Search Optimization with Query Likelihood Boosting and Two-Level Approximate Search for Edge Devices

Jianwei Zhang
jianwei.zhang@asu.edu
Arizona State University
Phoenix, Arizona, USA

Helian Feng
hlfeng@amazon.com
Amazon Alexa AI
Boston, Massachusetts, USA

Xin He
xih@amazon.com
Amazon Alexa AI
Boston, Massachusetts, USA

Grant P. Strimel
gsstrime@amazon.com
Amazon Alexa Speech
Pittsburgh, Pennsylvania, USA

Farhad Ghassemi
gfarhad@amazon.com
Amazon Alexa AI
Pittsburgh, Pennsylvania, USA

Ali Kebarighotbi
alikeba@amazon.com
Amazon Alexa AI
Boston, Massachusetts, USA

ABSTRACT

We present a novel search optimization solution for approximate nearest neighbor (ANN) search on resource-constrained edge devices. Traditional ANN approaches fall short in meeting the specific demands of real-world scenarios, e.g., skewed query likelihood distribution and search on large-scale indices with a low latency and small footprint. To address these limitations, we introduce two key components: a Query Likelihood Boosted Tree (QLBT) to optimize average search latency for frequently used small datasets, and a two-level approximate search algorithm to enable efficient retrieval with large datasets on edge devices. We perform thorough evaluation on simulated and real data and demonstrate QLBT can significantly reduce latency by 15% on real data and our two-level search algorithm successfully achieve deployable accuracy and latency on a 10 million dataset for edge devices. In addition, we provide a comprehensive protocol for configuring and optimizing on-device search algorithm through extensive empirical studies.

KEYWORDS

search engine indexing, approximate nearest neighbor, edge device

1 INTRODUCTION

As low-power edge devices like speakers and watches become increasingly integral to daily life, the need for locally executing traditional server-side tasks rises. Entity Resolution (ER), a key component for digital voice assistants (VAs) like Amazon Alexa and Apple Siri, is one of such tasks. ER maps entity mentions (i.e., query) to IDs/titles presented in a single or multiple datasets [9]. Approximate nearest neighbor (ANN) search algorithms are used for ER, which can identify and return the datapoints within a dataset that are the nearest to a query data point. Currently, most production ER applications rely on cloud resources with much larger memories and powerful processors than local devices to support advanced ANN methods. The increasing prevalence of VAs calls for local execution to improve latency and robustness against network connectivity and better address daily personalized consumer needs and data privacy [17]. Therefore, resolving the local ER challenge, constrained by limited computational resources, becomes crucial for ensuring a swift and consistent customer experience.

McGowan et al. proposed SmallER [19] to address several challenges of deploying ER on devices: limited CPU MIPS and run-time memory, capped on-device storage space for the index and search solutions, and strict query latency requirements. They utilize a compact encoded spatial partitioning projection tree (SPPT) for ANN search, which enables on-device ANN search for small to medium datasets. The SPPT is a balanced binary tree to provide equal latency (constant depth) for all queries. However, the query likelihood distribution of real-world traffic is usually highly-skewed, i.e., fathead vs. long tail [16], with a small number of frequently queried entities. Although SPPT is superior to other classic ANN methods for on-device search for small to medium datasets, the balanced structure leads to sub-optimal average latency. Furthermore, SPPT tends to have high latency on large datasets. Similarly, other classic ANN methods, such as locality sensitive hash and product quantization, cannot meet the accuracy or latency requirement either.

We present a novel solution for ANN search on resource-constrained edge devices. We addressed the limitations for traditional ANN algorithm in real world scenario with two key components: (1) a Query Likelihood Boosted Tree (QLBT) to optimize average search latency for frequently used small datasets; and (2) a two-level approximate search to enable efficient retrieval with large datasets on edge devices. We perform thorough evaluation on simulated and real data and demonstrate QLBT can significantly reduce latency by 15% on real data and our two-level search successfully achieve deployable accuracy and latency on a 10M dataset for edge devices. In addition, we provide a comprehensive protocol for configuring on-device ANN search through extensive empirical studies.

2 RELATED WORK

We briefly review classic ANN methods commonly used for ER. To improve search speed and reduce memory usage, these algorithms mildly relax accuracy constraints [7, 10]. Some graphic-methods, like hierarchical navigable small world [18] and others [20, 24], are excluded from our scope due to the on-device storage limitation.

Spatial Partitioning Tree Tree-based search structures for ANN have many variations including kd-tree [3], R-tree [12], random projection tree [6], etc. The core idea of these implementations is to partition the data into recursively divided regions represented in a tree structure. Similar neighbors are assumed to reside under a closer hierarchy of the resulting tree. While tree-based ANN performs well when the dataset is small, often when the dataset is large, extensive probing of many neighboring branches becomes unavoidable [23], leading to untenable search time.

Locality Sensitive Hash (LSH) LSH uses hash functions to allow similar keys or vectors stay in the same bucket with a higher likelihood. LSH is attractive because of its simplicity and fast search speed and simple random projection hash functions [5] is often effective. However, hyperparameters tuning for LSH is known to be notoriously difficult [4]. Thus, empirically tree-based methods have achieved generally better recall than LHS for ANN tasks on standardized datasets [22].

Product Quantization (PQ) PQ effectively encodes high dimensional vectors by an optimized sub-vector space codebook for storage compression and guide fast ANN search [8]. The distance between two vectors can be approximated by the distance between their codewords, which is faster than Euclidean distance computing. IndexIVFPQ method in FAISS is a SOTA ANN algorithm which uses PQ for sub-datasets searching [14]. However, PQ can be computational intensive on large dataset [23].

These existing ANN methods all fall short to meet the specific demands of real-world Edge device scenarios, specifically, the skewed query likelihood distribution and search on large dataset under low latency and footprint. In Section 3, we design a ER system to bridge the aforementioned gaps with a QLBT, a flexible two-level search and a comprehensive on-device search optimization protocol.

3 METHODOLOGY

3.1 Query likelihood boosted tree

VA ER systems' traffic usually has a heavily skewed frequency distribution with frequent queries for a few entities and rare visits for the rest, forming a long tail. The traditional ANN implementations (excluding augmentations like caching mechanisms) are designed to have approximately the same latency for each entity without factoring in query likelihood. We exploit the skewed frequency distribution to our advantage, adjusting for rare entities, to enhance average latency and optimize the experience for head traffic.

Inspired by the source coding techniques [11], we aim to minimize the anticipated depth of an entity in the projection tree to reduce the average latency during a query search:

$$\arg\min_{\theta} E[\text{Depth}(X;\theta)] = \arg\min_{\theta} \sum_{i} p(x_i)\text{Depth}(x_i;\theta),$$

where $x_i \in X$ is an entity, θ is the projection tree building rule, $Depth(x_i;\theta)$ is the depth of x_i in a projection tree built under rule θ, and $p(x_i)$ is the query likelihood for x_i.

We integrate the Shannon-Fano coding [15] with a randomized SPPT [19] to develop the QLBT. Our primary innovation lies in our splitting criteria where the splitting hyperplane is determined based on an equal probability of visiting the left and right branches. When the number of entities in the left and right branches are unbalanced (entities number in one branch is much smaller than another branch), the entities with higher query likelihood will be positioned closer to the root node, i.e. shallower depth and consequently reduced search time. Beside the novel splitting criteria, we also factor in the variability of the data projected along the corresponding projection to determine the best projection hyperplane.

To hedge the risk that unbalanced trees may have large latency on tail traffic (rare entities) and allow effective exhaustive searches on bottom leaves, we apply additional regulations on the QLBT building procedure for robustness: **(1) early stopping on query likelihood boosting** - once the partitioning tree building reach depth ℓ (we use $\ell = 3$), stop balancing on query likelihood (i.e., roll back to the balanced tree); **(2) pre-grouping leaves** - we pre-group leaves during building, i.e., maintain multiple objects at the leaves instead of a single entity. In our implementation, 8 entities are grouped into one leaf for fast retrieval. Experiments show pre-grouping leaves reduce the latency without affecting recall [19].

The recursive building procedure of QLBT is shown in Algorithm 1. We denote the entity embeddings in dataset as $\mathbf{e_1}, \mathbf{e_2}...\mathbf{e_N}$ where each $\mathbf{e_i}$ is a d-dimension vector, and entity query likelihood in real traffic as $p_1, p_2, ...p_N$. λ is a hyper-parameter to control the trade-off between query likelihood unbalance level and data separation which we performed grid search for optimization.

Algorithm 1 Query likelihood boosted tree building procedure

Require: Entity embedding $\mathbf{e_1}...\mathbf{e_m}$ of current node, current node depth D, early-stop boosting level ℓ, trade-off parameter λ

if $m \leq$ max leaf size **then**
  Save inference IDs of entities, return
else
  Generate K random projection $\mathbf{v_1}, ..., \mathbf{v_K}$ on unit sphere
  for Each projection vector $\mathbf{v_i}$ **do**
    Calculate projections: $\alpha_{i,j} = \mathbf{v_i} \cdot \mathbf{e_j}$
    Find $\tau^* = \arg\min_\tau |\sum_{\alpha_{i,j} \leq \tau} p(x_j) - \sum_{\alpha_{i,j} > \tau} p(x_j)|$
    $N_{\text{left}} = |\{\alpha_{i,j} \leq \tau^*\}|, N_{\text{right}} = |\{\alpha_{i,j} > \tau^*\}|$
    Unbalance score: $b_i = \max(N_{\text{left}}/N_{\text{right}}, N_{\text{right}}/N_{\text{left}})$
    Projection variations: $\sigma_i^2 = Var(\alpha_{i,1}, \ldots, \alpha_{i,m})$
    if $D \leq \ell$ **then**
      Projection score of $\mathbf{v_i}$: $score_i = \lambda \cdot \sigma_i^2 + (1-\lambda) \cdot b_i$
    else
      Projection score of $\mathbf{v_i}$: $score_i = \sigma_i^2$
    end if
  end for
  Best projection vector $\mathbf{v_{best}} \leftarrow \max_{\mathbf{v_i}} score_i$
  Split the entities of current node into left and right child-nodes with $\mathbf{v_{best}}$ and corresponding τ^*
  Repeat algorithm on left and right child-nodes
end if

Though a different construction, we use the same searching procedure described in [19] for QLBT. We emphasize that the boosted search tree must be updated for changes in users' query likelihood distribution. If only this distribution changes, a new search tree can be easily built, keeping other configurations the same. This method simplifies updates and enables personalized search experiences.

3.2 Two-level approximate search

Traditional ANN approaches such as projection trees, LSH, and PQ often fail to search large-scale datasets on edge devices with required low latency and small footprint. One effective strategy to improve search efficiency is pre-partitioning the entire dataset into several smaller subsets. These rules can be derived from the entity embedding distribution or other natural heuristics such as metadata, geolocation. After partitioning, appropriate search algorithms can

be applied to each subset individually. Pre-partitioning is also a common strategy employed to search large datasets in SOTA ANN algorithms, like IndexIVFPQ in FAISS [14].

We propose a two-level approximate search for edge ER, which is shown in Figure 2(a): (1) derive dataset pre-partitioning rules from features, e.g., vectorized entity embeddings, geolocation; (2) run K-means clustering on the partitioning features to create sub-datasets with corresponding centroids; (3) index the top-level on these centroids and complete ANN search based on entitiy embeddings over entities on the bottom-level. Our proposed method stems from the integration of top and bottom-level search mechanisms, enhancing efficiency over traditional methods. Utilizing feature-driven partitioning and K-means clustering, it can manage larger indices with lower latency, making it ideal for edge devices.

We offer a collection of choices for top- and bottom-level algorithms. We consider three algorithms for the top-level: (1) brute search, used when sub-datasets number is small; (2) kd-tree [3], used when the pre-partitioning features have low dimensions (e.g., latitude and longitude geolocation); (3) PQ [13], used when the pre-partitioning features have large dimensions (e.g. embedding). For the bottom-level we consider: (1) brute search, (2) QLBT, and (3) footprint-reduced LSH via a fixed set of random projections.

4 EXPERIMENT SETUP

To fairly compare different ANN search methods' performance on private and public datasets, we implement an ANN benchmark on an AWS t3.xlarge instance (3.1 GHz Intel Xeon processor, 16GB memory) based on an open-source ANN benchmark [1]. It provides a standardized way to evaluate different ANN algorithms. By setting a common ground for comparison, it allows us to understand how different algorithms perform under various scenarios, datasets, and parameters. All results are generated by this benchmark. Two searching limits are set to reflect the customer requirement for Edge ER: (1) P90 time less than 80ms with Python implementation; (2) recall@10 more than 80%, where recall@k is the percent of test dataset with the ground truth entity among top k returned entities.

4.1 Datasets

To comprehensively evaluate our on-device ANN search framework, we conduct experiments on a series of private and public ANN datasets with dataset sizes ranging from 10K to 10M. We use:

Radio Station: private, test sets sampled from real traffic, contains around 10K entities of 256d vectors.

SIFT: public, contains 1M entities of 128d vectors [13].

DEEP1B: public, contains 10M entities of 96d vectors [2].

4.2 Query likelihood simulation

To broadly assess the latency reduction with the proposed QLBT on various query likelihood distribution, we simulate different query likelihood distribution for 256 entities from Radio Station dataset via a Beta distribution. Each simulated dataset contains 10K queries sampled from Radio Station dataset. We use a query likelihood unbalancing score (based on information entropy [21]) to describe the likelihood unbalancing level of these entities: $1-\frac{-\sum_i^N p(x_i)\log_2 p(x_i)}{\log_2 N}$, where the N is the number of entities. High unbalance score denotes high skew level of the query likelihood distribution and a uniform distribution has a zero unbalance score. Our real world radio station query traffic has an unbalance score of 0.23.

5 RESULTS

5.1 Query likelihood boosted tree search

Using the simulation dataset described above, Figure 1 plots the relative search latency gain of QLBT as the unbalance score increases. We measure the P90 latency to achieve 0.95 recall@10 and benchmark against 0.95 recall@10 P90 latency of naive projection tree as baseline. Recall@k is defined as the portion of test dataset which has their ground truths among the top k returned entities. The results demonstrate that latency gain from QLBT increases with the unbalance level of the candidate query likelihood distribution.

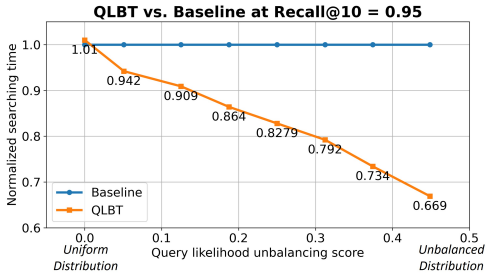

Figure 1: P90 time (recall@10 = 0.95) changes vs. query likelihood unbalancing score on QLBT and baseline tree.

We further validate our finding on real-world Radio Station traffic data which has an unbalancing score of 0.23. Applying QLBT, we achieve 16% latency reduction on P90 search time compared to naive projection tree at 0.95 Recall10 and 16% reduction on the average search time. This result aligns well with the results on simulated data (Figure 1), where an approximate 15% latency reduction by QLBT are observed on data of 0.23 unbalance score.

5.2 Two-level approximate search

We evaluate two-level search on two large scale retrieval datasets with more than 1M entities (SWIFT and DEEP1B) and produce the performance curve of recall10 vs 1/P90 latency of various search configurations. We record the best recall achieved on the given P90 constraint. Two searching limits are set as described in Section 4. In the performance curve plots (Figure 2 (b), (c), (d)), the algorithm appear on the most top-right has the best performance and only the curves fall in the green area meet the two searching limits.

Figure 2(b) shows the SIFT data experimental results. The one-level methods (tree and LSH) performance are shown in blue lines and two-level search algorithm with various configurations are in green lines. We also provide the recall@10 at 80ms P90 search time in Table 1). Neither of the one-level methods are able to achieve satisfactory recall at required latency. Whereas various two-level search configuration reach the acceptable performance range, with the best recall@10 at 0.980 from the top PQ + bottom Brute search configuration with 2^{15} sub-dataset split. The experiment result prove the necessity of a two-level search algorithm for large dataset.

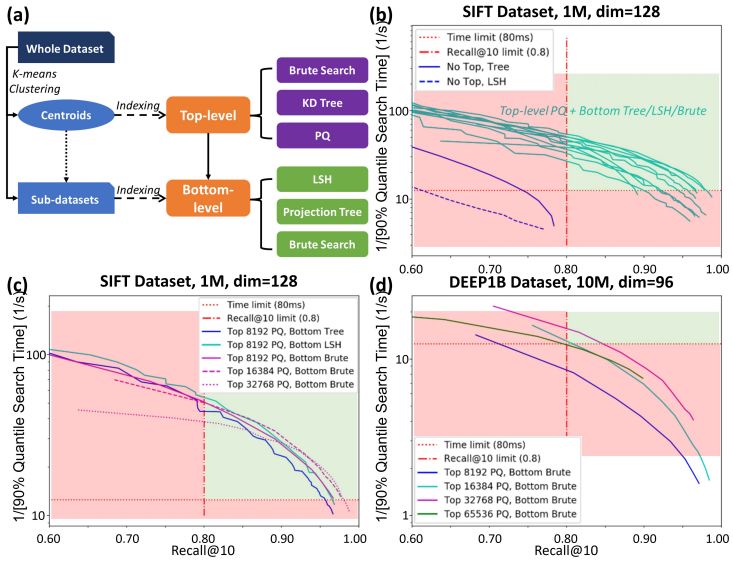

Figure 2: (a) Diagram for two-level search. (b) Pure tree and LSH vs. top PQ with 2^{10} to 2^{13} sub-datasets, bottom Tree/LSH/Brute. (c) Top PQ with 2^{13} and 2^{14} sub-datasets + bottom Tree/LSH/Brute on SIFT. (d) Top PQ + bottom Brute on DEEP1B.

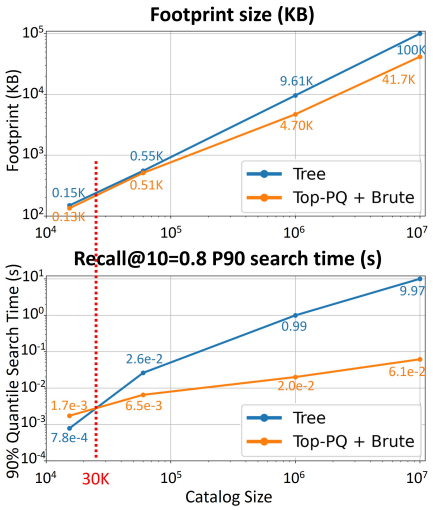

Figure 3: Footprint and P90 time comparison between one-level and two-level structure for catalogs with different sizes.

Table 1: Recall@10 when P90 time = 80ms for experiments results on SIFT.

Config	No Top /Tree	No Top /LSH	PQ-2^{10} /Tree	PQ-2^{10} /LSH	PQ-2^{11} /Tree	PQ-2^{11} /LSH	PQ-2^{12} /Tree	PQ-2^{12} /LSH	PQ-2^{13} /Tree	PQ-2^{13} /LSH	PQ-2^{13} /Brute	PQ-2^{14} /Brute	PQ-2^{15} /Brute
Recall	0.743 ×	0.613 ×	0.899	0.922	0.919	0.905	0.923	0.944	0.957	0.966	0.968	0.979	0.980✓

From Figure 2(b) and Table 1, we observe that the performance increases with the number of sub-datasets and brute force as the best performing bottom method. We increase the sub-datasets number to 2^{13} and 2^{14} and the result is shown in Figure 2(c). When the the average entities number within each subset is around 100, the two-level search achieves the optimal, and we notice again that brute search on the bottom-level outperforms ANN search methods (e.g., tree and LSH). In conclusion, two-level ANN search achieves optimal performance with PQ as the top-level search algorithm and brute search as the bottom-level and the average entities number in subset is around 100. The conclusion is also validated on the bigger 10M DEEP1B datased shown in Figure 2(d).

5.3 Edge ER Configuration Optimization

To provide guidance on optimal algorithm to generalize in more scenarios, we run experiments with different dataset sizes. The results, shown in Figure 3, reveal that footprint for one-level tree search and two-level search are similar for dataset size below 100K and two-level search has supreme P90 compared to one-level tree search when dataset size beyond 30K. Thus, we provide the guideline for on-device ANN search algorithm selection as below:

Dataset size is smaller than 30K:

(1) Traffic distribution available ⇒ likelihood boosted tree

(2) Traffic distribution not available ⇒ standard projection tree

Dataset size is larger than 30K:

(1) Top-level partitioning feature is high dimension (e.g., embedding) ⇒ two-level approximate search with PQ + brute search and average subsets entities number is around 100

(2) Top level partitioning feature is low dimension (e.g., geo-location) ⇒ two-level approximate search with top kd-tree and decide the bottom algorithm:

(a) If average subsets entities number ≤ than 100 ⇒ brute

(b) If average subsets entities number > 100 ⇒ tree

6 CONCLUSION

We propose two optimized ANN search algorithms for Edge ER, the query likelihood boosted tree and two-level approximate search. The QLBT reduces the average real world query latency by 16%. The two-level approximate search enables on-device ANN search with a 10M dataset, which is unachievable by previous methods. We also provide protocols for the best configuration of on-device search algorithm. For future work, we will implement the proposed algorithms on real edge devices and intend to include personalization into the on-device search.

ACKNOWLEDGMENTS

We thank Zhimin Peng for providing Local Search data and Zimeng (Chris) Qiu for providing Audible data and Jiyang Wang, Nicholas Dronen for helpful discussions on the topics covered.

REFERENCES

[1] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.

[2] Artem Babenko and Victor Lempitsky. 2016. Efficient indexing of billion-scale datasets of deep descriptors. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2055–2063.

[3] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.

[4] Deng Cai. 2019. A revisit of hashing algorithms for approximate nearest neighbor search. *IEEE Transactions on Knowledge and Data Engineering* 33, 6 (2019), 2337–2348.

[5] Moses S Charikar. 2002. Similarity estimation techniques from rounding algorithms. In *Proceedings of the thiry-fourth annual ACM symposium on Theory of computing*. 380–388.

[6] Sanjoy Dasgupta and Yoav Freund. 2008. Random projection trees and low dimensional manifolds. In *Proceedings of the fortieth annual ACM symposium on Theory of computing*. 537–546.

[7] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq Joty, Mourad Ouzzani, and Nan Tang. 2018. Distributed representations of tuples for entity resolution. *Proceedings of the VLDB Endowment* 11, 11 (2018), 1454–1467.

[8] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2013. Optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2946–2953.

[9] Lise Getoor and Ashwin Machanavajjhala. 2012. Entity resolution: theory, practice & open challenges. *Proceedings of the VLDB Endowment* 5, 12 (2012), 2018–2019.

[10] Daniel Gillick, Sayali Kulkarni, Larry Lansing, Alessandro Presta, Jason Baldridge, Eugene Ie, and Diego Garcia-Olano. 2019. Learning dense representations for entity retrieval. *arXiv preprint arXiv:1909.10506* (2019).

[11] Robert M Gray. 2011. *Entropy and information theory*. Springer Science & Business Media.

[12] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD international conference on Management of data*. 47–57.

[13] Herve Jegou, Matthijs Douze, and Cordelia Schmid. 2010. Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence* 33, 1 (2010), 117–128.

[14] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in one billion vectors: re-rank with source coding. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 861–864.

[15] Gareth A Jones and J Mary Jones. 2012. *Information and coding theory*. Springer Science & Business Media.

[16] Wouter Thomas Kritzinger and Melius Weideman. 2013. Search engine optimization and pay-per-click marketing strategies. *Journal of Organizational Computing and Electronic Commerce* 23, 3 (2013), 273–286.

[17] Mallory Locklear. 2018. *Panasonic and Alexa onboard bring offline voice control to your car*. https://www.engadget.com/2018-01-08-panasonic-alexa-onboard-offline-voice-control-vehicle.html

[18] Yu A Malkov and Dmitry A Yashunin. 2018. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence* 42, 4 (2018), 824–836.

[19] Ross Mcgowan, Jinru Su, Vince DiCocco, Thejaswi Muniyappa, Grant P Strimel, and Alexa Machine Learning. 2021. SmallER: Scaling Neural Entity Resolution for Edge Devices.. In *Interspeech*. 761–765.

[20] Yun Peng, Byron Choi, Tsz Nam Chan, and Jianliang Xu. 2022. Lan: Learning-based approximate k-nearest neighbor search in graph databases. In *2022 IEEE 38th international conference on data engineering (ICDE)*. IEEE, 2508–2521.

[21] Claude Elwood Shannon. 2001. A mathematical theory of communication. *ACM SIGMOBILE mobile computing and communications review* 5, 1 (2001), 3–55.

[22] Kaushik Sinha. 2014. LSH vs randomized partition trees: Which one to use for nearest neighbor search?. In *2014 13th International Conference on Machine Learning and Applications*. IEEE, 41–46.

[23] Hui Wang, Yong Wang, and Wan-Lei Zhao. 2022. Graph-based Approximate NN Search: A Revisit. *arXiv preprint arXiv:2204.00824* (2022).

[24] Wan-Lei Zhao, Hui Wang, and Chong-Wah Ngo. 2021. Approximate k-NN graph construction: a generic online approach. *IEEE Transactions on Multimedia* 24 (2021), 1909–1921.