Recent Approaches and Trends in Approximate Nearest Neighbor Search, with Remarks on Benchmarking

Martin Aumüller[†], Matteo Ceccarello[*]
[†]IT University of Copenhagen, maau@itu.dk
*University of Padova, matteo.ceccarello@unipd.it

Abstract

Nearest neighbor search is a computational primitive whose efficiency is paramount to many applications. As such, the literature recently blossomed with many works focusing on improving its effectiveness in an approximate setting. In this overview paper, we review recent advances of the state of the art and discuss some trends. Given the practical relevance of the problem, new approaches need to be thoroughly benchmarked. We therefore review some recent benchmarking efforts and provide advice on the benchmarking pipeline.

1 Introduction

Nearest neighbor search is a key component in many computer science applications. For example, using CLIP embeddings [50] both images and text can be mapped to dense vectors in a vector space; retrieving images that match a text description then boils down to embedding the text into a query vector and searching for images whose vectors are closest to the query vector under some distance measure. Using nearest neighbor search, large language models (LLMs) can also be augmented with knowledge that was not present in the training data [36]. Unfortunately, exact nearest neighbor search in high-dimensional data—such as the dense vectors generated by deep learning-based embedding pipelines—is a notoriously difficult problem that usually requires scanning through the whole dataset. This excludes the possibility of scalable approaches for billion-scale datasets that are common in today's applications.

To enable scalable nearest neighbor search, the research focus turned to approximate nearest neighbor search (ANN). In empirical settings, this usually means that an implementation does not provide a guarantee on the quality of the returned vectors. Instead, the user provides—based on knowledge of the data distribution and the query workloads—parameters that are used to build and to search the index data structure, respectively. Given a query point and some search parameters, the index is used to generate a candidate set for the query. The closest vectors among these candidates are returned as the (presumably) nearest neighbors for a given query. The smaller the candidate set, the faster the search, but also the lower the result accuracy. Section 2 gives an overview over general approaches to approximate nearest neighbor search.

ANN-benchmarks [10] presents the state-of-the-art benchmark on million-scale approximate nearest neighbor search implementations. In the benchmarking run published in April 2023, more than 30 implementations were tested on a collection of datasets. Each implementation was run on a single thread. The results for a single dataset are depicted in Figure 1. On a high level, many implementations achieve a throughput of more than 1,000 queries per second at an average recall of at least 90%. In particular, many implementations still perform well in the setting of average recall at least 99%. As a baseline, a bruteforce solution using BLAS achieved a throughput of 16 queries per second. Most of the approaches that perform best implement a graph-based approach with the notable exception of Google's ScaNN [25] and Meta's FAISS [35] which both implement a clustering-based approach as their main data structure. Please see the project website for more details on the used implementations. We review benchmarking efforts and pitfalls in Section 3.

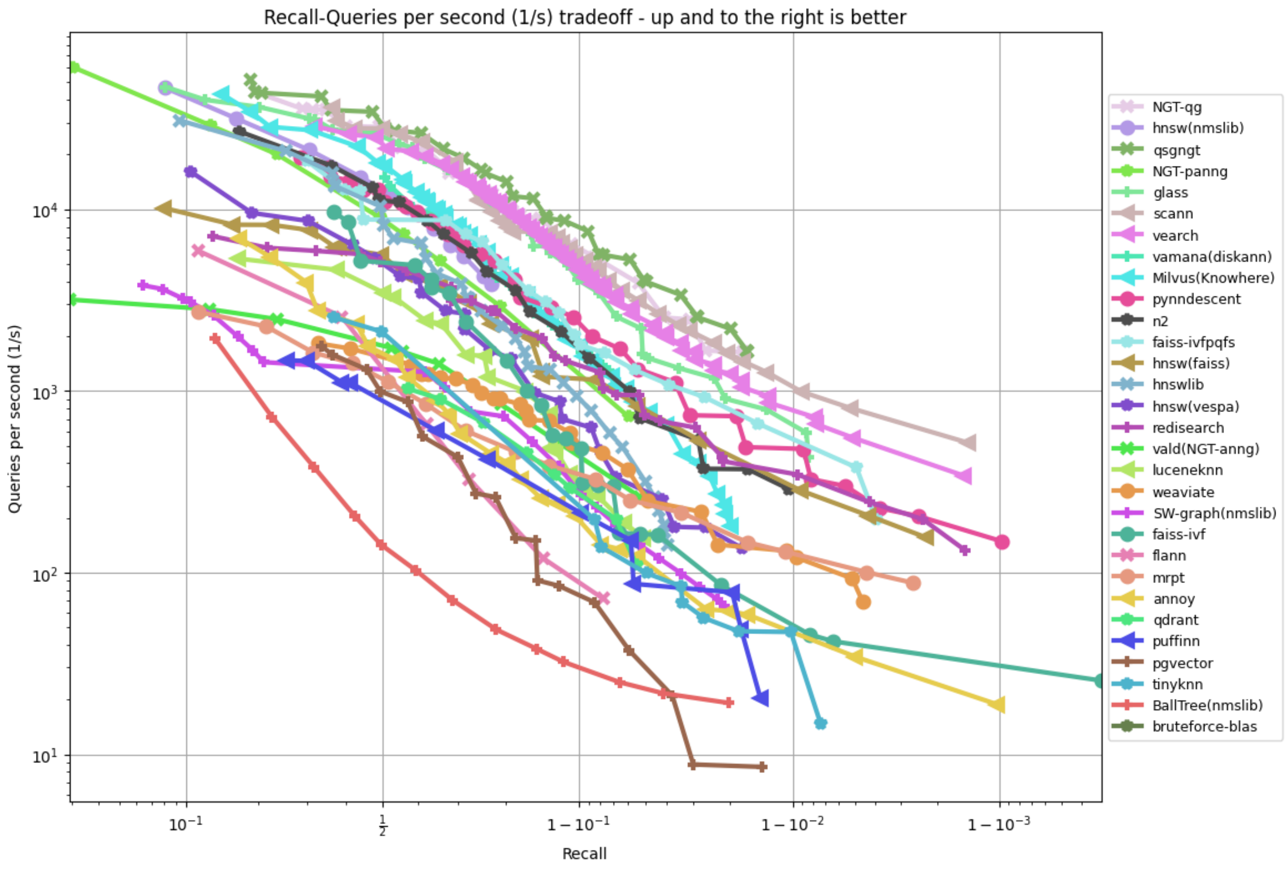

Figure 1: April 2023 run of ANN-benchmarks [10] (`http://ann-benchmarks.com`) on 100-dimensional vectors from GloVe embeddings [47] on a twitter corpus. The x-axis represents the average fraction of true 10-nearest neighbors returned (on a logarithmic scale); the y-axis provides the queries per second achieved, on a logarithmic scale. Both index building and searching is conducted on a single thread. Reported points represent the Pareto-frontier over a grid search on the parameter space. Interactive visualisations are available on the project website. Benchmarking run carried out by E. Bernhardsson.

This overview paper focuses on approaches that are not covered by the current benchmarking efforts. While theoretical breakthroughs have been achieved over the last years, for example by Andoni et al. [5], our focus lies on approaches that are supported by efficient implementations targeted to solve nearest neighbor search on real-world datasets. Section 4 is dedicated to this recent work. Finally, we close the overview by identifying recent trends and promising directions for future work in Section 5.

1.1 Problem setup

Formally, the task in k-nearest neighbor search is defined as follows. Let $(\mathcal{X}, \text{dist})$ be a metric space, and let $k \geq 1$ be an integer. Given a dataset $S \subseteq \mathcal{X}$ of n data points $(p_1, \ldots, p_n) \in \mathcal{X}^n$, the task is to build an index data structure that supports the following queries: Given a query point $q \in \mathcal{X}$, return a sequence $\mathcal{I}_q = (i_1, \ldots, i_k)$ of unique indices of data points in S such that $p_{i_1}, \ldots, p_{i_k}$ minimize the distance to q.

For simplicity, we will focus on the case that $\mathcal{X} = \mathbb{R}^d$, i.e., we consider d-dimensional real-valued vectors. Classical distance metrics are L_p norms, in particular for $p = 2$ (Euclidean distance), and inner product dissimilarity, associated with the task commonly known as maximum inner product search. Other interesting cases are length-d bitstrings $\mathcal{X} = \{0, 1\}^d$ under Hamming distance and collections of sets $\mathcal{X} = \mathcal{P}(U)$ under Jaccard similarity over a finite universe U with $\mathcal{P}(U)$ being the power set of U.

In the case that distances are unique, we can identify by $\mathcal{I}_q^*$ the set of indices of the true k-nearest neighbors of a query point q. As a quality measure, we consider the recall $|\mathcal{I}_q \cap \mathcal{I}_q^*|/k$. We call a method exact if it guarantees a recall of 1, and we call it approximate otherwise. In the context of approximate methods, papers often report on the average recall over a set Q of queries. If distances are not unique, distance-based recall variants are available [10].

2 A general overview over high-dimensional indexing

There exists a plethora of different approaches for solving nearest neighbor search. The most successful approaches can be categorized into *clustering-based*, *graph-based*, *hashing-based*, and *tree-based* approaches. For the notable exception of graph-based approaches, nearest neighbor search is usually solved by partitioning the space $\mathbb{R}^d$ into M disjoint parts $R_1, \ldots, R_M$ such that $\bigcup_{1 \le i \le M} R_i = \mathbb{R}^d$.

2.1 Indexing techniques for high-dimensional data

We provide a short overview of approaches and highlight a well-established method from each category. Each implementation comes with certain parameter choices used during the indexing phase (building the ANN data structure) and the querying phase (searching for the approximate nearest neighbors). To ease the explanation and make an attempt of unifying the landscape, we provide explanations that focus on a single build parameter M and a single search parameter ℓ.

Clustering-based approaches (IVF [35]). Given a dataset $S \subseteq \mathbb{R}^d$ and two parameters M and ℓ, run a clustering algorithm such as M-means to find M centroids. By associating each point with its closest centroid, the space is partitioned into M parts. The data structure that stores the centroids and the associated lists is referred to as an inverted file index (IVF). To find nearest neighbors to a query $q \in \mathbb{R}^d$, inspect the points associated to the ℓ closest centroids of q. Since this itself is a nearest neighbor search task, for large M an index over the centroids is employed. Clustering-based approaches usually provide very compelling index size since each point is stored only once with its associated centroid. The build time of a clustering-based approach is dominated by clustering the data points, which is often done on a sample. The final assignment carries out $O(nM)$ distance computations to centroids if the assignment is exact; as before, this can be sped up by indexing the centroids.

Graph-based approaches (HNSW [43]). Given a dataset $S \subseteq \mathbb{R}^d$ and parameters M, ℓ, the goal is to build a graph $G = (V, E)$, where each point is represented by a vertex and edges exist between a point and a "diverse" set of at most M points. Let us assume that such a graph G is given. To find the nearest neighbors of a query point q, HNSW uses a hierarchy of graphs to find a good entry point into the bottom-layer graph that indexes all points. Given such a start point, carry out a greedy hill climbing. In each round, consider the currently closest point to the query not considered before. Inspect the neighborhood and compute the distances to the query point. After each round, trim the list of current closest points (inspected and non-inspected) to ℓ, which is usually called the beam width. Terminate if all ℓ points have been considered. (Note that this is not a bound on the number of distance computations, since considered points might be trimmed off.) To build the graph, order all the points and insert them one-by-one using the search algorithm¸ often with a smaller ℓ' than used for the queries. From the points inspected in this search, a pruned set of M points is chosen as neighbors of the inserted point. Additionally, pruning might be necessary for its neighbors if their degree bound M is not met. Graph-based indexes usually provide compelling index sizes when M is small (the number of edges can be as large as Mn). The index build time is usually rather high for graph-based approaches, since individual searches are carried out for each data point.

Hashing-based approaches (FALCONN [3]) FALCONN is an approach based on locality-sensitive hashing optimized for inner product similarity on unit length vectors. It uses crosspolytope LSH as its LSH function. A crosspolytope LSH function is described by a rotation matrix, which is chosen at random. The hash value of a point is the closest base vector in $\mathbb{R}^d$ when applying the random rotation to the point. Given a set of points S and two parameter M, ℓ, the data structure works as follows. Choose M random LSH functions mapping data points to hash values $\mathcal{R}$, where each function is the concatenation of two to three random crosspolytope LSH functions. Hash each data point M times with independent hash function choices, and store the point in M buckets, one per hash function. These buckets together with the collection of hash functions form the index. Given a query point and collection of M hash tables, hash the query point using the same hash functions and consider the data points that also reside in the bucket as candidates. Traditionally, LSH suffers from large indexes since independent repetitions provide the "best" buckets in the sense that among all buckets, it is most likely to find a close points in the bucket identified by the query hash code. However, if space is a concern, one can use a smaller M value and if less than ℓ points are found, check neighboring buckets using a multiprobing approach. The index build time of a hashing-based approach is usually dominated by hashing each data point M times. Fast evaluation tricks, such as applying the fast Hadamard transform [3] and pooling/tensoring approaches [18] can be employed to lower this cost.

Tree-based approaches (MRPT [30]). MRPT builds a collection of trees based on sparse random projections. Given a set of points S and two parameters M, ℓ, the data structure works as follows. First, a node in a tree is described by a hyperplane a that splits up a point set $S' \subseteq S$. At the root, the whole dataset is taken into consideration, and a leaf is created as soon as the number of points at a node is below a certain threshold. Instead of a single tree, M trees are created to boost the quality of the results. Given a query point and a collection of M trees, carry out root-to-leaf-traversals in each tree for the query. In MRPT, a voting search is carried out by considering a point in a leaf as a candidate if it appears in at least ν different trees. MRPT results in compelling index sizes for small M values since each level of the tree contains a single random hyperplane, only storing the split value in a node. The build time is dominated by finding the individual splits, which typically requires in each node and over all trees, to evaluate the projection value and select the median. [32] describe a method to automatically select hyper-parameters for this approach.

2.2 System Architecture

The standard system architecture for ANN search is depicted in Figure 2. Given a dataset $S \subseteq \mathcal{X}$ and a set of build parameters $\mathcal{P}_{\text{build}}$, an index is built following the approaches mentioned in the previous subsection. Both the build time and the index size is often crucial for the feasibility of an approach. Given a query point $q \in \mathcal{X}$ and a set of query parameters $\mathcal{P}_{\text{query}}$, the index is used to $\underline{\text{generate the candidate set}}$. This candidate set is usually $\underline{\text{refined}}$ using a $\underline{\text{quantization}}$ or $\underline{\text{sketching}}$ technique that stores small summaries of each data point. The goal of the refinement is to discard candidate points that are $\underline{\text{unlikely}}$ to be among the k nearest neighbors. In a final $\underline{\text{reranking step}}$, exact distance computations between the refined candidates and the query point are carried out, and the indices of the k points with smallest distance to the query are returned as the answer to the query. In the case that memory resources are sparse, for example when dataset vectors do not fit into main memory, the final re-ranking step might not be carried out, which usually results in a loss in precision.

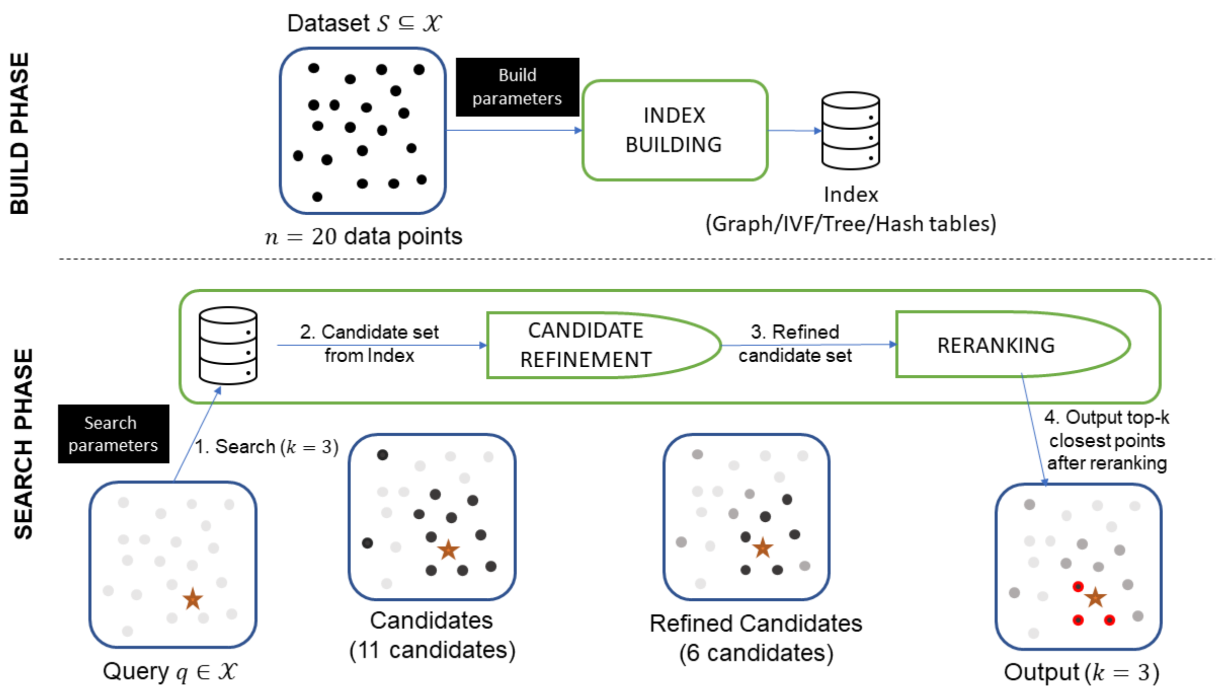
BUILD PHASE
Dataset $S \subseteq \mathcal{X}$
$n = 20$ data points
Build parameters
INDEX BUILDING
Index (Graph/IVF/Tree/Hash tables)
SEARCH PHASE
2. Candidate set from Index
CANDIDATE REFINEMENT
3. Refined candidate set
RERANKING
Search parameters
1. Search ($k = 3$)
4. Output top-k closest points after reranking
Query $q \in \mathcal{X}$
Candidates (11 candidates)
Refined Candidates (6 candidates)
Output ($k = 3$)

Figure 2: Overview of the phases of a traditional ANN implementation.

3 Benchmarking ANN implementations

Assessing the performance of a new method is at the same time crucial and challenging. The new method itself often has a lot of configurations to explore, and the number of baselines to compare with grows by the day, with each baseline featuring a lot of parameters. In such a scenario, benchmarking efforts such as the ANN-benchmarks [10] mentioned in the introduction— also used for benchmarking in billion-scale settings [53]— and the Lernaean Hydra framework for data series similarity [21] provide a very useful stepping stone.

In particular, a benchmarking infrastructure such as ANN-benchmarks provides a collection of baseline algorithms, along with sensible ranges of their parameters to test. Algorithms are then evaluated according to a standardized evaluation protocol: each approach is first set up with indexing parameters; then it is fed the data to be queried, allowing it to build index structures; finally several query groups are executed on the same index, with different query parameters.

We believe that it is much easier to integrate a new method in an existing benchmarking infrastructure, rather than implementing a custom one for each new paper. We therefore urge the community to adopt shared benchmarking infrastructures in order both to avoid reinventing the wheel and to make results more easily comparable across papers. Even if the core pipeline cannot be used, the data preprocessing (for example, the fixed definition of query workloads) and result postprocessing (for example, the availability of groundtruth data and evaluations scripts) can be used in isolation, ensuring reproducibility of results.

Observing the evolution of results of ANN-benchmarks throughout the last couple of years, and the experimental evaluations of the paper reviewed in this survey, we formulate in the following a few concrete suggestions on how to improve benchmarking of future works.

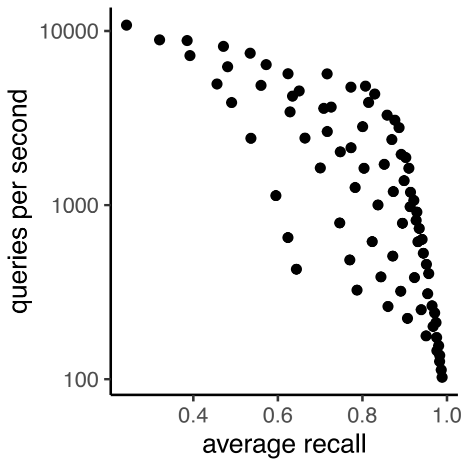

Figure 3: Recall/qps performance of several parameter combinations of HNSW on the Glove dataset.

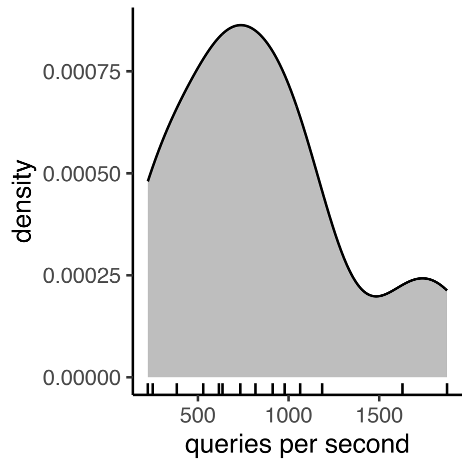

Figure 4: Distribution of queries per second of several parameter combinations of HNSW achieving at least recall 0.9.

3.1 Reporting on multiple metrics

Most of the times, the metrics considered for approximate nearest neighbor queries are the query throughput and the average recall. The query throughput—or equivalently the query time—are however very dependent on the implementation, and are thus measuring the performance of the implementation, rather than assessing the merits of the underlying algorithm [38].

While the actual performance of an implementation is what people are most interested in, considering other metrics can give further insights into the behavior of an algorithm. For instance, the number of distance computations performed can indicate how effective an approach is at reducing the amount of work to be performed. The interplay between the number of distance computations and the actual running time is also of interest: a linear scan through compressed points using product quantization may be faster than a more sophisticated index due to better use of the cache, despite carrying out more distance computations. As highlighted in [19], this is for example true for clustering-based approaches compared to graph-based approaches in million-scale settings. While the latter only carry out a fraction of distance computations, the more efficient memory layout of clustering-based approaches can make up for the additional calculations.

Furthermore, the index size and the index construction time are important metrics to complement the execution speed.

3.2 Selecting parameters

Many papers evaluate the proposed method by comparing with a few state of the art approaches, using a few configurations for each. Some papers use just a single configuration for the baseline, namely the default one provided by the implementation or the ones discussed in the associated publication. This approach can however lead to misleading comparisons, in that the performance of many approaches varies wildly in response to parameter changes, and differently across datasets.

For instance, HNSW is a commonly used baseline to compare with, and in many cases only a few parameter combinations are tested. Figure 3 reports the results—in terms of average recall and queries per second—of

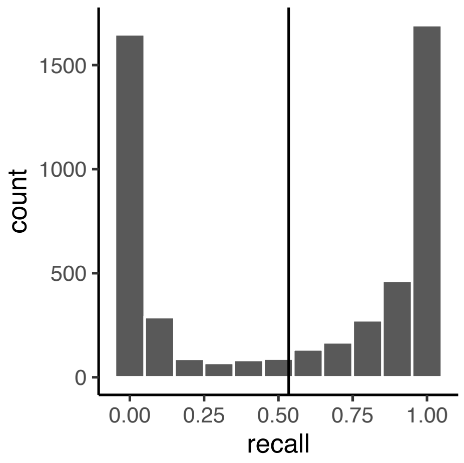

Figure 5: Histogram of the recall of single queries for a configuration of HNSW achieving average recall 0.55.

answering 5 000 queries on `glove-100-angular`[1] using HNSW in the implementation provided by the FAISS library [35]. The index building parameters being used are $M \in \{4, 8, 12, 16, 24, 36, 48, 64, 96\}$ and $efConstruction = 500$; the search parameters are varied in the range $ef \in \{10, 20, 40, 80, 120, 200, 400, 600, 800\}$. As one can observe, the outcomes vary wildly, both in terms of recall and in terms of query throughput. As such, using only a single parameter configuration as a baseline for comparison is very likely to result in suboptimal performance for the baseline itself. In this case we consider HNSW due to its popularity, but the same behavior can be observed with most approaches.

Even fixing a target recall and focusing on a single configuration achieving it does not make the performance more stable across parameter configurations. Figure 4 shows the distribution of the query throughput for all the aforementioned configurations that result in an average recall between 0.9 and 0.95. As we can see, the difference between the slowest and the fastest configurations is about two orders of magnitude.

3.3 Making workloads explicit

Many papers describe which datasets are part of their experimental section and then make a generic statement along the lines of n queries are run from the dataset. The reader is then left to conjecture that possibly queries are sampled at random from the dataset. However, it has been shown [8] that the practical performance of queries is greatly influenced by the intrinsic dimensionality of the queries themselves. Some works already address explicitly workloads with different difficulties [22, 68].

We therefore suggest to explicitly design query workloads that span a different range of difficulties, thus allowing to assess the performance of the approaches under test in finer detail. We provide Python code to compute intrinsic dimensionality measures of workloads at the following public repository: `https://github.com/Cecca/workloads-difficulty`.

3.4 Dangers of reporting on averages

In the previous section we considered the average recall of 5 000 queries as a performance indicator. This can be misleading at times, and hide interesting behavior.

[1]`http://ann-benchmarks.com/glove-100-angular.hdf5`

Consider again HNSW. Figure 5 reports the histogram of the recalls of individual queries of a run attaining average recall ≈ 0.55[2]. Strikingly, almost a third of the queries have recall 0, and a third of the queries have recall 1. In this case considering the average hides this bimodal behavior that may have important practical implications.

Therefore, we suggest to consider the distribution of the performance of individual queries, rather than drawing conclusions based on averages alone.

3.5 Implementation accessibility

The possibility to access the implementation backing the findings reported in any paper is of paramount importance for the community to verify and build upon results. Fortunately, in recent years the number of papers in nearest neighbor search that make their code accessible (usually as a Git repository hosted on online services such as GitHub or BitBucket) increased significantly.

Still, we note that code accessibility can be improved further. In some case, the code linked to a paper fails to compile following the instructions, most often due to differences between the environments of the code's author and the reader. Among the many solutions to this problem, we believe the most straightforward is to pair the code with container environments such as Docker [14] or Singularity [39]. Doing so also makes for an easier integration in existing benchmarking efforts, which often leverage containers in their infrastructure [10].

4 New approaches to ANN search

Having covered general approaches to high-dimensional indexing and remarks on benchmarking efforts, we will now focus on recent work. In the context of this overview paper, we report on approaches that appeared after the publication of the benchmarking paper [10].

4.1 Hashing-based Approaches

Recent works based on hashing have focused on extending classic LSH techniques by using new data structures, new query procedures, and by incorporating information from the data and query distribution.

PUFFINN [9] is an approach whose goal is to address approximate k-nearest neighbor queries while providing theoretical guarantees on the failure probability. In order to do so, it leverages the theoretical framework of Locality Sensitive Hashing (LSH) [15, 27]. While providing theoretical guarantees, LSH is known to have many parameters, whose setting is crucial to achieve good performance. To overcome this issue, PUFFINN adopts an adaptive approach based on the LSH forest trie data structure [13].

PM-LSH [66] is an approach focusing on L_p norms. Their key idea is as follows. A dimensionality reduction using the Johnson-Lindenstrauss transform is applied to project each point into a lower-dimensional space. The transformed points are then indexed by means of a PM-tree [55]. Queries are then carried out by performing several range queries on the PM-tree. While this approach is reminiscent of the earlier SRS approach [58], there is a somehow subtle difference. Where SRS runs k-NN queries in the projected space, which may suffer from the inaccuracy introduced by the projection (i.e. the second nearest neighbor in the projected space might not be the best candidate in the original space) PM-LSH runs a sequence of range queries, which they demonstrate to be more accurate.

FARGO [65] focuses on the Maximum Inner Product Search problem (MIPS). In order to apply LSH to the MIPS problem, the paper proposes an asymmetric transformation of data and queries so that all data points have the same norm, while retaining the original inner products with the queries. Then data points are indexed using LSH, and queried using a multi-probing approach.

[2]Using the `faiss` implementation of HNSW, with $efConstruction = 500$, $M = 16$ and $ef = 10$. Queries are the ones provided in the `glove-100-angular.hdf5` file from ANN-benchmarks.

CEO-MIPS [48] targets the MIPS problem as well, by performing several Gaussian random projections of the data and the queries. By leveraging the theory of concomitants of extreme order statistics, CEO-MIPS considers among all the projections of the query only the one with maximum value. If the i-th projection maximizes the absolute inner product, then it considers as candidates the data points whose i-th projection is large. The paper describes several variants of the approach that reduce the space usage.

FALCONN++ [49] improves on the Cross-Polytope-based hashing used in the FALCONN library [3]. The main insight is that mapping a data point into a bucket based on the random vector that maximizes the inner product (crosspolytope hashing), can also be used as an estimator of the distance between the point and the query vector, similarly to [48]. The paper proposes a threshold for the inner product to keep a point in a bucket, otherwise it is filtered away. This is the first practical implementation using the locality-sensitive filtering technique proposed by Andoni et al. [4] and by Christiani [17] in the context of approximate nearest neighbor search. We note that Rashtchian et al. [51] used this framework for computing similarity joins for skewed data.

LSH-CO-SUBSTRING [40] seeks to overcome one of the main hurdles of using LSH, that is selecting the number of repetitions. The key idea is that, instead of performing L repetitions of the LSH scheme, each vector is associated with a string of hash values of length m. Then, instead of defining buckets, a query looks for the hash strings with the longest colliding subsequence, allowing wraparounds at the string boundaries. The experimental analysis shows that the approach has an edge over other LSH-based approaches in terms of query time at a given recall, with markedly smaller index sizes.

LSH-APG [64] aspires to blend together LSH and graph-based approaches. In particular, LSH is used to speed up the construction of a graph-based index. At query time, LSH is again used to select a few entry points into the graph; these are then used to handle the search for the best query answers. Experiments show that LSH-APG has a larger index than graph-based methods, but such index can indeed be constructed much faster. Furthermore, the query performance at fixed parameters is shown to be better than other graph-based approaches.

DB-LSH [62] employs a dynamic bucketing scheme, modifying the classic LSH approach, to answer range queries. The traditional LSH approach for the Euclidean distance requires to project points onto a random direction, and then to bucket the projections at indexing time to build the hash codes. In DB-LSH such quantization is deferred to query time, so to be able to center the buckets around the query. In particular, at index time the random projections of the dataset points are indexed in an R*-tree. At query time, the R*-tree is used to answer a sequence of (rectangular) range searches, that are equivalent to dynamically bucketing the hash values around the query.

HD-INDEX [7] answers approximate k-nearest neighbor queries with the aim to use less space than LSH. The core idea is to partition the dataset with a regular grid, which is then traversed using a space-filling curve (such as Hilbert of Z-order). Points are then inserted in a tree-like data structure using their position along the space filling curve as keys. The rationale is that points that are close in a geometric sense are also close along the space filling curve. The experiments reported in the paper show that, compared to baselines the HD-INDEX answers queries with a better Mean Average Precision, in a shorter time.

4.2 Graph-based approaches

Graph-based approaches offer among the largest variety of known methods, see for example the survey paper by Wang et al. [63]. In general, recent works have focused on enabling the use of graph-based approaches on larger-than-memory data, on addressing the issue of indexing time, and on designing pruning/refinement strategies for the graph building process.

DISKANN [56] targets approximate nearest neighbor search in an external memory setting, where the size of the dataset makes it impossible to store the index and the data entirely in memory. Therefore, the main aim is to develop an index that minimizes the number of disk reads per query to amortize the disk latency. To this end, DISKANN refines a random k-regular graph using iterated beam searches from a central graph node, the medoid. It refines the graph in two steps with different pruning values. In difference to HNSW, no hierarchy is employed.

ELPIS [11] tackles one of the main issues of graph-based approaches, which is the index construction time. It does so by first partitioning the datasets using a tree-based data structure, and then by using HNSW to build graphs on the leaves. As such, the graph construction is carried out in parallel on smaller subsets of the data. This leads to a faster index construction and to a smaller index as well. In particular, ELPIS builds its tree by employing dimensionality reduction techniques borrowed from the data series community, observing that a high-dimensional vector can be considered as an instance of a data series.

Dobson et al. [19] give a detailed account on scaling graph-based nearest neighbor search implementations to billion-scale datasets. In particular, they describe parallelization techniques to deal with the potential data dependencies that can occur during the parallel insertion of points.

4.3 Clustering-based approaches

SCANN [25] extends the standard inverted file index based on (hierarchical) k-means clustering, designed for inner product spaces. Their motivation is that ℓ_2-based k-means clustering may favor centroids that do not preserve the ordering under inner product similarity. To mitigate this issue, they propose an anisotropic quantization technique which is more accurate for inner product similarity. A significant ingredient to SCANN's performance is a very efficient product quantization implementation based on SIMD instructions as described by Andre et al. [6]. Sun et al. [57] extend the approach with an efficient hyper-parameter selection technique.

4.4 Learning-based approaches

Algorithms with predictions [37] is a recent trend in the development of algorithms and data structures. The idea is that an oracle, for example a machine learning model, gives predictions for data that is stored in the data structure. An overview over progress in this field in general is given by Mitzenmacher and Vassilvitskii [45]. A thorough survey of deep learning-based methods for approximate nearest neighbor search is given in [42].

In the context of ANN search, two main research directions are (i) using a machine learning model to guide the candidate generation and (ii) using a machine learning model to set adaptive stopping criteria in a traditional data structure. In the former, the indexing part is augmented with a machine learning model; in the latter, the search phase is augmented using such a model.

ANN AS MULTILABEL CLASSIFICATION by Hyvönen et al. [31] proposes to formulate the following multi-label classification problem: Given $S \subseteq \mathcal{X}$ and $x \in \mathcal{X}$, let $y_i = [p_i \text{ is a } k\text{-NN of } x \text{ in } S]$. Thus, the (high-dimensional) label $y = (y_1, \ldots, y_n)$ represents the set of k nearest neighbors of x. Given these pairs $\{(x^{(i)}, y^{(i)})\}_{1 \leq i \leq n}$, we can train a classifier for this multi-label classification problem. Applied to space partitioning nearest neighbor search algorithms, such as trees, LSH, or clustering-based variants, the authors show that the pre-dominant approach that collects the points that fall into the same part of the partition (i.e., a leaf or a bucket) is not the natural classifier for the multi-label classification problem. Instead, one should use a majority vote based on groundtruth labels y of these points to decide on the candidates to check.

NEURALLSH [20] proposes to build the k-NN graph on the dataset S, and find a balanced partition of this graph into m disjoint parts. These m parts form the m buckets of the data structure. The label of a point x is an m-bit string, where the i-th bit is set if x has a k-NN in part i of the graph. The labeling is learned by means of a neural network, and the search is guided by predicting bucket probabilities using the neural network, and checking all buckets in sorted order, thresholding at a certain value.

BLISS [26] applies iterative repartitioning by learning the bucket assignment and redistributing points according to the learned assignments, in rounds. More precisely, data points are split up at random into B groups/buckets. The label of a data point x is a length-B bit string; label $i \in \{1, \ldots, B\}$ is set if the nearest neighbor of x is in group i. After learning the assignment, a prediction step is carried out for each data point, the top-K buckets with the highest probability are retrieved, and the data point is moved to the least loaded bucket among these top-K. The process is repeated R times independently. Experimentally, a small value of R is sufficient, and B is set to roughly $\sqrt{n}$.

LEARNING TO HASH, ROBUSTLY [2] proposes a learned LSH function for binary data under Hamming distance. In particular, rather than sampling bit-coordinates uniformly at random as in the standard LSH scheme, the paper describes a method to optimize the probability distribution over coordinates, for the given dataset. In contrast to the approaches mentioned above, they can guarantee worst-case running time comparable to the best known theoretical approaches, while being able to adapt to the difficulty of a query dynamically.

Li et al. [41] note that most approaches using indexes to reduce the number of candidates to evaluate do not adapt to the difficulty of a query, i.e., use the same stopping condition for all queries. The consequence is that to achieve a good recall a conservative stopping condition needs to be used, thus hurting the performance of easier queries. Therefore, they develop a prediction pipeline based on gradient boosting decision trees, allowing the implementation of an adaptive stopping condition. Such an adaptive stopping condition is then implemented in the IVF and HNSW indexes, with experiments showing a general reduction in latency.

Continuing along the line of the previous paper, Zheng et al. [67] focus on IVF indexes, with the aim of setting the number of cells to probe on a per query basis. To this end, they first modify the way data vectors are clustered, so to ensure that each cluster has a balanced number of entries. Then they use autoencoders, trained offline on sample queries, to estimate the number of cells to probe for a query.

4.5 Refining Candidates

In the context of nearest neighbor search, an important ingredient of the system pipeline is the refinement of a set of candidates. To this end, one wants to compress vectors in a way such that points likely to not be part of the k nearest neighbors are to be excluded without incurring an exact distance computation. A general overview of the techniques is given by Pagh in [46]. The main technique used is Product Quantization as introduced by Jegou et al. [34]. An overview over this technique and its variants is given by Matsui et al. [44].

FINGER [16] aims at improving the speed at which nearest neighbor graphs are traversed. The key observation is that during traversals most of the distances do not need to be computed exactly. Therefore, the paper proposes an estimation method to quickly estimate distances that can be used to improve the performance of any graph-based nearest neighbor algorithm. To showcase the performance of the approach, the paper integrates it in an HNSW implementation.

ADSAMPLING [23] aims at optimizing one of the most basic operations in nearest neighbor search: the distance comparison operation, which given a pair of points returns whether the points' distance is above or below a given threshold. First, the same random rotation is applied to each point. Then, given two vectors their distance is computed by considering the rotated dimensions one at a time, in a progressive sampling fashion, conducting a statistical hypothesis test on whether the two vectors are closer or farther than the threshold.

LVQ [1] describes a locally-adaptive vector quantization technique, which uses scalar quantization with individual lower and upper bounds on the coordinate values. Employed in a graph-based index, the authors show compelling performance up to billion-scale datasets.

5 Trends

5.1 Billion-Scale ANN search

Scaling ANN search to billion-scale datasets has been one of the core research directions of the past years. In particular, this is supported by the availability of diverse datasets through the NeurIPS 2021 Billion-Scale ANN challenge [53] and other large datasets such as Laion5B [52]. On this scale, index construction time and index size pose the most challenging part of the indexing pipeline. Dobson et al. [19] provide an empirical comparison of the scaling of different approaches from million to billion scale, focusing primarily on graph-based approaches.

They discuss different parallelization strategies to efficiently parallelize the index construction and the (batched) search. They conclude that graph-based nearest neighbor search performs best on billion-scale datasets in the high recall regime. In terms of index building times and index size, clustering- and hashing-based approaches were shown to be competitive.

5.2 New Tasks in ANN Search

In the following we describe extension of nearest neighbor search targeted towards real-world applications.

Filtered Search When data vectors are associated with some metadata, for example for the YFCC-100M [61] dataset or the LAION5B dataset [52], a natural extension to k-NN search is to incorporate the metadata into the query. Technically, each data vector is associated with a bag-of-words representation of its tags. The query vector additionally has tags $t_1, \ldots, t_T$, and the task is to find the (approximate) k-NN among all points in the dataset that contain the query's tags. A graph-based solution to this problem is described by Gollapudi et al. [24].

Out-of-Distribution Queries A common scenario in modern approximate nearest neighbor search is that data and query vectors originate from different distributions, but are embedded into the same space. For example, the Yandex TEXT2IMAGE dataset [53] consists of image embeddings produced by the Se-ResNext-101 model described by Hu et al. [28], while queries are textual embeddings produced by a variant of the DSSM model by Huang et al. [29]. As described in [53], the mapping to the shared 200-dimensional real valued vectors under inner product similarity is learned via minimizing a variant of the triplet loss using clickthrough data. A novel approach to explicitly deal with out-of-distribution queries is described in [33]. As pointed out in [19], cluster- and LSH-based approaches are particularly affected by these changes in distribution.

Streaming Search A particular challenge in the index building process is to maintain an index over dynamic insertions and deletions. Technically, a stream of add, delete, search operations is given, where each search is supposed to return the approximate k-nearest neighbors of all the elements that are in the index according to the add and delete operations. Simhadri [3] describes different applications of this setting. For example, for web search the index may contain roughly a trillion vectors and has to handle billions of updates per day. Searches have to be handled at a latency of at most 10 milliseconds with a throughput of 10,000-100,000 queries per second. Naïve solutions such as placing tombstones for deleted items quickly degrade performance, as would a complete rebuild of the index in a given interval size. Singh et al. [54] describe a graph-based approach to handle this issue and show competitive performance to the non-stream setting, and a big improvement over previous LSH-based approaches [59]. Their system handles thousands of add/delete operations per second while maintaining high recall/throughput comparable to the non-streaming setting. A special case of this dynamic setting is content drift, which was studied by Baranchuk et al. [12].

5.3 Conclusions

In recent years, the field of approximate nearest neighbor search has witnessed an exciting surge in the development of new approaches and the refinement of existing methods. Alongside the traditional k-nearest neighbor setting, new tasks are emerging: queries can incorporate different metadata such as tags, an index has to be kept over a dynamic stream of search, insert, and remove operations, or queries might arise from a different distribution than the data points. Moreover, the scale of today's datasets has reached billions of data points, requiring unprecedented scalability while retaining accuracy.

[3] `https://harsha-simhadri.org/pubs/ANNS-talk-Sep22.pptx`, (accessed on Sept 25, 2023.)

Given the practical relevance of the problem, it is crucial that new approaches are benchmarked thoroughly. Given the effort needed to set up a benchmarking infrastructure, we invite the research community to adopt and improve shared benchmarks which avoid the pitfalls that commonly affect experimental evaluations.

Acknowledgements: Martin Aumüller thanks Yusuke Matsui for insightful discussions. He also thanks his collaborators organizing and working on approximate nearest neighbor search challenges [53, 60], and the NeurIPS'23 challenge[4]. This research was supported by the Innovation Fund Denmark for the project DIREC (9142-00001B).

References

[1] Cecilia Aguerrebere, Ishwar Singh Bhati, Mark Hildebrand, Mariano Tepper, and Theodore Willke. Similarity search in the blink of an eye with compressed indices. Proc. VLDB Endow., 16(11):3433–3446, 2023. ISSN 2150-8097. doi: 10.14778/3611479.3611537. URL `https://doi.org/10.14778/3611479.3611537`.

[2] Alexandr Andoni and Daniel Beaglehole. Learning to hash robustly, guaranteed. In ICML, volume 162 of Proceedings of Machine Learning Research, pages 599–618. PMLR, 2022.

[3] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya P. Razenshteyn, and Ludwig Schmidt. Practical and optimal LSH for angular distance. In NIPS, pages 1225–1233, 2015.

[4] Alexandr Andoni, Thijs Laarhoven, Ilya P. Razenshteyn, and Erik Waingarten. Optimal hashing-based time-space trade-offs for approximate near neighbors. In SODA, pages 47–66. SIAM, 2017.

[5] Alexandr Andoni, Aleksandar Nikolov, Ilya P. Razenshteyn, and Erik Waingarten. Approximate nearest neighbors beyond space partitions. In SODA, pages 1171–1190. SIAM, 2021.

[6] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. Quicker ADC : Unlocking the hidden potential of product quantization with SIMD. IEEE Trans. Pattern Anal. Mach. Intell., 43(5):1666–1677, 2021.

[7] Akhil Arora, Sakshi Sinha, Piyush Kumar, and Arnab Bhattacharya. Hd-index: Pushing the scalability-accuracy boundary for approximate knn search in high-dimensional spaces. Proc. VLDB Endow., 11(8): 906–919, 2018.

[8] Martin Aumüller and Matteo Ceccarello. The role of local dimensionality measures in benchmarking nearest neighbor search. Inf. Syst., 101:101807, 2021.

[9] Martin Aumüller, Tobias Christiani, Rasmus Pagh, and Michael Vesterli. PUFFINN: parameterless and universally fast finding of nearest neighbors. In ESA, volume 144 of LIPIcs, pages 10:1–10:16. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019.

[10] Martin Aumüller, Erik Bernhardsson, and Alexander John Faithfull. Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. Inf. Syst., 87, 2020.

[11] Ilias Azizi, Karima Echihabi, and Themis Palpanas. Elpis: Graph-based similarity search for scalable data science. Proc. VLDB Endow., 16(6):1548–1559, 2023.

[4] `https://big-ann-benchmarks.com/`

[12] Dmitry Baranchuk, Matthijs Douze, Yash Upadhyay, and I. Zeki Yalniz. Dedrift: Robust similarity search under content drift. CoRR, abs/2308.02752, 2023.

[13] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In WWW, pages 651–660. ACM, 2005.

[14] Carl Boettiger. An introduction to docker for reproducible research. ACM SIGOPS Oper. Syst. Rev., 49(1): 71–79, 2015.

[15] Andrei Z. Broder. On the resemblance and containment of documents. In SEQUENCES, pages 21–29. IEEE, 1997.

[16] Patrick H. Chen, Wei-Cheng Chang, Jyun-Yu Jiang, Hsiang -Fu Yu, Inderjit S. Dhillon, and Cho-Jui Hsieh. FINGER: fast inference for graph-based approximate nearest neighbor search. In WWW, pages 3225–3235. ACM, 2023.

[17] Tobias Christiani. A framework for similarity search with space-time tradeoffs using locality-sensitive filtering. In SODA, pages 31–46. SIAM, 2017.

[18] Tobias Christiani. Fast locality-sensitive hashing frameworks for approximate near neighbor search. In SISAP, volume 11807 of Lecture Notes in Computer Science, pages 3–17. Springer, 2019.

[19] Magdalen Dobson, Zheqi Shen, Guy E. Blelloch, Laxman Dhulipala, Yan Gu, Harsha Vardhan Simhadri, and Yihan Sun. Scaling graph-based ANNS algorithms to billion-size datasets: A comparative analysis. CoRR, abs/2305.04359, 2023.

[20] Yihe Dong, Piotr Indyk, Ilya P. Razenshteyn, and Tal Wagner. Learning space partitions for nearest neighbor search. In ICLR. OpenReview.net, 2020.

[21] Karima Echihabi, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. Return of the lernaean hydra: Experimental evaluation of data series approximate similarity search. Proc. VLDB Endow., 13(3): 403–420, 2019. doi: 10.14778/3368289.3368303. URL `http://www.vldb.org/pvldb/vol13/p403-echihabi.pdf`.

[22] Karima Echihabi, Panagiota Fatourou, Kostas Zoumpatianos, Themis Palpanas, and Houda Benbrahim. Hercules against data series similarity search. Proc. VLDB Endow., 15(10):2005–2018, 2022.

[23] Jianyang Gao and Cheng Long. High-dimensional approximate nearest neighbor search: with reliable and efficient distance comparison operations. Proc. ACM Manag. Data, 1(2):137:1–137:27, 2023.

[24] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, Amit Singh, and Harsha Vardhan Simhadri. Filtered-DiskANN: Graph algorithms for approximate nearest neighbor search with filters. In WWW, pages 3406–3416. ACM, 2023.

[25] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. Accelerating large-scale inference with anisotropic vector quantization. In ICML, volume 119 of Proceedings of Machine Learning Research, pages 3887–3896. PMLR, 2020.

[26] Gaurav Gupta, Tharun Medini, Anshumali Shrivastava, and Alexander J. Smola. BLISS: A billion scale index using iterative re-partitioning. In KDD, pages 486–495. ACM, 2022.

[27] Sariel Har-Peled, Piotr Indyk, and Rajeev Motwani. Approximate nearest neighbor: Towards removing the curse of dimensionality. Theory Comput., 8(1):321–350, 2012.

[28] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. In CVPR, pages 7132–7141. Computer Vision Foundation / IEEE Computer Society, 2018.

[29] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry P. Heck. Learning deep structured semantic models for web search using clickthrough data. In CIKM, pages 2333–2338. ACM, 2013.

[30] Ville Hyvönen, Teemu Pitkänen, Sotiris K. Tasoulis, Elias Jaasaari, Risto Tuomainen, Liang Wang, Jukka Corander, and Teemu Roos. Fast nearest neighbor search through sparse random projections and voting. In IEEE BigData, pages 881–888. IEEE Computer Society, 2016.

[31] Ville Hyvönen, Elias Jääsaari, and Teemu Roos. A multilabel classification framework for approximate nearest neighbor search. In NeurIPS, 2022.

[32] Elias Jääsaari, Ville Hyvönen, and Teemu Roos. Efficient autotuning of hyperparameters in approximate nearest neighbor search. In PAKDD (2), volume 11440 of Lecture Notes in Computer Science, pages 590–602. Springer, 2019.

[33] Shikhar Jaiswal, Ravishankar Krishnaswamy, Ankit Garg, Harsha Vardhan Simhadri, and Sheshansh Agrawal. OOD-DiskANN: Efficient and scalable graph ANNS for out-of-distribution queries. CoRR, abs/2211.12850, 2022.

[34] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. Product quantization for nearest neighbor search. IEEE Trans. Pattern Anal. Mach. Intell., 33(1):117–128, 2011.

[35] Jeff Johnson, Matthijs Douze, and Hervé Jégou. Billion-scale similarity search with gpus. IEEE Trans. Big Data, 7(3):535–547, 2021.

[36] Urvashi Khandelwal, Omer Levy, Dan Jurafsky, Luke Zettlemoyer, and Mike Lewis. Generalization through memorization: Nearest neighbor language models. In ICLR. OpenReview.net, 2020.

[37] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In SIGMOD Conference, pages 489–504. ACM, 2018.

[38] Hans-Peter Kriegel, Erich Schubert, and Arthur Zimek. The (black) art of runtime evaluation: Are we comparing algorithms or implementations? Knowl. Inf. Syst., 52(2):341–378, 2017.

[39] Gregory M. Kurtzer, Vanessa Sochat, and Michael W. Bauer. Singularity: Scientific containers for mobility of compute. PLOS ONE, 12(5):1–20, 05 2017. doi: 10.1371/journal.pone.0177459. URL `https://doi.org/10.1371/journal.pone.0177459`.

[40] Yifan Lei, Qiang Huang, Mohan S. Kankanhalli, and Anthony K. H. Tung. Locality-sensitive hashing scheme based on longest circular co-substring. In SIGMOD Conference, pages 2589–2599. ACM, 2020.

[41] Conglong Li, Minjia Zhang, David G. Andersen, and Yuxiong He. Improving approximate nearest neighbor search through learned adaptive early termination. In SIGMOD Conference, pages 2539–2554. ACM, 2020.

[42] Mingjie Li, Yuan-Gen Wang, Peng Zhang, Hanpin Wang, Lisheng Fan, Enxia Li, and Wei Wang. Deep learning for approximate nearest neighbour search: A survey and future directions. IEEE Trans. Knowl. Data Eng., 35(9):8997–9018, 2023.

[43] Yury A. Malkov and Dmitry A. Yashunin. Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Trans. Pattern Anal. Mach. Intell., 42(4):824–836, 2020.

[44] Yusuke Matsui, Yusuke Uchida, Hervé Jégou, and Shin'ichi Satoh. A survey of product quantization. ITE Transactions on Media Technology and Applications, 6(1):2–10, 2018.

[45] Michael Mitzenmacher and Sergei Vassilvitskii. Algorithms with predictions. Commun. ACM, 65(7): 33–35, 2022.

[46] Rasmus Pagh. Similarity sketching. In Encyclopedia of Big Data Technologies. Springer, 2019.

[47] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In EMNLP, pages 1532–1543. ACL, 2014.

[48] Ninh Pham. Simple yet efficient algorithms for maximum inner product search via extreme order statistics. In KDD, pages 1339–1347. ACM, 2021.

[49] Ninh Pham and Tao Liu. Falconn++: A locality-sensitive filtering approach for approximate nearest neighbor search. In NeurIPS, 2022.

[50] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In ICML, volume 139 of Proceedings of Machine Learning Research, pages 8748–8763. PMLR, 2021.

[51] Cyrus Rashtchian, Aneesh Sharma, and David P. Woodruff. Lsf-join: Locality sensitive filtering for distributed all-pairs set similarity under skew. In WWW, pages 2998–3004. ACM / IW3C2, 2020.

[52] Christoph Schuhmann, Romain Beaumont, Richard Vencu, Cade Gordon, Ross Wightman, Mehdi Cherti, Theo Coombes, Aarush Katta, Clayton Mullis, Mitchell Wortsman, Patrick Schramowski, Srivatsa Kundurthy, Katherine Crowson, Ludwig Schmidt, Robert Kaczmarczyk, and Jenia Jitsev. LAION-5B: an open large-scale dataset for training next generation image-text models. In NeurIPS, 2022.

[53] Harsha Vardhan Simhadri, George Williams, Martin Aumü ller, Matthijs Douze, Artem Babenko, Dmitry Baranchuk, Qi Chen, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. Results of the neurips'21 challenge on billion-scale approximate nearest neighbor search. In NeurIPS (Competition and Demos), volume 176 of Proceedings of Machine Learning Research, pages 177–189. PMLR, 2021.

[54] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. FreshDiskANN: A fast and accurate graph-based ANN index for streaming similarity search. CoRR, abs/2105.09613, 2021.

[55] Tomás Skopal, Jaroslav Pokorný, and Václav Snásel. Nearest neighbours search using the pm-tree. In DASFAA, volume 3453 of Lecture Notes in Computer Science, pages 803–815. Springer, 2005.

[56] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, and Rohan Kadekodi. Rand-nsg: Fast accurate billion-point nearest neighbor search on a single node. In NeurIPS, pages 13748–13758, 2019.

[57] Philip Sun, Ruiqi Guo, and Sanjiv Kumar. Automating nearest neighbor search configuration with constrained optimization. In ICLR. OpenReview.net, 2023.

[58] Yifang Sun, Wei Wang, Jianbin Qin, Ying Zhang, and Xuemin Lin. SRS: solving c-approximate nearest neighbor queries in high dimensional euclidean space with a tiny index. Proc. VLDB Endow., 8(1):1–12, 2014.

[59] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. Proc. VLDB Endow., 6(14):1930–1941, 2013.

[60] Eric S. Tellez, Martin Aumüller, and Edgar Chavez. Overview of the SISAP 2023 indexing challenge. In SISAP, 2023.

[61] Bart Thomee, David A. Shamma, Gerald Friedland, Benjamin Elizalde, Karl Ni, Douglas Poland, Damian Borth, and Li-Jia Li. YFCC100M: the new data in multimedia research. Commun. ACM, 59(2):64–73, 2016.

[62] Yao Tian, Xi Zhao, and Xiaofang Zhou. DB-LSH: locality-sensitive hashing with query-based dynamic bucketing. In ICDE, pages 2250–2262. IEEE, 2022.

[63] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. Proc. VLDB Endow., 14(11):1964–1978, 2021.

[64] Xi Zhao, Yao Tian, Kai Huang, Bolong Zheng, and Xiaofang Zhou. Towards efficient index construction and approximate nearest neighbor search in high-dimensional spaces. Proc. VLDB Endow., 16(8):1979–1991, 2023.

[65] Xi Zhao, Bolong Zheng, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. FARGO: fast maximum inner product search via global multi-probing. Proc. VLDB Endow., 16(5): 1100–1112, 2023.

[66] Bolong Zheng, Xi Zhao, Lianggui Weng, Quoc Viet Hung Nguyen, Hang Liu, and Christian S. Jensen. PM-LSH: a fast and accurate in-memory framework for high-dimensional approximate NN and closest pair search. VLDB J., 31(6):1339–1363, 2022.

[67] Bolong Zheng, Ziyang Yue, Qi Hu, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S. Jensen. Learned probing cardinality estimation for high-dimensional approximate NN search. In ICDE, pages 3209–3221. IEEE, 2023.

[68] Kostas Zoumpatianos, Yin Lou, Ioana Ileana, Themis Palpanas, and Johannes Gehrke. Generating data series query workloads. VLDB J., 27(6):823–846, 2018.