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

SmallER: Scaling Neural Entity Resolution for Edge Devices

Ross McGowan, Jinru Su, Vince DiCocco, Thejaswi Muniyappa, Grant P. Strimel

Alexa Machine Learning, Amazon.com, USA
{rosmcgow,jinru,dicocv,thejasw,gsstrime}@amazon.com

Abstract

In this paper we introduce SmallER, a scalable neural entity resolution system capable of running directly on edge devices. SmallER addresses constraints imposed by the on-device setting such as bounded memory consumption for both model and catalog storage, limited compute resources, and related latency challenges introduced by those restrictions. Our model includes distinct modules to learn syntactic and semantic information and is trained to handle multiple domains within one compact architecture. We use compressed tries to reduce the space required to store catalogs and a novel implementation of spatial partitioning trees to strike a balance between reducing runtime latency and preserving recall relative to full catalog search. Our final model consumes only 3MB of memory at inference time with classification accuracy surpassing that of previously established, domain-specific baseline models on live customer utterances. For the largest catalogs we consider (300 or more entries), our proxy metric for runtime latency is reduced by more than 90%.

Index Terms: entity resolution, edge machine learning, entity normalization, similarity learning, similarity search

1. Introduction

Entity resolution (ER) is the process of finding entity mentions that refer to the same entity, and then inferring a canonical form from a set of match candidates [1]. ER is an important component of the spoken language understanding (SLU) model pipeline for digital voice assistants (VAs) such as Amazon Alexa, Google Assistant, and Apple Siri. The goal of ER in this context is to match entities mentioned within a user's utterances to canonical forms from predefined catalogs. This allows customers to speak more naturally to VAs without requiring them to precisely phrase canonical catalog values when making requests.

Entities in the context of VAs can be any set of objects that downstream software applications can use or interact with, such as smart home appliances, radio stations, songs, or contact names. Information about entities is stored in the form of a *catalog*. For each entity, the catalog includes its canonical name, identifier, and potentially some set of synonyms, i.e. alternative names for the same entity. Each type of entity has its own separate catalog. A *mention* is the part of a spoken utterance, as transcribed by the Automatic Speech Recognition (ASR) model, that has been tagged by a Natural Language Understanding (NLU) model as one of the designated entity types. For example, in the utterance,"turn on kitchen lights," the words *kitchen lights* are labeled as *smart home appliances*, forming the mention.

Modern production ER systems and solutions to related tasks such as *entity normalization* [2], *record linkage* [3], and *deduplication* [4] utilize the expansive compute resources of cloud infrastructure. In particular, they have the ability to leverage parallel compute functionality over powerful processors, employ large volumes of runtime memory, build on database support, and exploit caching services to deliver effective solutions with responsive query speed for latency sensitive applications such as real-time VAs. With further VA adoption, however, there has been a recent trend to enable SLU technology on the local device [5] where the aforementioned technologies no longer apply. Edge-deployed SLU allows users to interact with their VA when there is no internet connectivity or when serving the request locally improves the user experience. Supporting VA inference locally often requires the underlying SLU models to be redesigned and optimized for resource-constrained devices [6, 7, 8, 9]. This introduces several challenges for supporting ER on the device:

- Memory and CPU MIPS available to on-device ER are a small fraction of what's available for a cloud-based ER system.
- Large catalogs supporting ER can also consume a significant footprint. Contact names from ones phone, points-of-interest for navigation, and song catalogs all can easily be over ten thousand – and sometimes even one hundred thousand entities – which can quickly pose resource problems.
- Similarly, large catalogs can also significantly challenge runtime latency when working with limited local compute. Search solutions which scale well across parallel resources in the cloud will reduce to scale linearly in the number of catalog entries on edge devices. Even classic ER approaches to reduce the search space, such as blocking [10, 11], are challenging to deploy locally or need to be further adapted for the edge environment. To complicate matters, because ER occurs after the user speaks (i.e. after ASR and NLU) yet is required before a downstream action can be initiated, the ER execution time has a direct, significant impact on response latency and ultimately the user experience.

SmallER addresses all of these constraints with classification accuracy matching, and even surpassing in some cases, that of previously established cloud and edge ER models. Our contributions include 1) a small-footprint neural architecture capable of learning syntactic and semantic information simultaneously, 2) the application of compact tries to reduce the space required to represent catalogs on device, and 3) a novel implementation of spatial partitioning trees which at inference time strikes a balance between reducing runtime latency (by reducing the search space) and preserving recall relative to exhaustive catalog search.

http://dx.doi.org/10.21437/Interspeech.2021-98

2. Approach

SmallER uses a *contrastive learning* approach [12, 13] to conduct its modeling. Our neural architecture is trained to encode mentions and entities as high-dimensional vectors where these representations account for relative similarities between pairs of elements across a geometric measure [14, 15, 16]. We follow a Siamese training method, which has proven effective for a variety of applications [17, 18, 19], by executing the same encoder for both the mention from the utterance and the match candidates from the catalog. We train the model applying the triplet loss function [20, 21], where every candidate mention is presented with a matching and non-matching example, and implement semi-hard online batch mining [22]. The resulting objective minimizes the L_2 distance between the matching mention-entity pairs and maximizes the distance between the non-matching pairs simultaneously. By designing the model in this manner, at runtime we are able conduct inference directly over the catalog entity embeddings rather than their string representations. We perform a nearest neighbor search over entity embeddings using each mention within an utterance as a query. This procedure affords SmallER the ability to leverage precomputed, auxiliary data structures, as described in Section 3, for efficient search and storage.

SmallER's neural model architecture is capable of learning the following types of information:

Semantic: For catalogs like smart home appliances, where there is often shared semantic meaning between dissimilar string pairs (e.g., "lamp" and "light", "living room" and "den"), semantic information serves to connect these entries and ultimately improve classification accuracy. Often, even relatively common words give the model adequate examples of this type to learn from.

Syntactic: For catalogs like contact names, it is more common that string pairs differ primarily according to edit distance (e.g., "Liz" and "Elizabeth", "John" and "John Doe"). Where there is greater variation between string pairs, we have found that inclusion of semantic features is less likely to improve classification accuracy. Thus, encoding information about the syntax is imperative as well.

Because the respective benefit of these types of information varies by domain, SmallER encodes syntactic and semantic information separately within our model. This differs from previously proposed model architectures, where either one type of information is encoded [23, 24] or both types are fed to the same module simultaneously [25].

We encode syntactic information with a CNN module that receives a matrix representation of strings as proposed in [23]. We use a 30 x 100 matrix where rows correspond to twenty-six letter characters plus period, apostrophe, space, and a special unknown character; and the columns correspond to one-hot vector representations of the character sequence. We pass this matrix along to a convolutional layer with 32 filters of size 30 x 3, and then a fully connected layer with 512 output neurons. This type of architecture has proven successful at capturing syntactic information relevant to downstream measures like the Levenshtein distance.

We use a bidirectional LSTM module to encode semantic information as proposed in [24, 26]. In it, we have one Bi-LSTM layer followed by a fully connected layer with 256 output neurons. An important consideration for on-device applications is to keep the size of the word embeddings matrix in the semantic module as small as possible. We accomplish this simply by encoding a low number of dimensions in our matrix (30). We randomly initialize embeddings during training for the 10,000 most common tokens in our dataset, and use the `<unk>` token as a stand-in for all out-of-vocabulary tokens.

The outputs of the CNN and LSTM modules are then concatenated and passed through a fully connected layer that returns a 512-dimensional vector representation of the entity. As a final step, we apply L_2 normalization to the output vector. Our final model has approximately three million parameters across all components.

A prerequisite for deploying our model to edge devices is quantization of weights. SmallER uses quantization aware training (QAT) [27], specifically 8-bit QAT with absolute-cosine regularization [28], as this approach has proven effective for other on-device SLU tasks. We apply QAT to all model layers because in addition to reducing the footprint, with accelerated hardware support, it replaces expensive floating point operations at inference [29].

3. Compact search structures

SmallER employs two classes of data structures for its compression and search: compact tries and spatial partitioning trees. Applying these structures is critical to maintaining a small footprint on device while simultaneously providing accelerated inference speed for real time settings.

3.1. Compact tries

SmallER uses a trie implementation to encode its catalogs. A trie (or prefix tree) is a tree-like data structure which implements a dictionary mapping, typically string keys to a value type. In a standard trie, each node of the tree stores a character and has a prefix implicitly associated with it defined by the node's unique path from the root. Hence, for a particular node, a simple walk up the tree is used to determine its prefix. In essence a trie is just a specialized tree. SmallER leverages the Level Order Unary Degree Sequence (LOUDS) [30] to minimize the memory footprint consumed by the tree topology trie. Instead of linking nodes using explicit pointers, which are memory expensive, LOUDS encodes the tree structure using a succinct representation requiring just two bits per node. LOUDS accomplishes this by viewing each node as having an implicit integer label assigned to it by a level order tree traversal. A bit array storing the degrees of each node encoded in unary (e.g. degree of 4 is encoded as `11110`) is constructed to store the tree structure. Using only rank and select queries[1] over this bit array allows one to fully navigate the tree operating on the integer node labels. This representation gives a minimally compact bijection between catalog string entries and unique IDs[2]. This efficient mapping provides a succinct representation which our search structure in Section 3.2 exploits for further footprint reduction.

3.2. Spatial partitioning trees

In order to accelerate inference on large catalogs, SmallER uses spatial partitioning trees to reduce the search space. Namely, the trees provide SmallER with an approximate nearest neighbor (ANN) scheme that curtails the number of pointwise distance comparisons required to find a "good" entity match quickly.

[1] Rank returns the number of 1s preceding an index in a bit array. Select is the inverse of this operation. Both can be implemented in $O(1)$ time with minimal additional overhead.

[2] We note one can additionally apply the MARISA optimization [31] which recursivelly applies LOUDS for an even more compact representation.

While a variety of ANN techniques have been proposed, we chose partition trees over contenders like locality sensitive hashing (LSH) [32, 33] because the quality of those methods is sensitive to several parameters that are highly data dependent and difficult to tune [34]. Meanwhile, SmallER catalogs can be of a variety of sizes and composition across different users. Not only can trees adapt to heterogeneous data sets with less tuning [35], but we also detail here a method to represent them succinctly for on-device applications. Furthermore, unlike other methods, added search robustness can be easily built in to partitioning trees without an increased storage by altering search configurations.

Many methods have been proposed for constructing trees that adapt to the intrinsic dimensionality of high dimensional data for efficient ANN quality [36, 37]. SmallER's implementation is based on randomized partition trees [38], which use random projections; PCA trees [39], which split nodes along axes of greatest variance; and virtual spill trees [40], which avoid hard thresholding during search. SmallER uses a combination of the three, applying the most attractive properties of each to minimize memory footprint and enhance query speed for our application. Our tree is constructed from the catalog embeddings produced by the model as follows:

1. Seed a random number generator with numbers 0 through 255 (i.e. 8-bit seeds). For each seed, generate a random projection vector from the unit sphere. Of these projection candidates, determine the projection which maximizes the variance of the data.
2. Find a threshold *T* which evenly divides the data along the projection.
3. Store this float and the winning seed at each node, then split the data according to this dividing hyperplane and recurse on each subdivision in the tree.
4. Continue the recursion until the working dataset of a node is a single point. At this leaf, store an integer reference to the entity given by the trie from Section 3.1.

This results in a tree where each internal node represents a dividing hyperplane to direct queries traversing down the tree. The tree needs only to store the 8-bit seed and single float (can also be quantized) at each internal node of the tree, simple integer references on the leaves, and 2-bits per node to encode the tree topology by again applying the LOUDS encoding. Entity embeddings can be either stored contiguously in an auxiliary array indexed by IDs or generated lazily by the model only when needed if further space savings are required.

By using tree-based ANN search, we know that there will be some reduction in recall relative to the baseline exhaustive nearest neighbor search on *all* entities in a catalog. The challenge then is to balance the trade-off between reducing the search space (which translates to lower runtime latency) and preserving recall. To that end, we additionally add a configurable value α to each node as in virtual spill trees to provide soft thresholding during search. If the projected query is less than $T + \alpha$ at a node, one searches left and if greater than $T - \alpha$, one searches right. Note that now it is possible to search *both* child subtrees of a node. The α value is typically chosen automatically during construction to capture the $10 - 30\%$ of projected points closest to T.

Additionally, we applied the following augmentations to the search process:

Pruning: We keep track of the distance to the closest neighbor found thus far. If the projected distance to T of the query at a given node is more than best distance, and on the wrong side, it can be pruned because no leaf node in that subtree would have a nearer neighbor. The practical effect is that recall is not compromised at all, with some amount of search space reduction.

Prioritization: We strategically order how we traverse the tree. Without it, one simply might traverse depth-first, but with it we maintain a priority queue of the most promising branches to search each step based on the projection distances at visited nodes. The further from the dividing hyperplane, the more promising the branch.

Budgeting: This caps the number of nodes we visit in a given query. The intuition is that searching large catalogs is inherently more difficult – even if we exhaustively search all entities in a large catalog, there is a higher probability of returning an incorrect result. Thus, there is a point of diminishing return at which the increased latency of searching through a large catalog is not worth the incremental increase in probability of returning a correct match.

Leaf Grouping: A common technique for search trees is to maintain multiple objects at the leaves instead of single entities. Here we do the same in effect by exhaustively searching all leaves past a specified depth. We define this threshold typically in terms of "levels from bottom". A level from the bottom of 3 gives a grouping of size $2^3 = 8$. For small catalogs, where we are less concerned about a drastic search space reduction, this ensures an exhaustive search below some catalog size.

4. Data

A key goal of SmallER is to train one model capable of performing ER classification tasks for multiple domains, as this greatly simplifies the on-device model architecture and ongoing training and maintenance efforts. To accomplish this objective we sampled hundreds of thousands of annotated, live utterances across seven domains: Books, Contact Names, Music Artists, Music Service, SmartHome, Song Names, and Terrestrial Radio. All utterances have been processed so that users are not identifiable ("de-identified"). The model was trained on one unified dataset across all domains, and then was tested on separate datasets for each domain. Each domain brings its own challenges and relies on syntactic and semantic information in different ways as described in Section 2. The SmartHome test set consists of utterances that are particularly challenging examples for a production cloud SmartHome ER model. The catalog for Music Service is a static catalog of third-party radio station names that reflects what is used for customers in production. The catalogs for Contact Names, Music Artists, Song Names, and Terrestrial Radio are augmented with varying numbers of entries for each utterance. These 'synthetic' catalogs are built by randomly sampling negative examples from positive match entities found across all utterances within a given domain's dataset. Synthesizing catalogs has proven effective for adding robustness to the task and adaptability for related production use cases.

5. Experimental results

In prior work, the authors trained logistic regression-based ER models to return a probability score that two strings were a match. These models serve as our baselines. For the baseline models, there are syntactic feature extraction routines, such as the pair-wise Levenshtein distance and Jaro-Winkler distance, and also the incorporation of semantic features through pre-trained word embeddings as proposed in [24].

5.1. Supporting multiple domains with one model

Table 1 shows that for all seven of our domain-specific test sets, SmallER's multi-domain neural model outperforms domain-specific baselines. We are encouraged by the fact that we see the largest gains in recall for domains with large catalogs, such as Books (+11.7% relative to baseline), Music Service (+41.1%) and SmartHome (+19.9%), which consists of challenging utterances for the production cloud model.

Table 1: *SmallER results compared to its logistic regression (LR) baselines. Results for the production candidate SmallER solution for domains where we extensively tested tree-based search as described in section 5.3 are in bold. All baseline recall numbers are greater than 0.6, with the majority being higher than 0.9.*

Domain	Model	QAT	Trees	Relative Recall@1
Books	Books LR	No	No	-
	SmallER	No	No	+11.7%
	SmallER	Yes	No	+10.5%
	SmallER	**Yes**	**Yes**	**+9.1%**
Contact Names (CN)	CN LR	No	No	-
	SmallER	No	No	+1.6%
	SmallER	Yes	No	-0.4%
Music Artists (MA)	MA LR	No	No	-
	SmallER	No	No	+1.9%
	SmallER	Yes	No	0.0%
Music Service (MS)	MS LR	No	No	-
	SmallER	No	No	+41.1%
	SmallER	Yes	No	+36.6%
	SmallER	**Yes**	**Yes**	**+32.4%**
SmartHome	SmartHome LR	No	No	-
	SmallER	No	No	+19.9%
	SmallER	Yes	No	+16.8%
	SmallER	**Yes**	**Yes**	**+14.9%**
Song Names (SN)	SN LR	No	No	-
	SmallER	No	No	+1.1%
	SmallER	Yes	No	-0.5%
Terrestrial Radio (TR)	TR LR	No	No	-
	SmallER	No	No	+11.7%
	SmallER	Yes	No	+9.5%

5.2. Results after quantization aware training

Table 1 shows the effect of QAT on each of our domain-specific testsets. Given SmallER's relatively low number of parameters, some reduction in recall is to be expected after QAT, but the QAT model still outperforms the baselines except for Contact Names and Music Artists, where the difference is not statistically significant. After QAT, the on-device disk space and runtime memory consumption for the SmallER model is approximately 3MB. Furthermore, catalog entries are compressed overall by a factor of $2.5\times$.

5.3. Effect of trees on recall and search space

For testing results with spatial partitioning trees, we focus on three domains: Books, Music Service, and SmartHome. Books and SmartHome catalogs home are highly customer-specific but vary in size, as customers tend to have more book listings than SmartHome appliances. Music Service includes a static catalog of 302 public radio stations. The focus on these three highlights how our tree search will perform across catalogs with different roles and compositions.

The implementation of the search trees reduces recall as expected, but for Books (+9.1%), Music Service (+32.4%), and SmartHome (+14.9%) the QAT model with tree-based search still improves upon the baselines from Table 1. Furthermore, we are most encouraged by the estimated reduction in runtime latency for our queries when using the trees. For each query, we count the number of branching nodes and leaf nodes with distance calculations as a proxy for runtime latency. Without tree-based search, the Euclidean distance needs to be calculated between the mention vector and all vectors from the catalog. We experimented with different combinations of values for the α, leaf grouping, and budgeting hyperparameters. We found consistent ideal settings across customers and catalogs were α values between 0.2-0.3, a leaf grouping threshold between 3-5, and a budgeting threshold of 100. For the largest Books catalogs, we see dramatic reductions in the median number of nodes with calculations across all queries (see Table 2). For Books catalogs with 200-299 entries, we see an 86.8% reduction in nodes with calculations. For Books catalogs with 300 or more entries, we see a 91.2% reduction. For Music Service, we report an impressive reduction of 94.7%.

Table 2: *Recall@1 for tree-based search across various catalog sizes for Books. While recall reduces by larger amounts with larger catalogs, the search space is reduced by over 90% relative to full catalog search. Recall@1 and median nodes with calculations are relative to our multi-domain neural model with full catalog search.*

Books Catalog Size	QAT	Trees	Relative Recall@1	Relative Median Nodes w/ Calculations
0-99	No	No	-	-
	No	Yes	-0.6%	-19.2%
	Yes	Yes	-1.7%	-19.2%
100-199	No	No	-	-
	No	Yes	-2.4%	-77.9%
	Yes	Yes	-4.1%	-77.9%
200-299	No	No	-	-
	No	Yes	-3.0%	-86.4%
	Yes	Yes	-5.1%	-86.8%
300+	No	No	-	-
	No	Yes	-3.4%	-91.2%
	Yes	Yes	-4.8%	-91.2%

6. Discussion

SmallER is an on-device ER solution that simultaneously 1) supports multiple domains with high classification accuracy, which allows for efficient training, deployment, and maintenance; 2) learns to encode semantic and syntactic information about entities in a manner that allows us to implement tree-based search methods that greatly reduce runtime latency for large catalogs; and 3) effectively compresses the size of customer catalogs to stay within limited resource constraints on the device. For future work, we intend to move beyond syntax and semantics to also include contextual signals such as geographic location into our model. This has intuitive value for domains such as navigation, where distance from points of interest with highly similar names is an important consideration.

7. Acknowledgements

The authors would like to thank Yasser Gonzalez, Bei Jia, Majid Laali, and Jeff King for their development of the data pipeline we used to create our datasets, and for many helpful discussions throughout all stages of this project.

8. References

[1] L. Getoor and A. Machanavajjhala, "Entity resolution: Theory, practice and open challenges," *Proceedings of the VLDB Endowment*, vol. 5, pp. 2018–2019, 08 2012.

[2] P. Christen, *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.*, ser. Data-Centric Systems and Applications. Springer, 2012.

[3] N. Koudas, S. Sarawagi, and D. Srivastava, "Record linkage: Similarity measures and algorithms," in *ACM SIGMOD International Conference on Management of Data*, 2006, p. 802–803.

[4] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on Knowledge and Data Engineering*, vol. 19, no. 1, pp. 1–16, 2007.

[5] M. Locklear, "Panasonic and Alexa onboard bring offline voice control to your car," 2018. [Online]. Available: https://www.engadget.com/2018/01/08/panasonic-alexa-onboard-offline-voice-control-vehicle/

[6] G. P. Strimel, K. M. Sathyendra, and S. Peshterliev, "Statistical model compression for small-footprint natural language understanding," *INTERSPEECH*, 2018.

[7] Y. He, T. N. Sainath, and R. P. et al., "Streaming end-to-end speech recognition for mobile devices," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2019, pp. 6381–6385.

[8] K. M. Sathyendra, S. Choudhary, and L. Nicolich-Henkin, "Extreme model compression for on-device natural language understanding," *The 28th International Conference on Computational Linguistics (COLING)*, 2020.

[9] J. Macoskey, G. Strimel, and A. Rastrow, "Bifocal neural ASR: Exploiting keyword spotting for inference optimization," in *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Jun. 2021.

[10] R. C. Steorts, S. L. Ventura, M. Sadinle, and S. E. Fienberg, "A comparison of blocking methods for record linkage," in *Privacy in Statistical Databases*, 2014, pp. 253–268.

[11] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas, "Comparative analysis of approximate blocking techniques for entity resolution," *Proceedings of the VLDB Endowment*, vol. 9, pp. 684–695, 2016.

[12] R. Hadsell, S. Chopra, and Y. LeCun, "Dimensionality reduction by learning an invariant mapping," in *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, 2006, pp. 1735–1742.

[13] G. Chechik, V. Sharma, U. Shalit, and S. Bengio, "Large scale online learning of image similarity through ranking," *The Journal of Machine Learning Research*, vol. 11, p. 1109–1135, Mar. 2010.

[14] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. Russell, "Distance metric learning, with application to clustering with side-information," in *International Conference on Neural Information Processing Systems*, 2002, p. 521–528.

[15] M. Schultz and T. Joachims, "Learning a distance metric from relative comparisons," in *International Conference on Neural Information Processing Systems*, 2003, p. 41–48.

[16] K. Q. Weinberger and L. K. Saul, "Distance metric learning for large margin nearest neighbor classification," *The Journal of Machine Learning Research*, vol. 10, pp. 207–244, 2009.

[17] G. Koch, R. Zemel, and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition," in *International Conference on Machine Learning*, 2015.

[18] P. Neculoiu, M. Versteegh, and M. Rotaru, "Learning text similarity with Siamese recurrent networks," in *Workshop on Representation Learning for NLP*, 2016, pp. 148–157.

[19] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf, "Deepface: Closing the gap to human-level performance in face verification," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1701–1708.

[20] E. Hoffer and N. Ailon, "Deep metric learning using triplet network," in *Similarity-Based Pattern Recognition*, 2015, pp. 84–92.

[21] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A unified embedding for face recognition and clustering," in *IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.

[22] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," 2017. [Online]. Available: https://arxiv.org/abs/1703.07737

[23] L. Gómez, M. Rusiñol, and D. Karatzas, "LSDE: Levenshtein space deep embedding for query-by-string word spotting," in *International Conference on Document Analysis and Recognition*, 2017, pp. 499–504.

[24] M. Ebraheem, S. Thirumuruganathan, S. Joty, M. Ouzzani, and N. Tang, "Distributed representations of tuples for entity resolution," *Proceedings of the VLDB Endowment*, vol. 11, no. 11, p. 1454–1467, 2018.

[25] S. Fakhraei, J. Mathew, and J. L. Ambite, "NSEEN: Neural semantic embedding for entity normalization," 2018. [Online]. Available: http://arxiv.org/abs/1811.07514

[26] E. Linhares Pontes, S. Huet, A. C. Linhares, and J.-M. Torres-Moreno, "Predicting the semantic textual similarity with Siamese CNN and LSTM," in *Actes de la Conférence TALN*, 2018, pp. 311–320.

[27] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko, "Quantization and training of neural networks for efficient integer-arithmetic-only inference," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 2704–2713.

[28] H. D. Nguyen, A. Alexandridis, and A. Mouchtaris, "Quantization aware training with absolute-cosine regularization for automatic speech recognition," *INTERSPEECH*, 2020.

[29] Y. Yang, A. Chen, X. Chen, J. Ji, Z. Chen, Y. Dai, and A. Others, "Deploy large-scale deep neural networks in resource constrained IoT devices with local quantization region," *arXiv*, 2018. [Online]. Available: http://arxiv.org/abs/1805.09473

[30] G. Jacobson, "Space-efficient static trees and graphs," *IEEE Annual Symposium on Foundations of Computer Science*, pp. 549–554, 1989.

[31] S. Yata, "Dictionary compression by nesting prefix/patricia tries," *Annual Meeting of the Association for Natural Language*, 2011.

[32] P. Indyk and R. Motwani, "Approximate nearest neighbors: towards removing the curse of dimensionality," *ACM Symposium on Theory of Computing*, pp. 604–613, 1998.

[33] M. Datar, P. Indyk, N. Immorlica, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," *Annual Symposium on Computational Geometry*, pp. 253–262, 2004.

[34] W. Dong, Z. Wang, W. Josephson, M. Charikar, and K. Li, "Modeling LSH for performance tuning," *International Conference on Information and Knowledge Management, Proceedings*, pp. 669–678, 2008.

[35] K. Sinha, "LSH vs randomized partition trees: Which one to use for nearest neighbor search?" Institute of Electrical and Electronics Engineers Inc., 2014, pp. 41–46.

[36] A. Beygelzimer, S. Kakade, and J. Langford, "Cover trees for nearest neighbor," *International Conference on Machine Learning*, vol. 148, pp. 97–104, 2006.

[37] S. S. Vempala and G. Tech, "Randomly-oriented k-d trees adapt to intrinsic dimension," *Foundations of Software Technology and Theoretical Computer Science (FSTTCS)*, pp. 48–57, 2012.

[38] S. Dasgupta and Y. Freund, "Random projection trees and low dimensional manifolds," in *ACM Symposium on Theory of Computing*. Association for Computing Machinery, 2008, pp. 537–546.

[39] J. McNames, "A fast nearest-neighbor algorithm based on a principal axis search tree," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 23, no. 9, pp. 964–976, 2001.

[40] S. Dasgupta and K. Sinha, "Randomized partition trees for exact nearest neighbor search," in *Annual Conference on Learning Theory*, vol. 30, 2013, pp. 317–337.