S. Singh, J. Kim, M. Kumar, S. Jin, S. Singh, and C. Kim, "Utterance invariant training for hybrid two-pass end-to-end speech recognition," in *Proc. Interspeech*, 2020.

[6] W. Chan, N. Jaitly, Q. Le, and O. Vinyals, "Listen, attend and spell: A neural network for large vocabulary conversational speech recognition," in *Proc. ICASSP*, 2016.

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

[8] R. Prabhavalkar, K. Rao, T. N. Sainath, B. Li, L. Johnson, and N. Jaitly, "A comparison of sequence-to-sequence models for speech recognition." in *Proc. Interspeech*, 2017.

[9] A. Garg, G. Vadisetti, D. Gowda, S. Jin, A. Jayasimha, Y. Han, J. Kim, J. Park, K. Kim, S. Kim, Y. Lee, K. Min, and C. Kim, "Streaming on-device end-to-end asr system for privacy-sensitive voicetyping," in *Proc. Interspeech*, 2020.

[10] A. Kumar, S. Singh, D. Gowda, A. Garg, S. Singh, and C. Kim, "Utterance confidence measure for end-to-end speech recognition with applications to distributed speech recognition scenarios," in *Proc. Interspeech*, 2020.

[11] K. Kim, K. Lee, D. Gowda, J. Park, S. Kim, S. Jin, Y.-Y. Lee, J. Yeo, D. Kim, S. Jung *et al.*, "Attention based on-device streaming speech recognition with large speech corpus," in *Proc. ASRU*. IEEE, 2019.

[12] Y. He, T. N. Sainath, R. Prabhavalkar, I. McGraw, R. Alvarez, D. Zhao, D. Rybach, A. Kannan, Y. Wu, R. Pang *et al.*, "Streaming end-to-end speech recognition for mobile devices," in *Proc. ICASSP*, 2019, pp. 6381–6385.

[13] T. N. Sainath, R. Prabhavalkar, S. Kumar, S. Lee, A. Kannan, D. Rybach, V. Schogol, P. Nguyen, B. Li, Y. Wu, Z. Chen, and C. Chiu, "No need for a lexicon? evaluating the value of the pronunciation lexica in end-to-end models," in *Proc. ICASSP*, 2018.

[14] V. Yadav and S. Bethard, "A survey on recent advances in named entity recognition from deep learning models," *arXiv preprint arXiv:1910.11470*, 2019.

[15] R. C. De Amorim and M. Zampieri, "Effective spell checking methods using clustering algorithms," in *Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP*, 2013.

[16] A. Raghuvanshi, V. Ramakrishnan, V. Embar, L. Carroll, and K. Raghunathan, "Entity resolution for noisy asr transcripts," in *Proc. EMNLP*, 2019.

[17] M. Hulden, "Fast approximate string matching with finite automata," in *Sociedad Española para el Procesamiento del Lenguaje Natural*, 2009.

[18] Y. Bassil and M. Alwani, "Post-editing error correction algorithm for speech recognition using bing spelling suggestion," *arXiv preprint arXiv:1203.5255*, 2012.

[19] S. Zhang, M. Lei, and Z. Yan, "Automatic spelling correction with transformer for ctc-based end-to-end speech recognition," *arXiv preprint arXiv:1904.10045*, 2019.

[20] K. 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," 2015.

[21] K. Kim, K. Lee, D. Gowda, J. Park, S. Kim, S. Jin, Y. Y. Lee, J. Yeo, D. Kim, S. Jung, J. Lee, M. Han, and C. Kim, "Attention based on-device streaming speech recognition with large speech corpus," in *Proc. ASRU*, 2019.

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

[23] G. Pundak, T. N. Sainath, R. Prabhavalkar, A. Kannan, and D. Zhao, "Deep context: end-to-end contextual speech recognition," in *2018 IEEE Spoken Language Technology Workshop (SLT)*.

[24] M. Mohri, F. Pereira, and M. Riley, "Weighted finite-state transducers in speech recognition," *Computer Speech & Language*, vol. 16, no. 1.

[25] Ç. Gülçehre, O. Firat, K. Xu, K. Cho, L. Barrault, H. Lin, F. Bougares, H. Schwenk, and Y. Bengio, "On using monolingual corpora in neural machine translation," *CoRR*, vol. abs/1503.03535, 2015.

[26] 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.

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

[28] H. Pande, "Effective search space reduction for spell correction using character neural embeddings," in *Proc. EACL*, 2017.

[29] R. Grundkiewicz and M. Junczys-Dowmunt, "Minimally-augmented grammatical error correction," in *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT)*, 2019.

[30] A. Bruguier, R. Prabhavalkar, G. Pundak, and T. N. Sainath, "Phoebe: Pronunciation-aware contextualization for end-to-end speech recognition," in *Proc. ICASSP*. IEEE, 2019.

[31] P. Norvig, "How to write a spelling corrector," *Online at: http://norvig. com/spell-correct. html*, 2007.

[32] J. Jun and L. Lei, "ASR post-processing correction based on NER and pronunciation primitive," in *2011 7th International Conference on Natural Language Processing and Knowledge Engineering*. IEEE, 2011.

[33] J. Guo, T. N. Sainath, and R. J. Weiss, "A spelling correction model for end-to-end speech recognition," in *Proc. ICASSP*. IEEE, 2019, pp. 5651–5655.

[34] C. Kim, S. Kim, K. Kim, M. Kumar, J. Kim, K. Lee, C. Han, A. Garg, E. Kim, M. Shin, S. Singh, L. Heck, and D. Gowda, "End-to-end training of a large vocabulary end-to-end speech recognition system," in *Proc. ASRU* , 2019.

[35] Y.-B. Kim, D. Kim, A. Kumar, and R. Sarikaya, "Efficient large-scale neural domain classification with personalized attention," in *Proc. ACL (Volume 1: Long Papers)*, 2018, pp. 2214–2224.

[36] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," in *Proc. NAACL HLT*, 2016.

[37] L. Philips, "The double metaphone search algorithm," *C/C++ users journal*, vol. 18, no. 6, pp. 38–43, 2000.

[38] R. Sennrich, B. Haddow, and A. Birch, "Neural machine translation of rare words with subword units," *arXiv preprint arXiv:1508.07909*, 2015.

[39] D. Lee, P. Kapoor, and B. Kim, "Deeptwist: Learning model compression via occasional weight distortion," *CoRR*, vol. abs/1810.12823, 2018.

[40] C.-C. Chiu and C. Raffel, "Monotonic chunkwise attention," *arXiv preprint arXiv:1712.05382*, 2017.

[41] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," in *Soviet physics doklady*, vol. 10, no. 8.