

# Benchmarking Filtered Approximate Nearest Neighbor Search Algorithms on Transformer-based Embedding Vectors

Patrick Iff  
ETH Zurich, Switzerland  
iffp@inf.ethz.ch

Paul Brügger  
ETH Zurich, Switzerland  
pbruegger@student.ethz.ch

Marcin Chrapek  
ETH Zurich, Switzerland  
marcin.chrapek@inf.ethz.ch

Maciej Besta  
ETH Zurich, Switzerland  
maciej.best@inf.ethz.ch

Torsten Hoefler  
ETH Zurich, Switzerland  
htor@inf.ethz.ch

## ABSTRACT

Advances in embedding models for text, image, audio, and video drive progress across multiple domains, including retrieval-augmented generation, recommendation systems, vehicle/person re-identification, and face recognition. Many applications in these domains require an efficient method to retrieve items that are close to a given query in the embedding space while satisfying a filter condition based on the item’s attributes, a problem known as Filtered Approximate Nearest Neighbor Search (FANNS). In this work, we present a comprehensive survey and taxonomy of FANNS methods and analyze how they are benchmarked in the literature. By doing so, we identify a key challenge in the current FANNS landscape: the lack of diverse and realistic datasets, particularly ones derived from the latest transformer-based text embedding models. To address this, we introduce a novel dataset consisting of embedding vectors for the abstracts of over 2.7 million research articles from the arXiv repository, accompanied by 11 real-world attributes such as authors and categories. We benchmark a wide range of FANNS methods on our novel dataset and find that each method has distinct strengths and limitations; no single approach performs best across all scenarios. ACORN, for example, supports various filter types and performs reliably across dataset scales but is often outperformed by more specialized methods. SeRF shows excellent performance for range filtering on ordered attributes but cannot handle categorical attributes. Filtered-DiskANN and UNG excel on the medium-scale dataset but fail on the large-scale dataset, highlighting the challenge posed by transformer-based embeddings, which are often more than an order of magnitude larger than earlier embeddings. We conclude that no universally best method exists.

### Code:

<https://github.com/spcl/fanns-benchmark>

### Datasets:

<https://huggingface.co/datasets/SPCL/arxiv-for-fanns-small>  
<https://huggingface.co/datasets/SPCL/arxiv-for-fanns-medium>  
<https://huggingface.co/datasets/SPCL/arxiv-for-fanns-large>

## 1 INTRODUCTION

Advances in embedding models for text [78, 80, 148], image [89, 125], video [121], audio [13, 29], and other modalities [81] have significantly enhanced semantic search, where items are mapped

to a high-dimensional vector space, and similarity is measured by the distance between embedding vectors. Efficient similarity search algorithms are essential to navigate this space. Given the scale of modern datasets, exact nearest neighbor search (ENNS) algorithms are too slow, necessitating the use of approximate nearest neighbor search (ANNS) methods [14, 51, 54, 58, 68, 71, 72, 93, 94, 122].

Many real-world applications, including large language models (LLMs) with retrieval-augmented generation [50, 117, 140], recommendation systems [75, 123, 144], vehicle and person re-identification [88, 153], face and voice recognition [55, 115, 144], and e-commerce [82, 138], require retrieving only items that satisfy specific filtering conditions on item attributes, e.g., access rights for a document, a video’s timestamp, or a product’s price. These requirements have driven the development of filtered approximate nearest neighbor search (FANNS). Both academia and industry have recognized its growing importance, leading to numerous research publications [41, 53, 60, 85, 101, 109, 133, 143, 153, 154] and adoption in database solutions such as Pinecone [111], Vespa [129], and Vectara [128]. Evaluating FANNS methods requires vector datasets that reflect real-world workloads. Yet, despite the rapid rise of transformer-based text embeddings, no FANNS benchmarks with such vectors are known to us.

Our contributions are threefold. First, to clarify the requirements for effective FANNS benchmarks, we **survey** the current FANNS landscape and propose a comprehensive taxonomy of existing methods. We classify approaches along three key dimensions: *filtering approach*, *indexing technique*, and *supported filter types*. Second, we present a **novel dataset** featuring vectors generated by embedding paper abstracts from the arXiv dataset [28] using the transformer-based model `stella_en_400M_v5` [104, 148]. Each item in our dataset has 11 real-world attributes. Third, we **benchmark** a diverse set of FANNS methods on this dataset. Our evaluation reveals that each method involves distinct trade-offs, and no single approach dominates across all conditions. For example, ACORN supports multiple filter types and scales well, but is often outperformed by more specialized methods. Filtered-DiskANN and UNG perform strongly at medium scale but fail on our largest dataset. SeRF performs consistently well but supports only range filters on ordered attributes. These results demonstrate that FANNS remains a nuanced design space, and that selecting the right method depends on the specific workload and filtering requirements.

## 2 BACKGROUND

### 2.1 FANNS Problem Statement

In FANNS, we are given  $n$  **items**,  $m \geq 1$  **attributes**, and  $p$  **queries**. Each *item*  $I_i = (v_i, a_{i,1}, \dots, a_{i,m})$  for  $i \in \{1, \dots, n\}$  consists of a  $d$ -dimensional embedding vector  $v_i$  and a value for each of the  $m$  *attributes* ( $a_{i,1}$  to  $a_{i,m}$ ). An *item* can, e.g., represent a research paper with  $v_i$  being an embedding of its abstract and attributes corresponding to venue, year of publication, and authors. A *query*  $Q_j = (q_j, k_j, f_j)$  for  $j \in \{1, \dots, p\}$  consists of a  $d$ -dimensional query vector  $q_j$ , the number of *items* to return  $k_j$ , and a filter function  $f_j$  that determines whether an *item* matches the filter. For a *query*,  $q_j$  can, e.g., be the embedding of a search term with  $f_j$  only returning true for papers submitted to VLDB. We use  $o_j \leq m$  to denote the number of *attributes* the filter  $f_j$  depends on. Table 1 summarizes all parameters.

Table 1: (§2.1) Parameters used in FANNS.

Parameter	Description
$n$	Number of items
$m$	Number of attributes
$p$	Number of queries
$d$	Dimensionality of embedding and query vectors
$v_i$	Embedding vector of the $i$ -th item
$a_{i,l}$	$l$ -th attribute of the $i$ -th item
$q_j$	Query vector of the $j$ -th query
$k_j$	Number of items to return for the $j$ -th query
$f_j$	Filter expression of the $j$ -th query
$o_j$	Number of attributes considered in $f_j$

To answer a *query*  $Q_j$ , we approximately retrieve the  $k_j$  *items* that satisfy the filter  $f_j$  and have embedding vectors closest to the query vector  $q_j$  according to a given distance function (e.g., Euclidean distance, cosine similarity, Manhattan distance, or Hamming distance). Most FANNS algorithms return only *items* that match the filter; however, some approximate the filtering step, occasionally retrieving *items* that do not satisfy it. We refer to this relaxed version of the FANNS problem as the approximately filtered approximate nearest neighbor search (AFANNS) problem.

### 2.2 Example Use-Case of FANNS

To illustrate the practical use of FANNS, we introduce an example application depicted in Figure 1. We reference components and actions in Figure 1 using letters **A** and numbers **1**, respectively. Consider a semantic search engine for research papers; a system that retrieves the most relevant papers for a given *query*, even when different terminology is used to describe the same concept. To provide fine-grained control over search results, we incorporate filters for venue, publication year, and authors.

**Index Construction:** To build such a system, we transform each paper in our database **A** into an *item*, as described in Section 2.1. The paper’s venue, publication year, and authors define **1** the *item*’s *attributes*. The paper’s abstract is processed **2** by a text embedding model **B**, such as NV-Embed [78, 105], LENS [79, 80], Stella [104, 148], GTE [3, 84], or BGE [25, 106], to generate the *item*’s embedding vector. To efficiently answer FANNS queries for these *items*, we insert **3** them into a FANNS index **C**.

**Query execution:** When a user enters **4** a search query, we transform it into a FANNS *query*, as described in Section 2.1. The filter  $f_j$  and the number of requested results  $k_j$  are set **5** based on the user’s input. The query text is processed **6** by the same text embedding model **B** used during index construction to generate the query vector. The *query* is submitted **7** to a FANNS algorithm **E**, which utilizes **8** the FANNS index **C** to retrieve the IDs of the most relevant papers. These IDs are forwarded **9** to a retriever **D**, which fetches **10** the corresponding papers from the database **A** and returns **11** them to the user.

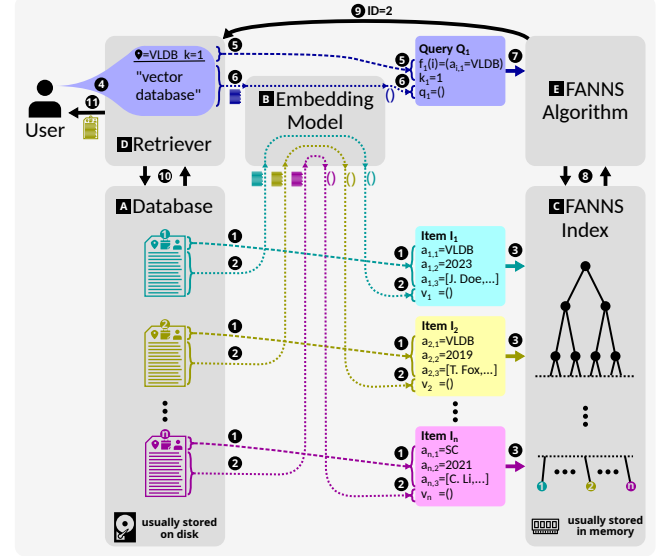


Figure 1: (§2.2) Example use case of FANNS.

### 2.3 Attribute Types

We provide a list of *attribute* types supported by FANNS algorithms.

- **Unordered attributes:** These *attributes* contain a single value from a domain with no total order. An example is the venue of a published paper, as each paper appears in a single venue, and no total order exists between venues.
- **Ordered attributes:** These *attributes* contain a single value from a totally ordered domain. An example is the publication year of a paper, as each paper has a single publication year, and years follow a total order.
- **Set attributes:** These *attributes* contain a set of values from an unordered domain. For instance, the authors of a research paper can be represented as a *set attribute*.

### 2.4 Filter Types

We categorize filters that apply to a single *attribute* into three types. Table 2 shows the compatibility of filters types with *attribute* types.

- **Exact match (EM) filter:** Matches only if an *item*’s *attribute* value is equal to the queried value.
- **Range (R) filter:** Used on an *ordered attribute* and matches only if an *item*’s *attribute* value is within the queried range.
- **Exact match in set (EMIS) filter:** Applies to a *set attribute* and matches only if the *item*’s set attribute contains the queried value.

FANNS algorithms that apply filtering to multiple *attributes* of the same type implement **multiple exact match (MEM)**, **multiple range (MR)**, and **multiple exact match in set (MEMIS)** filters. We define a **combined (C) filter** as a filter capable of combining the three basic single-attribute filters using arbitrary logical operators.

Table 2: (§2.4) Which filter applies to which attribute.

	Unordered attribute	Ordered attribute	Set attribute
Exact match filter	✓	✓	✗
Range filter	✗	✓	✗
Exact match in set filter	✗	✗	✓

## 2.5 Approximate Nearest Neighbor Search

Since many FANNS algorithms build on ANNS methods without filtering, we first introduce the most relevant ANNS algorithms. These can be categorized into tree-based, hash-based, graph-based, and quantization-based methods. For a comprehensive discussion of ANNS, we refer to the surveys by Han et al. [62] and Echiabi et al. [39]. Experimental evaluations [11, 12] indicate that no single ANNS algorithm consistently outperforms others; rather, performance depends heavily on dataset characteristics.

**2.5.1 Tree-based methods.** Tree-based methods partition the vector space into multiple regions by constructing a search tree, where nodes represent either regions or boundaries between them. Lower tree levels typically correspond to finer partitioning of the vector space. Tree-based methods, such as **k-d trees** [19], **R-trees** [61], **ball trees** [107], **cover trees** [22], **RP trees** [30], **M-trees** [27], **K-means trees** [126], and **best bin first** [17], along with other variants [65, 91, 103, 118], are more efficient for lower-dimensional vectors, as they often suffer from the curse of dimensionality [95].

**2.5.2 Hash-based methods.** Hash-based methods use a set of hash functions to map vectors from a continuous  $d$ -dimensional space into discrete hash buckets. These functions are designed so that similar vectors are mapped into the same bucket with high probability. When querying a hash-based index, the query vector is hashed, and the vectors in the corresponding bucket are compared to the query vector. Hash-based methods include **locality-sensitive hashing (LSH)** [31, 68], **spectral hashing** [139], **iDEC** [54], **deep hashing** [86], **mmlsh** [70], **PM-LSH** [152], **R2LSH** [90], **EI-LSH** [87], and others [5, 6, 48, 52, 66, 83, 92, 108, 124].

**2.5.3 Graph-based methods.** Graph-based methods represent vectors as vertices in a graph. Two vertices are connected if their corresponding vectors are close to each other, forming a neighborhood graph [8, 130]. For simplicity, we define the *length* of an edge as the distance between its endpoints’ vectors. To find the nearest neighbors of a query vector, the graph is traversed starting from one or multiple entry points, following the direction of the smallest distance to the query vector. A drawback of neighborhood graphs is that for large datasets, many edges must be traversed, leading to high query times. **Navigable small world (NSW)** graphs [93] mitigate this issue by introducing long edges that significantly reduce the number of steps needed for traversal. **Hierarchical navigable small world (HNSW)** graphs [94] further improve the

performance of NSW by introducing a hierarchical structure, where higher layers contain longer edges and lower layers contain shorter ones. The HNSW graph is traversed from the highest to the lowest level. Additional graph-based ANNS methods include **DiskANN** [71], **FreshDiskANN** [120], **NSG** [46], **GRNG** [43], and others [44, 45, 63, 69, 150].

**2.5.4 Quantization-based methods.** Quantization-based methods map the  $n$  vectors to a set of  $c \ll n$  clusters, each represented by a cluster centroid. Each vector is assigned to the cluster whose centroid is closest to the vector. One can either store all vectors assigned to a given cluster, as in **inverted file (IVF)** [122], or approximate all vectors of a cluster by the respective cluster centroid, which is less accurate but requires less storage space and allows for faster query times. To find the nearest neighbors of a query vector, we only compare it to the database vectors in the  $w \leq c$  clusters whose centroids are closest to the query vector. **Product quantization (PQ)** [72] is a more advanced quantization scheme that splits the vector space into  $s$  orthogonal subspaces. Each subspace is quantized separately by finding a set of  $c$  ( $d/s$ )-dimensional cluster centroids and assigning each vector to its closest cluster within each subspace. One of the most prominent quantization-based ANNS algorithms is **inverted file with product quantization (IVF-PQ)** [72], which consists of two levels. The first level employs an IVF index, but instead of storing the full vectors within each cluster, it approximates them by constructing a separate PQ index for each cluster. Other quantization-based ANNS methods include **additive quantization** [14], **BAPQ** [58], **LOPQ** [73], **OPQ** [51], **RaBitQ** [49], **SPANN** [26], and additional approaches [7, 15, 98, 134, 137].

## 3 SURVEY AND TAXONOMY OF FANNS

### 3.1 Filtering Approaches

Two common approaches to solving the FANNS problem are **pre-filtering** and **post-filtering**. In *pre-filtering*, an attribute-only index such as a B-tree [16], B+-tree [1], or qd-tree [146] identifies all items matching the filter. The  $k$  nearest neighbors (KNN) of these items are then determined by computing the distance to the query vector or approximating this distance using quantized embedding vectors precomputed during index construction. In *post-filtering*, a nearest neighbor search is performed on an unfiltered ANNS index, followed by filtering out non-matching items. This can be done by retrieving  $k' > k$  items in the hope that enough pass the filter or by iteratively retrieving more items until  $k$  matches are found. Many graph-based ANNS indices support a third approach, **in-filtering**, where attributes are ignored during index construction, and only vertices satisfying the filter are considered during query execution. A fourth approach involves a **hybrid index** that integrates embedding vectors and attributes in a single index.

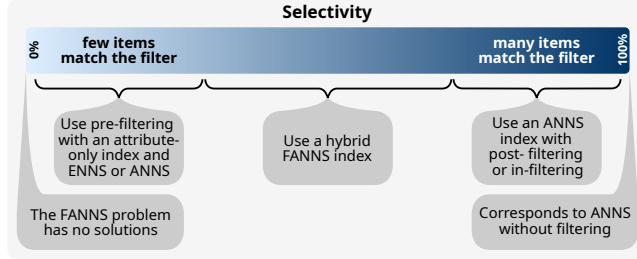
The efficiency of different approaches to solving the FANNS problem depends on the dataset and filtering condition (see Figure 2). We define the selectivity of a filter  $f$  on a dataset  $D = \{I_i \mid i \in \{1, \dots, n\}\}$  as in previous work<sup>1</sup> [85, 101, 109, 149]:

$$\text{selectivity} = \frac{|\{I_i \mid I_i \in D \wedge f(I_i) = \text{true}\}|}{|D|}.$$

<sup>1</sup>Note that some works [131, 138, 143] use the inverse definition of selectivity.

**Table 3: (§3) Overview of FANNS methods.** <sup>†</sup> Does not mention a filtering step during query execution. \*We have not been able to find the source code of this work. ❶ The index is constructed for a single ordered attribute, in-filtering is proposed to handle secondary attributes. ❷ Any EMIS filter can be used as an EM filter by using a set attribute with only a single value.

Method	Year	Problem	Approach	Technique(s)	Filters							Open source
					EM	R	EMIS	MEM	MR	MEMIS	C	
Rii [97]	2018	FANNS	pre-/in-filtering	quantization	✓	✓	✓	✓	✓	✓	✓	✓[96]
MA-NSW [141]	2019	FANNS	hybrid index	graph	✓	✗	✗	✓	✗	✗	✗	✗*
PASE [144]	2020	FANNS	post-filtering	quantization / graph	✓	✓	✓	✓	✓	✓	✓	✓[4]
AnalyticDB-V [138]	2020	FANNS	pre-/post-filtering	quantization	✓	✓	✓	✓	✓	✓	✓	✗*
Milvus [131]	2021	FANNS	pre-/post-filtering	quantization / graph	✗	✓	✗	✗	✓	✗	✗	✓[112]
NHQ [132, 133]	2022	AFANNS	hybrid index	graph	✓	✗	✗	✓	✗	✗	✗	✓[47]
HQANN [145]	2022	AFANNS <sup>†</sup>	hybrid index	graph	✓	✗	✗	✓	✗	✗	✗	✗*
AIRSHIP [151]	2022	FANNS	in-filtering	graph	✓	✓	✓	✓	✓	✓	✓	✗*
HQI [101]	2023	FANNS	hybrid index	tree + quantization	✓	✓	✗	✓	✓	✗	✓	✗*
VBASE [149]	2023	FANNS	post-filtering	any	✓	✓	✓	✓	✓	✓	✓	✓[100]
FDANN [53]	2023	FANNS	hybrid index	graph	❷	✗	✓	✗	✗	✗	✗	✓[99]
CAPS [60]	2023	FANNS	hybrid index	quantization + tree	✓	✗	✗	✓	✗	✗	✗	✓[59]
ARKGraph [153]	2023	FANNS	hybrid index	graph	✗	✓	✗	✗	✗	✗	✗	✓[9]
ACORN [109]	2024	FANNS	in-filtering	graph	✓	✓	✓	✓	✓	✓	✓	✓[57]
β-WST [41]	2024	FANNS	hybrid index	tree + graph	✗	✓	✗	✗	✗	✗	✗	✓[40]
SeRF [154]	2024	FANNS	hybrid index	graph	✗	✓	✗	✗	❶	✗	✗	✓[10]
iRangeGraph [143]	2024	FANNS	hybrid index	tree + graph	✗	✓	✗	✗	❶	✗	❶	✓[142]
UNIFY [85]	2024	FANNS	hybrid index	graph	✗	✓	✗	✗	✗	✗	✗	✓[33]
UNG [24]	2024	FANNS	hybrid index	graph	❷	✗	✓	❷	✗	✓	✗	✓[23]



**Figure 2: (§3.1) The optimal approach for solving FANNS depends on the filter’s selectivity within a given dataset.**

In other words, selectivity is the fraction of database items that match the filter. *Pre-filtering* is most efficient when selectivity is low, meaning only a few items remain for the ENNS. *Post-filtering* and *in-filtering* are most efficient when selectivity is high, i.e., most items pass the filter. If selectivity is too low, ANNS may not return enough valid candidates for *post-filtering*, and graph traversal with *in-filtering* may fail due to a sparsely connected or disconnected graph. A *hybrid index* is most efficient when selectivity is moderate, making none of the three previous approaches optimal.

### 3.2 Classification of FANNS Methods

We classify FANNS methods along three dimensions:

- **Approach:** Each FANNS method follows one or a combination of the following approaches: pre-filtering, post-filtering, in-filtering, or hybrid indexing (see Section 3.1).
- **Technique:** FANNS methods build upon the same four fundamental techniques as ANNS methods: tree-based, hash-based, graph-based, or quantization-based (see Section 2.5).

- **Filter Type:** Different FANNS methods support various filter types (see Section 2.4 for details).

Table 3 provides an overview of existing FANNS methods classified along these three dimensions.

### 3.3 Explanation of FANNS Methods

**3.3.1 Rii.** The Rii [97] index is based on IVF-PQ. It takes a bitmap of matching items as input. If the selectivity is low, it compares the query vector directly with the efficiently accessible, contiguously stored PQ codes (pre-filtering). If the selectivity is high, it first performs the IVF step of IVF-PQ to identify the closest clusters. Within those clusters, a PQ code is compared to the query vector only if the corresponding item matches the filter (in-filtering).

**3.3.2 MA-NSW.** MA-NSW [141] is a graph-based FANNS index that supports both EM and MEM filters. It constructs multiple NSW-like, graph-based ANNS indices, one for each possible combination of attribute values. During query execution, MA-NSW queries the ANNS index that matches the query’s filter expression to approximately retrieve the KNN that satisfy the filter conditions.

**3.3.3 PASE.** PASE [144] integrates ANNS into the PostgreSQL [56] database by implementing the IVF and HNSW ANNS indices. It employs an iterative post-filtering approach, in which it repeatedly retrieves items from the ANNS index and filters them until  $k$  matching items are found.

**3.3.4 AnalyticDB-V.** Alibaba’s AnalyticDB-V [138] extends the AnalyticDB [147] SQL database with FANNS capabilities. It introduces its own ANNS index, called *Voronoi graph product quantization* (VG PQ), which is similar to IVF-PQ. For FANNS implementation, it supports both pre-filtering and post-filtering.

**3.3.5 Milvus.** Milvus [131] extends the FAISS ANNS library [38, 114] by incorporating range filtering for one or multiple attributes. It supports both pre- and post-filtering, as well as a partition-based filtering scheme that partitions the data based on the most frequently filtered attribute and builds a separate ANNS index for each partition.

**3.3.6 NHQ.** NHQ [133] proposes a method to extend any proximity graph-based ANNS index into a hybrid FANNS index. Instead of constructing the ANNS index solely based on embedding distance, it introduces a fusion distance that combines embedding distance with attribute dissimilarity. While this approach is applicable to any proximity graph-based ANNS index, NHQ also presents construction schemes for two novel proximity graphs. Since its query execution scheme operates using the fusion distance without an explicit filtering step, NHQ may return items that do not match the filter. Thus, it addresses the AFANNS problem rather than the FANNS problem.

**3.3.7 HQANN.** Like NHQ [133], HQANN [145] employs a fusion distance to transform any proximity graph-based ANNS index into a hybrid FANNS index. The key difference between HQANN and NHQ is that HQANN prioritizes attribute dissimilarity in the fusion distance, whereas NHQ emphasizes embedding distance as the dominant factor.

**3.3.8 AIRSHIP.** AIRSHIP [151] (see Figure 3) is a graph-based FANNS index capable of handling arbitrary filter functions. It relies on a graph-based ANNS index constructed without considering attributes but adapts query execution to FANNS through in-filtering and a series of optimizations. During query execution, AIRSHIP traverses the entire graph but includes only vertices that satisfy the filter in the result set. Additionally, it selects a starting point within a cluster of items guaranteed to match the filter and traverses the graph concurrently in multiple directions.

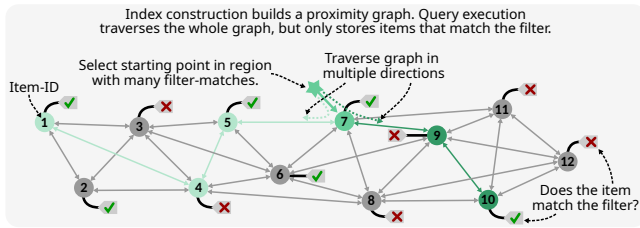


Figure 3: (§3.3.8) Visualization of the AIRSHIP index.

**3.3.9 HQI.** Apple’s HQI [101] (see Figure 4) is a hybrid, workload-aware index designed for batch processing of FANNS queries. To integrate attribute filtering with nearest neighbor search, HQI first transforms embedding vectors into attributes by applying k-means clustering and storing each item’s cluster ID as an attribute. It then employs a balanced qd-tree [146], an attribute-only index, to partition the items. Each non-leaf node in the qd-tree contains a predicate over the item attributes (including the cluster ID), with its two child nodes holding items that do or do not satisfy the predicate. The tree’s leaves form a non-overlapping partition of all items. During query execution, HQI identifies the query vector’s  $w$  closest cluster IDs and prunes all branches of the tree that do not contain any of these  $w$  cluster IDs or whose attributes do not match the filter. An IVF index is used within each leaf.

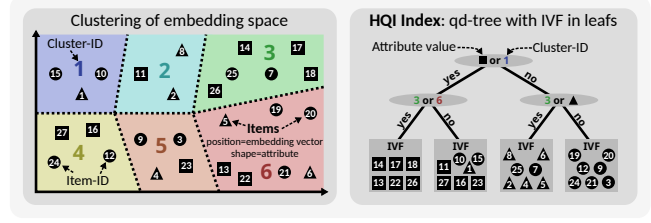


Figure 4: (§3.3.9) Visualization of the HQI index.

**3.3.10 VBASE.** VBASE [149] extends PostgreSQL [56] by integrating ANNS into the database system. Its key insight is that most ANNS query execution methods follow a two-stage process: an initial stage where traversal moves from a starting point in the direction of the query vector, and a second stage where the search gradually moves away from the query vector to identify its KNN. Rather than employing a naïve pre-filtering approach, where the ANNS index is queried with a fixed  $k' \geq k$  and non-matching items are filtered out, VBASE refines query execution by running the ANNS query only until the traversal starts moving away from the query vector. From that point onward, it iteratively retrieves and filters additional items until the required  $k$  matches are found.

**3.3.11 FDANN.** Microsoft’s Filtered-DiskANN (FDANN) [53] (see Figure 5) implements a graph-based index with EMIS filtering. In addition to supporting an EMIS filter with a single query-label, it also allows filtering with multiple query-labels, where at least one of them must be present in the item’s set attribute. Filtered-DiskANN introduces two methods for constructing FANNS indices based on Vamana graphs [71]: *FilteredVamana* and *StitchedVamana*. During query execution, it traverses the graph starting from a set of entry points that are guaranteed to contain the query-label. Throughout its NSW-like graph traversal, it considers only those vertices that satisfy the filter criteria.

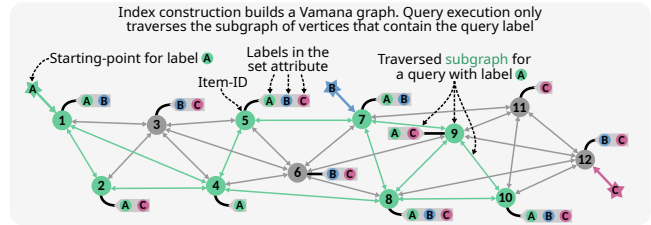


Figure 5: (§3.3.11) The Filtered-DiskANN index visualized.

**3.3.12 CAPS.** In contrast to most FANNS methods, CAPS [60] (see Figure 6) introduces a quantization- and tree-based index that is easier to parallelize than the more common graph-based indices. The CAPS index consists of two levels. The first level organizes items into clusters, for example, using an IVF [122]. This clustering can be based solely on embedding distance or on a combination of embedding distance and attribute dissimilarity. At the second level, CAPS constructs an *attribute frequency tree* (AFT), a tree-based attribute-only index. Each level of the AFT splits items into two buckets: one containing items with the most common attribute value (a leaf node) and another with the remaining items (a non-leaf node), unless the maximum depth is reached.



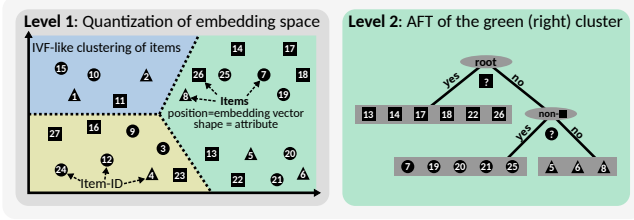


Figure 6: (§3.3.12) Visualization of the CAPS index.

During query execution, CAPS first selects the  $w$  clusters closest to the query vector (first level). Within these  $w$  clusters, it uses AFTs to identify items matching the filter and then performs ENNS on the filtered results (second level).

**3.3.13 ARKGraph.** Given a set of items, each with a single *ordered attribute*, ARKGraph [153] addresses the more general problem of constructing the *K-nearest-neighbor graph* (KGraph) [37] for the subset of items that satisfy a range filter. The resulting KGraph can be utilized for solving the FANNS problem or for other data analysis tasks. A naïve approach would store  $O(n^2)$  KGraphs, one for each possible query range. For any given item  $I$  with attribute value  $x$ , this would require maintaining  $O(n^2)$  different lists of KNN, one for each possible query range  $[l, r]$  such that  $l \leq x \leq r$ . ARKGraph’s first key insight is that the KNN of  $I$  for any query range  $[l, r]$  can be efficiently computed by merging the KNN of  $I$  from the partial query ranges  $[l, x]$  and  $(x, r]$ . Thus, instead of storing KNN lists for all  $O(n^2)$  query ranges, it suffices to maintain KNN lists for only  $O(n)$  partial query ranges and reconstruct the KNN dynamically for any given range. The second key insight is that an item often shares the same KNN across multiple similar partial query ranges. By grouping query ranges and storing only one KNN set per group, ARKGraph further reduces index size and storage overhead. Refer to Figure 7 for a comparison of the three approaches to storing an item’s KNN across multiple query ranges.

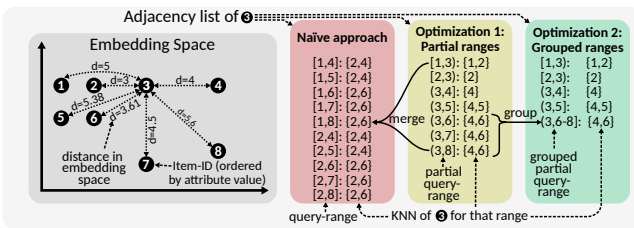


Figure 7: (§3.3.13) Visualization of the adjacency lists of item 3 in ARKGraph using the three different approaches.

**3.3.14 ACORN.** ACORN [109] (see Figure 8) supports arbitrary filter types and filtering across multiple attributes. The ACORN index is a denser variant of the HNSW ANNS index [94], constructed without considering attributes. During query execution, ACORN traverses only the subgraph induced by vertices that match the filter. This approach differs from AIRSHIP [151], which traverses the entire graph but includes only matching vertices in the result set. The ACORN index is designed such that its subgraphs approximate an HNSW index [94], ensuring efficient nearest neighbor search while maintaining filtering constraints.

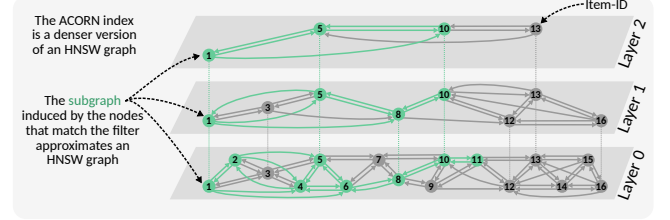
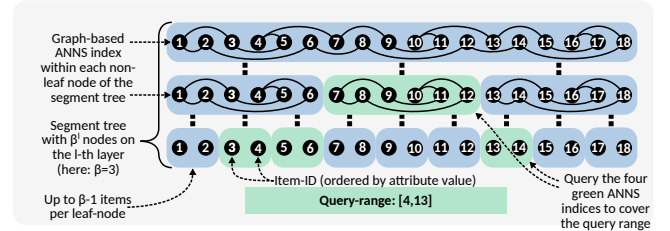


Figure 8: (§3.3.14) Visualization of the ACORN index.

**3.3.15  $\beta$ -WST.**  $\beta$ -WST [41] (see Figure 9) is a FANNS index designed for range filtering on a single attribute. A  $\beta$ -WST index is structured as a segment tree [34] with a uniform branching factor of  $\beta$  (see Figure 9). At layer  $l$ , the attribute value range is divided into  $\beta^l$  segments, with a graph-based ANNS index constructed for each segment. The authors propose multiple query strategies for  $\beta$ -WST. The default strategy selects a minimal set of tree nodes that fully cover the query range, queries the corresponding ANNS indices, and merges the results. Additional strategies include *OptimizedPostfiltering*, *TreeSplit*, and *SuperPostfiltering*.

Figure 9: (§3.3.15) Visualization of the  $\beta$ -WST for  $\beta = 3$ .

**3.3.16 SeRF.** SeRF [154] introduces the *segment graph*, a data structure designed to efficiently answer range-filtered FANNS queries with half-bounded query ranges (i.e., where the query range has an upper limit but no lower limit). The segment graph losslessly compresses  $n$  HNSW indices [94], corresponding to the  $n$  possible half-bounded query ranges, into a single index, which eliminates the need to explicitly construct all  $n$  HNSW indices. This is achieved by incrementally inserting items into an HNSW index in attribute-value order. Each edge in the resulting graph stores the attribute value at which it was added and, if pruned during construction, the attribute value at which it was removed. As a result, the HNSW graph contains edges that are valid only within specific attribute value intervals. When executing a query on the segment graph with a half-bounded query range, only edges whose validity interval contains the upper limit of the query range are considered. To generalize this approach for arbitrary query ranges (i.e., with both upper and lower limits), SeRF introduces the *2D segment graph* (see Figure 10). This structure compresses  $n$  segment graphs, one for each possible lower limit, into a single graph with  $n$  vertices, where edges store two validity intervals: one for the lower and one for the upper limit of the query range. Additionally, SeRF proposes an in-filtering approach to extend its method to queries involving two or more attributes.

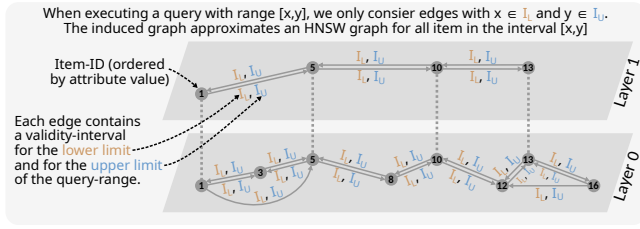


Figure 10: (§3.3.16) SeRF’s 2D segment graph visualized.

**3.3.17 iRangeGraph.** iRangeGraph [143] (see Figure 11) is a graph-based FANNS index designed for range filtering. It is primarily built around a single *ordered attribute* but also supports in-filtering or post-filtering for secondary attributes of arbitrary types. iRangeGraph constructs a set of graph-based ANNS indices, referred to as *elemental graphs*. These elemental graphs are organized in a segment tree [34] with  $O(\log n)$  layers. At layer  $l$ , the entire attribute-value range is divided into  $2^l$  segments, and a graph-based ANNS index is built for each segment, ensuring that each item appears once in each of the  $\log n$  layers. Query execution employs an efficient on-the-fly method to dynamically merge a subset of elemental graphs into a single graph-based ANNS index containing only items within the query range. While its index construction is similar to  $\beta$ -WST [41], iRangeGraph differs significantly in query execution: instead of querying multiple graph-based indices for different subsets of items and merging the results, it constructs a single graph-based ANNS index dynamically at query time.

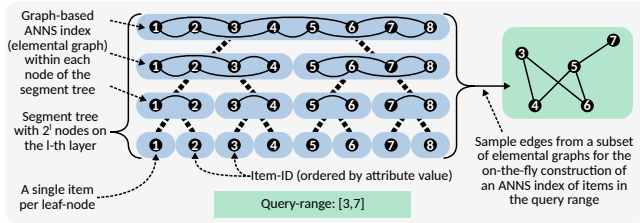


Figure 11: (§3.3.17) Visualization of the iRangeGraph index.

**3.3.18 UNIFY.** UNIFY [85] is a graph-based method that supports range filtering. It introduces two new types of proximity graph, the *segmented inclusive graph* (SIG) and its HNSW-like variant the *hierarchical segmented inclusive graph* (HSIG) (see Figure 12). UNIFY assumes a single, *ordered attribute*. It divides the range of possible attribute values into  $s$  segments, s.t., each segment contains approximately the same number of items. The high-level idea of a SIG is to build a proximity graph [135] for each of the  $2^s - 1$  possible combinations of segments, and then merge these graphs into a single graph that contains the  $2^s - 1$  smaller graphs as subgraph. Due to a clever construction algorithm, the SIG can be built without explicitly constructing all  $2^s - 1$  smaller graphs. The SIG is stored as a segmented adjacency list, where outgoing edges are grouped by their destination segment. When querying the SIG, we only traverse the subgraph that correspond to those segments that intersect with the query range. The HSIG is a multi-layered variant of the SIG which is traversed from top to bottom in order to speed-up the query execution. To handle queries with low selectivity, UNIFY supports a pre-filtering variant and for queries with high selectivity, it supports unfiltered ANNS with post-filtering.

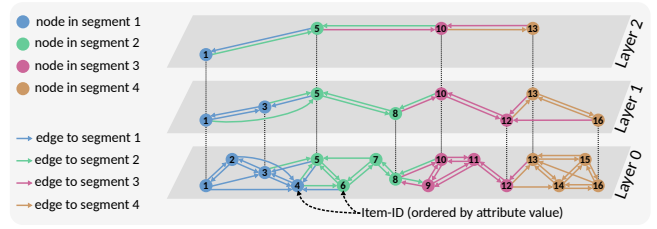


Figure 12: (§3.3.18) UNIFY’s HSIG index visualized.

**3.3.19 UNG.** UNG [24] (see Figure 13) is a graph-based FANNS index that supports both EMIS and MEMIS filters. It first identifies all distinct label sets in the database and constructs a *Label Navigating Graph* (LNG), which is a *Directed Acyclic Graph* (DAC) with one vertex for each distinct label set and a directed edge from label set  $a$  to label set  $b$  if  $a$  is a subset of  $b$ . The *Unified Navigating Graph* (UNG) index is built by constructing a proximity graph for each vertex in the LNG, where the vertices of each proximity graph represent the items with the corresponding label set. To unify these proximity graphs, cross-edges are added between vertices in the proximity graphs of label sets  $a$  and  $b$  if there is a corresponding edge from  $a$  to  $b$  in the LNG. When querying the UNG, the standard graph traversal is initiated, starting in all proximity graphs whose label sets are minimal supersets of the query label set.

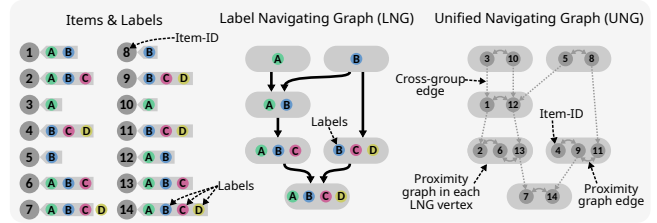
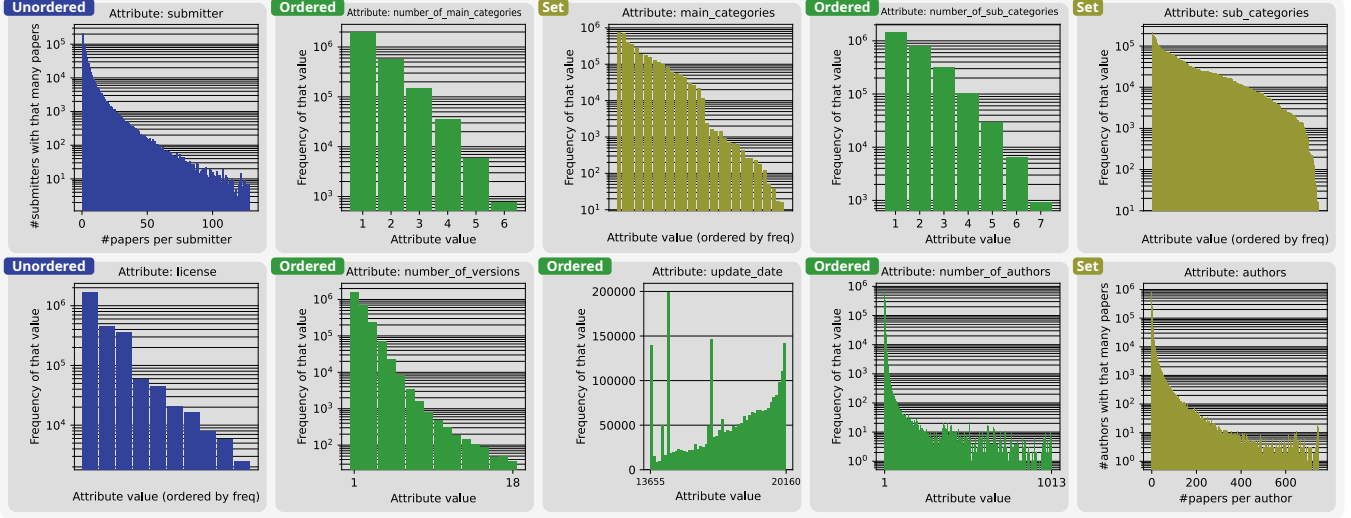


Figure 13: (§3.3.19) Visualization of the UNG index.

## 4 A NEW DATASET TO BENCHMARK FANNS

Benchmarking FANNS methods requires suitable datasets. Such datasets should contain vectors from a state-of-the-art embedding model (requirement 1), real-world attributes (requirement 2), a set of queries (vectors and filters) together with the pre-computed ground truth (requirement 3), and be publicly available (requirement 4).

The most common approach to evaluate FANNS methods that we observed in scientific literature is to use an ANNS dataset such as SIFT [76], GIST [76], MNIST [77], GloVe [110], UQ-V [47], Msong [47], Audio [47], Crawl [47], Enron [47], BIGANN [119], Paper [47], Deep1B [36], MSTuring [119], or YandexT2I [119] and extend it with synthetic, randomly generated attributes [24, 41, 53, 60, 85, 101, 131, 133, 141, 145, 151, 154] (failing to meet requirement 2). Another common, though more labor-intensive, approach is to repurpose a dataset not originally intended for FANNS such as TripClick [113], LAION [116], RedCaps [35], YouTube-Rgb [2], Youtube-Audio [2], Words [42], MTG [127], or WIT-Image [32] by manually creating embedding vectors and queries, or by crawling the web to obtain suitable real-world attributes [24, 41, 85, 109, 143, 154] (failing to meet requirement 3). Finally, some papers originating from industry rely on proprietary in-house datasets that are not publicly available [101, 138, 145] (violating requirement 4).



**Figure 14: (§4) Distribution of the unordered attributes (blue), ordered attributes (green), and set attributes (yellow) in the arxiv-for-fanns-large dataset. We show inverse histograms for *authors* and *submitter* since they have many possible values and only few items with a given value. Outliers are omitted for clarity. The *has\_comment* attribute is true for 74.216% of items.**

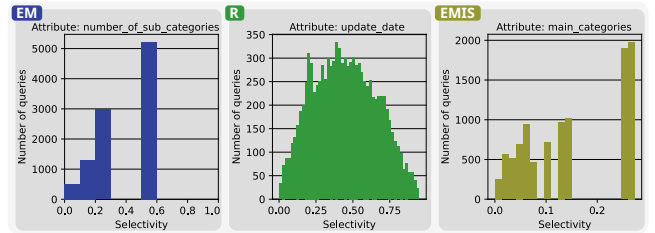
The only two datasets that appear to satisfy all four requirements are *MNIST* [77] and *Paper* [47]. The *MNIST* dataset, where the labels correspond to the digits 0 to 9, and the *Paper* dataset, where each attribute has at most three possible values, cover only a narrow subset of potential FANNS use cases. Moreover, we observe that datasets with text embeddings from state-of-the-art, transformer-based models such as NV-Embed [78, 105], LENS [79, 80], GTE [3, 84], Stella [104, 148], and BGE [25, 106] are absent from the current ANNS and FANNS benchmarks (not in line with requirement 1). These models typically generate normalized embedding vectors with around 4096 dimensions, representing a fundamentally different FANNS workload compared to existing datasets, which usually contain embeddings with fewer than 1000 dimensions. We therefore argue that an additional datasets with transformer-based embeddings would be highly valuable to the community.

**Table 4: (§4) Attributes in the novel arXiv dataset.**

Name	Attribute Type	Data type
submitter	unordered attribute	string
has_comment	unordered attribute	boolean
number_of_main_categories	ordered attribute	integer
main_categories	set attribute	list of strings
number_of_sub_categories	ordered attribute	integer
sub_categories	set attribute	list of strings
license	unordered attribute	string
number_of_versions	ordered attribute	integer
update_date	ordered attribute	integer
number_of_authors	ordered attribute	integer
authors	set attribute	list of strings

To address this gap, we introduce the *arxiv-for-fanns* dataset, where each item corresponds to a research paper. The dataset is available in three scales: *small* (1k items), *medium* (100k items), and *large* (over 2.7M items). Each item includes a 4096-dimensional, normalized text embedding of the paper abstract, computed using

the *stella\_en\_400M\_v5* model [104, 148], based on abstracts from the arXiv dataset [28], thus satisfying requirement 1. Among the top 10 models on the English MTEB leaderboard [102], Stella is the most lightweight<sup>2</sup> [18]. Each item also contains 11 real-world attributes (see Table 4) covering all three attribute types (see Section 2.3), with diverse value distributions shown in Figure 14, fulfilling requirement 2. To satisfy requirement 3, each dataset variant includes three sets of 10k queries for EM, R, and EMIS filters (see Section 2.4), along with their corresponding precomputed ground truth. Query vectors are generated by embedding search terms synthesized by GPT-4o [67], while filter values are sampled from the dataset attributes. Figure 15 illustrates the selectivity distribution (as defined in Section 3.1) across queries in *arxiv-for-fanns-large*. To support unfiltered ANNS evaluation, we also provide ground truth results without any filtering. All three versions of the dataset are publicly available on Hugging Face<sup>3</sup>, thereby fulfilling requirement 4.



**Figure 15: (§4) Selectivity of the three query sets.**

## 5 BENCHMARKING FANNS

We benchmark a selection of FANNS methods on our newly introduced *arxiv-for-fanns* dataset (see Section 4). Experiments with the medium-scale dataset are conducted on a laptop, while those on the large-scale dataset are executed on a compute cluster.

<sup>2</sup>As of March 11, 2025

<sup>3</sup><https://huggingface.co/datasets/SPCL/arxiv-for-fanns-large>



**Table 5: (§5.2) Parameters used for benchmarking (found through parameter search, see Section 5.2).**

Method	Experiment	Parameters
ACORN [109]	all	$efs \in \{10, 15, 20, 25, 30, 50, 100, 250, 500, 750\}$
	medium, EM filter	$M = 16, M_\beta = 24, \gamma = 10$
	medium, R filter	$M = 32, M_\beta = 24, \gamma = 12$
	medium, EMIS filter	$M = 16, M_\beta = 24, \gamma = 15$
	large, EM filter	$M = 32, M_\beta = 32, \gamma = 10$
	large, R filter	$M = 32, M_\beta = 16, \gamma = 12$
	large, EMIS filter	$M = 48, M_\beta = 48, \gamma = 15$
CAPS (kmeans) [60]	all	$m \in \{200, 400, 1k, 5k, 10k, 20k, 40k, 60k\}$
	medium, EM filter	$B = 128$
	large, EM filter	$B = 512$
FDANN (stitched) [53]	all	$L \in \{10, 20, 30, 50, 100, 150, 200, 300, 500, 1k\}$
	medium, EM filter	$R_{\text{small}} = 32, L_{\text{small}} = 80, R_{\text{stitched}} = 48, \alpha = 1.1$
	medium, EMIS filter	$R_{\text{small}} = 32, L_{\text{small}} = 100, R_{\text{stitched}} = 48, \alpha = 1.2$
	large, EM filter	$R_{\text{small}} = 96, L_{\text{small}} = 100, R_{\text{stitched}} = 64, \alpha = 1.4$
	large, EMIS filter	$R_{\text{small}} = 64, L_{\text{small}} = 80, R_{\text{stitched}} = 64, \alpha = 1.2$
NHQ (kgraph) [133]	all	$L_{\text{search}} \in \{10, 25, 50, 75, 100, 150, 200, 300, 400\}$ , iter = 12, $M = 1.0$
	medium, EM filter	$K = 100, L = 80, S = 15, R = 200, \text{RANGE} = 70, PL = 200, B = 0.5, \text{weight\_search} = 1M$
	large, EM filter	$K = 80, L = 60, S = 10, R = 200, \text{RANGE} = 60, PL = 300, B = 0.6, \text{weight\_search} = 1M$
NHQ (nsw) [133]	all	$efSearch \in \{50, 100, 150, 200, 300, 500, 1k, 2k, 4k\}$
	medium, EM filter	$M = 40, \text{MaxM0} = 40, efConstruction = 300, \text{weight\_search} = 1M$
	large, EM filter	$M = 40, \text{MaxM0} = 60, efConstruction = 150, \text{weight\_search} = 1M$
UNG [24]	all	$L_{\text{search}} \in \{10, 20, 30, 40, 50, 100, 150, 200, 300, 500, 1k\}$
	medium, EM filter	$\delta = 2, R = 24, L = 80, \alpha = 1.4, \sigma = 16$
	medium, EMIS filter	$\delta = 8, R = 32, L = 100, \alpha = 1.4, \sigma = 16$
	large, EM filter	$\delta = 6, R = 48, L = 150, \alpha = 1.6, \sigma = 24$
	large, EMIS filter	$\delta = 6, R = 48, L = 150, \alpha = 1.4, \sigma = 24$
SeRF [154]	all	$ef\_search \in \{4, 8, 16, 32, 64, 128, 256, 512, 1024\}$
	medium, R filter	$\text{index\_k} = 100, ef\_construction = 100, ef\_max = 500$
	large, R filter	$\text{index\_k} = 100, ef\_construction = 100, ef\_max = 600$

## 5.1 Collected Metrics

We focus on the well-established recall vs. queries per second (QPS) plots, which illustrate the trade-off between accuracy and query throughput achieved by each method. We define recall@ $k$  as:

$$\text{recall}@k = \frac{|\text{knn}_{\text{alg}} \cap \text{knn}_{\text{gt}}|}{k}$$

where  $\text{knn}_{\text{alg}}$  denotes the set of  $k$  nearest neighbors returned by the algorithm, and  $\text{knn}_{\text{gt}}$  is the ground truth. In addition, we report the index construction time, peak memory usage during both index construction and query execution, and the index size.

## 5.2 Parameter Search

Since all FANNS methods considered involve tunable parameters that influence index construction, we perform a dedicated parameter search for each combination of method, dataset scale (medium vs. large), and filter type prior to benchmarking. Given that some methods expose up to eight parameters, a full grid search is computationally infeasible. Instead, we adopt a greedy parameter search strategy, outlined in Algorithm 1. The algorithm takes as input a list of tunable parameters, their corresponding candidate values, and the index of each parameter’s default value inside the list of candidate values. Candidate and default values are selected based on the literature and publicly available implementations of each method. The `get_reward()` function performs two iterations of building

the index with a given parameter configuration and querying it using 50 randomly sampled queries from the dataset. It computes the recall vs. QPS curve averaged over both iterations and returns the highest QPS achieved at a recall of at least 0.95. If a method does not reach a recall of 0.95, it returns the highest achieved recall instead. The resulting parameter are listed in Table 5.

## 5.3 Benchmarking Methodology

Once the parameter search is complete, we benchmark each method five times (without warm-up) and report the mean and standard deviation for all measured metrics following scientific benchmarking practice [64]. Index construction is performed using all available hardware threads on the respective system (details provided in the next paragraph), while query execution is restricted to a single thread<sup>4</sup>, following common practice [24, 60, 133]. We search for  $k = 10$  nearest neighbors and report recall@10, which is a standard choice in the literature [24, 53, 109, 133, 145]. All benchmarks use Euclidean distance, as it is supported by all methods under evaluation. Since our transformer-based embeddings are normalized, KNN under Euclidean distance is equivalent to KNN under cosine distance. We exclude any preprocessing, such as sorting items by attribute value or transforming vectors into alternate formats when reporting index construction time.

<sup>4</sup>Our script reports multiple threads during query execution of CAPS and SeRF, however, CPU utilization confirms that only one thread is active and the others are idle.

**Algorithm 1 (§5.2) Greedy Parameter Search**


---

**Require:** params, value\_lists, default\_indices

```

1: indices = default_indices
2: P = {p : value_lists[p][indices[p]] for p in params}
3: best_reward = get_reward(P)
4: do_repeat = true
5: while do_repeat do
6:   do_repeat = false
7:   for all par ∈ params do
8:     for all change ∈ [-1, 1] do
9:       P = {p : value_lists[p][indices[p]] for p in params}
10:      P[par] = value_lists[par][indices[par] + change]
11:      reward = get_reward(P)
12:      if reward > best_reward then
13:        if reward > (best_reward * 1.01) then
14:          do_repeat = true
15:        end if
16:        best_reward = reward
17:        indices[par] = indices[par] + change
18:      end if
19:    end for
20:  end for
21: end while
22: return {p : value_lists[p][best_indices[p]] for p in params}

```

---

**5.4 Software and Hardware Configuration**

To ensure reproducibility, we encapsulate our benchmarking infrastructure in a Docker container based on Ubuntu 20.04.6, which includes all necessary dependencies for the evaluated methods. Inside the container, we use Python 3.8.10, gcc 9.4.0, and g++ 9.4.0. Benchmarks on the *arxiv-for-fanns-medium* dataset are conducted on a laptop equipped with 16 GB of RAM and an Intel Core i7-1165G7 CPU with 4 physical cores and 8 hardware threads, running Arch Linux with kernel version 6.15.5-arch1-1. For the *arxiv-for-fanns-large* dataset, we execute benchmarks on a single node of a compute cluster with 384 GB of RAM and two Intel Xeon Gold 6154 CPUs, each with 18 physical cores and 36 hardware threads, running CentOS Linux 8.

**5.5 Remarks on Algorithms**

*NHQ-kgraph*. We exclude the call to `optimize_graph()` from query execution timing, as it is independent of the number of queries and its cost is amortized when processing large batches.

*ACORN*. Algorithmically, ACORN only needs to check the filter condition for visited vertices during graph traversal. However, the implementation we benchmark performs this check for all items prior to traversal. Since prior work is inconsistent on whether this cost is included [24] or not [109], we report two recall vs. QPS curves for ACORN: one including the filtering overhead (labeled “upper bound”) and one excluding it (labeled “lower bound”).

*SeRF*. Unlike all other methods, SeRF performs index construction and query execution within a single process, without persisting the index to disk. Therefore, we report only the overall peak memory usage, and we omit the index size from our results.

*FDANN*. The code provided in the FDANN repository [99] stores two versions of the index, each approximately the size of the original dataset. We optimize the code to store only one index at a time, reducing the overall index size to half.

**5.6 Results on *arxiv-for-fanns-medium***

*Recall vs. QPS (Figure 16)*. We observe that ACORN, the only evaluated method applicable to all three filter types, is generally outperformed by more specialized methods. For the EM filter, NHQ (kgraph), FDANN, and UNG achieve the best performance and perform similarly, whereas CAPS, NHQ (nsw), and ACORN are slower at comparable recall levels. The relative performance of ACORN, NHQ (kgraph), and UNG is consistent with previous benchmarks [24] on lower-dimensional datasets. FDANN, which showed inferior performance in those earlier studies, excels with our transformer-based embeddings, while NHQ (nsw) and CAPS appear to underperform. For the EMIS filter, previous benchmarks on lower-dimensional datasets [24] indicate that UNG outperforms FDANN and ACORN; however, on our transformer-based embeddings, FDANN demonstrates superior performance.

*Index Construction Time (Figure 17, left)*. We observe that index construction time varies by up to an order of magnitude across methods. CAPS is the fastest to build, followed by FDANN and NHQ (kgraph). SeRF, UNG, and NHQ (nsw) are significantly slower, with ACORN being the slowest across all filters.

*Peak Memory Usage (Figure 17, middle)*. Except for NHQ (kgraph) and SeRF, which show noticeably higher peak memory usage, differences in memory consumption during index construction and query execution are moderate across methods.

*Index Size (Figure 17, right)*. The index built by most methods is approximately 1.6 GB, corresponding to the size of the dataset. CAPS and NHQ (kgraph) produce very small indexes, only a few megabytes; however, they need to access the original database vectors during query execution. If we account for the size of the database vectors, the effective index size for these methods is also around 1.6 GB.

**5.7 Challenges In Scaling Up the Dataset Size**

We encountered several challenges when scaling our benchmarking from the *arxiv-for-fanns-medium* dataset with 100,000 items to the *arxiv-for-fanns-large* dataset with over 2.7 million items. While 2.7 million vectors may not sound particularly large, note that our transformer-based embeddings have 4,096 dimensions, significantly more than most existing datasets. As a result, our dataset is over 85× larger than the well-known SIFT1M dataset [76], which contains 128-dimensional embedding vectors.

Due to this large scale, 4 out of the 7 methods we benchmarked required modifications to their source code to run with the large dataset. UNG [24], CAPS [60], and NHQ (KGraph) [133] all suffered from `int32` overflows during memory allocation. Additionally, CAPS originally allocated a data structure on the stack, which we moved to the heap to support the large dataset. FDANN [53], as mentioned in Section 5.5, initially stored two versions of the index, exceeding the local memory of our cluster’s compute node. Addressing the issues enabled benchmarking all 7 algorithms.

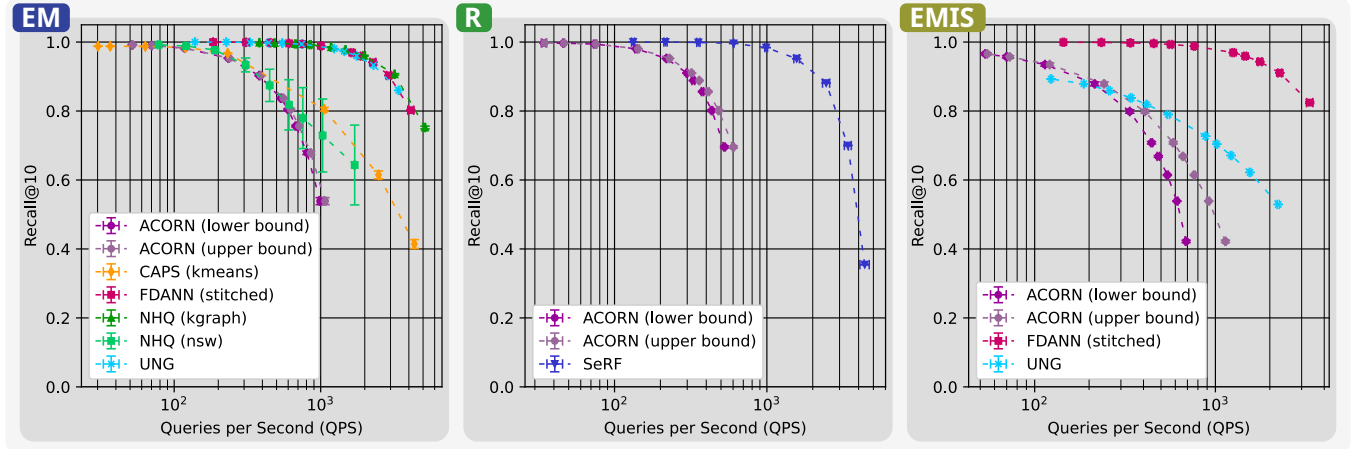


Figure 16: (§5.6) Recall@10 vs. QPS plots for the three filter types (EM, R, EMIS) on the arxiv-for-fanns-medium dataset.

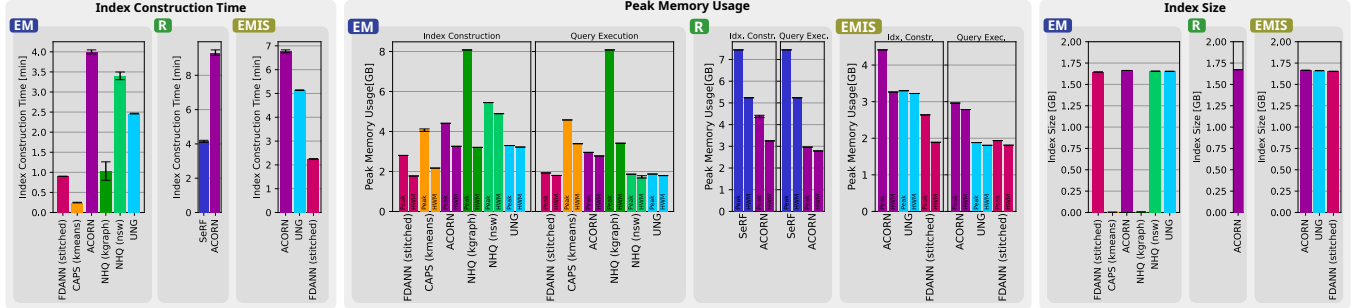


Figure 17: (§5.6) Index construction time (left), peak memory usage (middle), and index size (right) for the three filter types (EM, R, EMIS) on the arxiv-for-fanns-medium dataset.

## 5.8 Results on arxiv-for-fanns-large

*Recall vs. QPS (Figure 18).* Comparing the recall vs. QPS curves of the large-scale dataset to those of the medium-scale dataset reveals key insights into method scalability. ACORN, both NHQ variants, and SeRF maintain their relative performance, with query throughput dropping by only a factor of about 4 despite the dataset being 27 $\times$  larger. This indicates good scalability, especially considering that both experiments used a single thread for query execution, albeit on different machines. FDANN, CAPS, and UNG perform poorly on the large dataset and fail to reach more than 25% recall. We cannot fully rule out that the poor performance is due to sub-optimal parameters, as our greedy search may have converged to a local optimum. A full grid search or a different initialization could potentially lead to better results. However, even with the greedy search, we encountered computational limits, as the parameter search for some methods with long index construction times and many parameters ran for multiple weeks. These findings highlight the challenge of tuning FANNS methods on large datasets, where index construction is expensive. We believe further research into efficient parameter tuning strategies would benefit the community.

*Index Construction Time (Figure 19, left).* Index construction on the 27 $\times$  larger dataset increased the runtime of most methods by roughly an order of magnitude. Since the large dataset was processed with 9 $\times$  more threads, this suggests that index construction time grows faster than linearly with dataset size or that speedup from additional threads is sublinear. CAPS remains the fastest

method, while ACORN continues to lag behind most others. UNG has by far the longest construction time, which may be due to UNG failing to reach the target recall of 0.95, causing the parameter search to shift toward more expensive configurations.

*Peak Memory Usage (Figure 19, middle).* Memory usage of all methods scales roughly linearly with dataset size. Relative differences during index construction and query execution remain similar to those on the medium-scale dataset.

*Index Size (Figure 19, right).* UNG stands out by compressing the index to about one fourth of the dataset size without requiring access to the original vectors at query time. As in the medium-scale setting, CAPS and NHQ (kgraph) produce compact indexes of just a few megabytes but require access to the original vectors at query time. For the remaining methods, the index size is approximately equal to the dataset size.

## 6 RELATED WORK

In the first part of our work, we survey the current state of the FANNS field. While we are not aware of any existing surveys specifically focused on FANNS, several works review the related ANNS domain, such as the surveys by Echihabi et al. [39] and Han et al. [62]. Other surveys explore prominent application areas of ANNS and FANNS, most notably in the context of retrieval-augmented generation (RAG) for LLMs [50] and reasoning language models (RLMs) with integrated RAG capabilities [20].

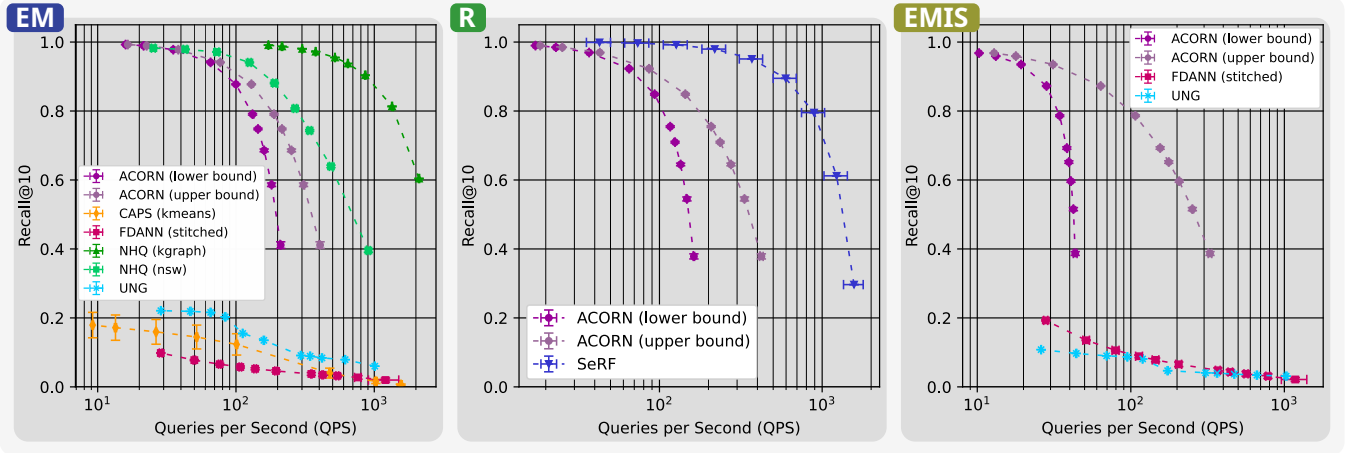


Figure 18: (§5.8) Recall@10 vs. QPS plots for the three filter types (EM, R, EMIS) on the arxiv-for-fanns-large dataset.

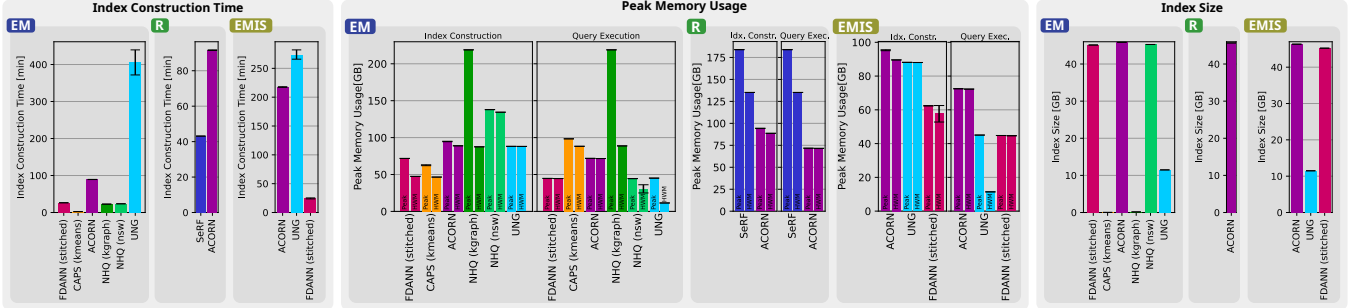


Figure 19: (§5.8) Index construction time (left), peak memory usage (middle), and index size (right) for the three filter types (EM, R, EMIS) on the arxiv-for-fanns-large dataset.

The second part of our paper introduces a novel dataset featuring transformer-based embedding vectors, designed to reflect emerging RAG workloads. This type of data is not represented in the existing landscape of ANNS and FANNS datasets [36, 76, 110, 119]. Our embeddings are generated using the `stella_en_400M_v5` model [104, 148], a widely used embedding model in LLM research [21, 74]. In contrast to our arxiv-for-fanns dataset, end-to-end RAG benchmarks such as the Massive Text Embedding Benchmark [102] contain text passages rather than embedding vectors, making them unsuitable for direct evaluation of FANNS methods.

Finally, we benchmark a range of existing FANNS methods [24, 53, 60, 109, 133, 154] on our dataset. While each of these methods includes evaluation results in its respective publication, there is no comprehensive benchmark of FANNS methods on modern, transformer-based embeddings. In the broader ANNS field, several large-scale benchmarks [11, 12, 136] have been established, providing valuable comparisons across methods and datasets.

## 7 CONCLUSION

The rapid progress of embedding models for various modalities has intensified the need for fast and accurate FANNS methods. In response, many approaches supporting EM, R, and EMIS filtering have emerged recently. To structure this evolving field, we present a taxonomy and survey of modern FANNS methods. We identify a key gap: the lack of open datasets with transformer-based embeddings

enriched with real-world attributes. To address this, we release the arxiv-for-fanns dataset with over 2.7 million 4096-dimensional embedding vectors and 11 real-world attributes. We benchmark several FANNS methods on this dataset and analyze their performance under this challenging workload. No single method excels universally. ACORN is robust and flexible but often outperformed by specialized methods. SeRF performs well for range filters on ordered attributes but lacks categorical filtering support. Filtered-DiskANN and UNG perform well on medium-scale data but fail to scale, highlighting the difficulty of handling high-dimensional transformer-based embeddings. We conclude that the large dimensionality of transformer-based vectors poses a major challenge for FANNS methods. Moreover, we find that efficient parameter tuning remains one of the most pressing open problems in this area.

## 8 ACKNOWLEDGEMENTS

We thank the Swiss National Supercomputing Centre (CSCS) for access to their Ault system, which we used to execute our benchmarks. Furthermore, we thank the PLGrid Consortium for access to their Athena system, which we used to create the arxiv-for-fanns dataset. This work was supported by the ETH Future Computing Laboratory (EFCL), financed by a donation from Huawei Technologies. It also received funding from the European Research Council (Project PSAP, No. 101002047) and from the European Union’s HE research and innovation programme under the grant agreement No. 101070141 (Project GLACIATION).



## REFERENCES

- [1] David J Abel. 1984. A B+-tree structure for large quadrees. *Computer Vision, Graphics, and Image Processing* 27, 1 (1984), 19–31.
- [2] Sami Abu-El-Haija, Anja Hauth, Lu Jiang, Nisarg Kothari, Joonseok Lee, Hanhan Li, Paul Natsev, Joe Ng, Sobhan Naderi Parizi, George Toderici, Balakrishnan Varadarajan, Sudheendra Vijayanarasimhan, and Shouo-I Yu. 2025. YouTube 8M. <https://research.google.com/youtube8m/download.html>. Accessed: 2025-03-06.
- [3] Alibaba-NLP. 2025. gte-Qwen2-7B-instruct. <https://huggingface.co/Alibaba-NLP/gte-Qwen2-7B-instruct>. Accessed: 2025-01-22.
- [4] Alipay. 2025. Pase: PostgreSQL Ultra-High Dimensional Approximate Nearest Neighbor Search Extension. <https://github.com/alipay/PASE>. Accessed: 2025-03-04.
- [5] Alexandr Andoni, Piotr Indyk, Thijs Laarhoven, Ilya Razenshteyn, and Ludwig Schmidt. 2015. Practical and optimal LSH for angular distance. *Advances in neural information processing systems* 28 (2015).
- [6] Alexandr Andoni and Ilya Razenshteyn. 2015. Optimal data-dependent hashing for approximate near neighbors. In *Proceedings of the forty-seventh annual ACM symposium on Theory of computing*. 793–801.
- [7] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2016. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. In *42nd International Conference on Very Large Data Bases*, Vol. 9. 12.
- [8] Kazuo Aoyama, Kazumi Saito, Hiroshi Sawada, and Naonori Ueda. 2011. Fast approximate similarity search based on degree-reduced neighborhood graphs. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*. 1055–1063.
- [9] Data Curation Lab at Rutgers University. 2025. ARKGraph: All-Range Approximate K-Nearest-Neighbor Graph. <https://github.com/rutgers-db/ARKGraph>. Accessed: 2025-02-25.
- [10] Data Curation Lab at Rutgers University. 2025. SeRF. <https://github.com/rutgers-db/SeRF>. Accessed: 2025-02-24.
- [11] Martin Aumüller, Erik Bernhardsson, and Alec Faithfull. 2025. ANN Benchmarks. <https://ann-benchmarks.com/index.html>. Accessed: 2025-01-22.
- [12] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.
- [13] Yusuf Aytar, Carl Vondrick, and Antonio Torralba. 2016. Soundnet: Learning sound representations from unlabeled video. *Advances in neural information processing systems* 29 (2016).
- [14] Artem Babenko and Victor Lempitsky. 2014. Additive quantization for extreme vector compression. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 931–938.
- [15] Artem Babenko and Victor Lempitsky. 2014. The inverted multi-index. *IEEE transactions on pattern analysis and machine intelligence* 37, 6 (2014), 1247–1260.
- [16] Rudolf Bayer and Edward McCreight. 1970. Organization and maintenance of large ordered indices. In *Proceedings of the 1970 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. 107–141.
- [17] Jeffrey S Beis and David G Lowe. 1997. Shape indexing using approximate nearest-neighbor search in high-dimensional spaces. In *Proceedings of IEEE computer society conference on computer vision and pattern recognition*. IEEE, 1000–1006.
- [18] Massive Text Embedding Benchmark. 2025. Overall MTEB English leaderboard. [https://huggingface.co/spaces/mteb/leaderboard\\_legacy](https://huggingface.co/spaces/mteb/leaderboard_legacy). Accessed: 2025-03-11.
- [19] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.
- [20] Maciej Besta, Julia Barth, Eric Schreiber, Ales Kubicek, Afonso Catarino, Robert Gerstenberger, Piotr Nyczyk, Patrick Iff, Yueling Li, Sam Houlston, et al. 2025. Reasoning language models: A blueprint. *arXiv preprint arXiv:2501.11223* (2025).
- [21] Maciej Besta, Lorenzo Paleari, Ales Kubicek, Piotr Nyczyk, Robert Gerstenberger, Patrick Iff, Tomasz Lehmann, Hubert Niewiadomski, and Torsten Hoeffler. 2024. Checkembed: Effective verification of llm solutions to open-ended tasks. *arXiv preprint arXiv:2406.02524* (2024).
- [22] Alina Beygelzimer, Sham Kakade, and John Langford. 2006. Cover trees for nearest neighbor. In *Proceedings of the 23rd international conference on Machine learning*. 97–104.
- [23] Yuzheng Cai. 2025. Unified Navigating Graph Algorithm for Filtered Approximate Nearest Neighbor Search. <https://github.com/YZ-Cai/Unified-Navigating-Graph>. Accessed: 2025-04-23.
- [24] Yuzheng Cai, Jiayang Shi, Yizhuo Chen, and Weiguo Zheng. 2024. Navigating Labels and Vectors: A Unified Approach to Filtered Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–27.
- [25] Jianlv Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. 2024. Bge m3-embedding: Multi-lingual, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216* (2024).
- [26] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.
- [27] Paolo Ciaccia, Marco Patella, Pavel Zezula, et al. 1997. M-tree: An efficient access method for similarity search in metric spaces. In *Vldb*, Vol. 97. Citeseer, 426–435.
- [28] Cornell University, Devrishi, Joe Tricot, Brian Maltzan, Shamsi Brinn, and Timo Bozsolk. 2025. arXiv Dataset. <https://www.kaggle.com/datasets/Cornell-University/arxiv>. Accessed: 2025-03-13.
- [29] Aurora Linh Cramer, Ho-Hsiang Wu, Justin Salamon, and Juan Pablo Bello. 2019. Look, listen, and learn more: Design choices for deep audio embeddings. In *ICASSP 2019-2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 3852–3856.
- [30] 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.
- [31] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.
- [32] Google Research Datasets. 2025. WIT : Wikipedia-based Image Text Dataset. <https://github.com/google-research-datasets/wit>. Accessed: 2025-03-06.
- [33] SJTU DBGroup. 2025. UNIFY - Unified Index for Range Filtered Approximate Nearest Neighbors Search. <https://github.com/sjtu-dbgroup/UNIFY>. Accessed: 2025-02-20.
- [34] Mark De Berg. 2000. *Computational geometry: algorithms and applications*. Springer Science & Business Media.
- [35] Karan Desai, Gaurav Kaul, Zubin Aysola, and Justin Johnson. 2025. RedCaps: Web-curated image-text data created by the people, for the people. <https://redcaps.xyz/>. Accessed: 2025-03-06.
- [36] Artem Babenko Dmitry Baranchuk. 2025. Benchmarks for Billion-Scale Similarity Search. <https://research.yandex.com/blog/benchmarks-for-billion-scale-similarity-search>. Accessed: 2025-03-06.
- [37] Wei Dong, Charikar Moses, and Kai Li. 2011. Efficient k-nearest neighbor graph construction for generic similarity measures. In *Proceedings of the 20th international conference on World wide web*. 577–586.
- [38] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvassy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). *arXiv:2401.08281 [cs.LG]*
- [39] Karima Echihabi, Kostas Zoumpatianos, and Themis Palpanas. 2021. New trends in high-d vector similarity search: al-driven, progressive, and distributed. *Proceedings of the VLDB Endowment* 14, 12 (2021), 3198–3201.
- [40] Josh Engels. 2025. RangeFilteredANN. <https://github.com/JoshEngels/RangeFilteredANN>. Accessed: 2025-02-25.
- [41] Joshua Engels, Benjamin Landrum, Shangdi Yu, Laxman Dhulipala, and Julian Shun. 2024. Approximate Nearest Neighbor Search with Window Filters. *arXiv preprint arXiv:2402.00943* (2024).
- [42] Eric Yang Farrall. 2025. Word Embeddings. <https://huggingface.co/datasets/efarrall/word-embeddings>. Accessed: 2025-04-23.
- [43] Cole Foster, Berk Sevilimis, and Benjamin Kimia. 2025. Generalized relative neighborhood graph (GRNG) for similarity search. *Pattern Recognition Letters* 188 (2025), 103–110.
- [44] Cong Fu, Changxu Wang, and Deng Cai. 2019. Satellite system graph: Towards the efficiency up-boundary of graph-based approximate nearest neighbor search. *CoRR* (2019).
- [45] Cong Fu, Changxu Wang, and Deng Cai. 2021. High dimensional similarity search with satellite system graph: Efficiency, scalability, and unindexed query compatibility. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44, 8 (2021), 4139–4150.
- [46] Cong Fu, Chao Xiang, Changxu Wang, and Deng Cai. 2017. Fast approximate nearest neighbor search with the navigating spreading-out graph. *arXiv preprint arXiv:1707.00143* (2017).
- [47] Yujian Fu. 2025. NHQ: Native Hybrid Query Framework for Vector Similarity Search with Attribute Constraint. <https://github.com/YujianFu97/NHQ>. Accessed: 2025-02-27.
- [48] Junhao Gan, Jianlin Feng, Qiong Fang, and Wilfred Ng. 2012. Locality-sensitive hashing scheme based on dynamic collision counting. In *Proceedings of the 2012 ACM SIGMOD international conference on management of data*. 541–552.
- [49] Jianyang Gao and Cheng Long. 2024. RaBitQ: quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.
- [50] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Haofen Wang, and Haofen Wang. 2023. Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997* 2 (2023).
- [51] 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.
- [52] Aristides Gionis, Piotr Indyk, Rajeev Motwani, et al. 1999. Similarity search in high dimensions via hashing. In *Vldb*, Vol. 99. 518–529.

- [53] Siddharth Gollapudi, Neel Karia, Varun Sivashankar, Ravishankar Krishnaswamy, Nikit Begwani, Swapnil Raz, Yiyong Lin, Yin Zhang, Neelam Mahapatro, Premkumar Srinivasan, et al. 2023. Filtered-diskann: Graph algorithms for approximate nearest neighbor search with filters. In *Proceedings of the ACM Web Conference 2023*. 3406–3416.
- [54] Long Gong, Huayi Wang, Mitsunori Ogiwara, and Jun Xu. 2020. iDEC: indexable distance estimating codes for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 13, 9 (2020).
- [55] Patrick Grother, Patrick Grother, Mei Ngan, and Kayee Hanaoka. 2019. Face recognition vendor test (frvt) part 2: Identification.
- [56] The PostgreSQL Global Development Group. 2025. PostgreSQL. <https://www.postgresql.org/>. Accessed: 2025-01-22.
- [57] Department of Computer Science Guestin Lab at Stanford University. 2025. ACORN. <https://github.com/guestin-lab/ACORN>. Accessed: 2025-02-25.
- [58] Qin-Zhen Guo, Zhi Zeng, Shuwu Zhang, Guixuan Zhang, and Yuan Zhang. 2016. Adaptive bit allocation product quantization. *Neurocomputing* 171 (2016), 866–877.
- [59] Gaurav Gupta. 2025. CAPS. <https://github.com/gaurav16gupta/constrainedANN>. Accessed: 2025-02-26.
- [60] Gaurav Gupta, Jonah Yi, Benjamin Coleman, Chen Luo, Vihan Lakshman, and Anshumali Shrivastava. 2023. CAPS: A Practical Partition Index for Filtered Similarity Search. *arXiv preprint arXiv:2308.15014* (2023).
- [61] 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.
- [62] Yikun Han, Chunjiang Liu, and Pengfei Wang. 2023. A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703* (2023).
- [63] Ben Harwood and Tom Drummond. 2016. Fanng: Fast approximate nearest neighbour graphs. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 5713–5722.
- [64] Torsten Hoefler and Roberto Belli. 2015. Scientific benchmarking of parallel computing systems: twelve ways to tell the masses when reporting performance results. In *Proceedings of the international conference for high performance computing, networking, storage and analysis*. 1–12.
- [65] Michael E Houle and Michael Nett. 2014. Rank-based similarity search: Reducing the dimensional dependence. *IEEE transactions on pattern analysis and machine intelligence* 37, 1 (2014), 136–150.
- [66] Qiang Huang, Jianlin Feng, Yikai Zhang, Qiong Fang, and Wilfred Ng. 2015. Query-aware locality-sensitive hashing for approximate nearest neighbor search. *Proceedings of the VLDB Endowment* 9, 1 (2015), 1–12.
- [67] Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, et al. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276* (2024).
- [68] Piotr Indyk and Rajeev Motwani. 1998. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 604–613.
- [69] Masajiro Iwasaki. 2016. Pruned bi-directed k-nearest neighbor graph for proximity search. In *International Conference on Similarity Search and Applications*. Springer, 20–33.
- [70] Omid Jafari, Parth Nagarkar, and Jonathan Montaña. 2020. mmlsh: A practical and efficient technique for processing approximate nearest neighbor queries on multimedia data. In *International Conference on Similarity Search and Applications*. Springer, 47–61.
- [71] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnaswamy, and Rohan Kadekodi. 2019. Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems* 32 (2019).
- [72] 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.
- [73] Yannis Kalantidis and Yannis Avrithis. 2014. Locally optimized product quantization for approximate nearest neighbor search. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2321–2328.
- [74] Sejong Kim, Hyunseo Song, Hyunwoo Seo, and Hyunjun Kim. 2025. Optimizing retrieval strategies for financial question answering documents in retrieval-augmented generation systems. *arXiv preprint arXiv:2503.15191* (2025).
- [75] Noam Koenigstein, Parikshit Ram, and Yuval Shavitt. 2012. Efficient retrieval of recommendations in a matrix factorization framework. In *Proceedings of the 21st ACM international conference on Information and knowledge management*. 535–544.
- [76] Hervé Jégou Laurent Amsaleg. 2025. Datasets for approximate nearest neighbor search. <http://corpus-texmex.irisa.fr/>. Accessed: 2025-03-06.
- [77] Yann LeCun. 2025. mnist. <https://huggingface.co/datasets/ylecun/mnist/viewer?views%5B%5D=train>. Accessed: 2025-03-06.
- [78] Chankyu Lee, Rajarshi Roy, Mengyao Xu, Jonathan Raiman, Mohammad Shoeibi, Bryan Catanzaro, and Wei Ping. 2024. NV-Embed: Improved Techniques for Training LLMs as Generalist Embedding Models. *arXiv preprint arXiv:2405.17428* (2024).
- [79] Yibin Lei. 2025. LENS-d8000. <https://huggingface.co/yibinlei/LENS-d8000>. Accessed: 2025-01-22.
- [80] Yibin Lei, Tao Shen, Yu Cao, and Andrew Yates. 2025. Enhancing Lexicon-Based Text Embeddings with Large Language Models. *arXiv preprint arXiv:2501.09749* (2025).
- [81] Adam Lerer, Ledell Wu, Jiajun Shen, Timothee Lacroix, Luca Wehrstedt, Abhijit Bose, and Alex Peysakhovich. 2019. Pytorch-biggraph: A large scale graph embedding system. *Proceedings of Machine Learning and Systems* 1 (2019), 120–131.
- [82] Jie Li, Haifeng Liu, Chuanghua Gui, Jianyu Chen, Zhenyuan Ni, Ning Wang, and Yuan Chen. 2018. The design and implementation of a real time visual search system on JD E-commerce platform. In *Proceedings of the 19th International Middleware Conference Industry*. 9–16.
- [83] Mingjie Li, Ying Zhang, Yifang Sun, Wei Wang, Ivor W Tsang, and Xuemin Lin. 2020. I/O efficient approximate nearest neighbour search based on learned functions. In *2020 IEEE 36th international conference on data engineering (ICDE)*. IEEE, 289–300.
- [84] Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281* (2023).
- [85] Anqi Liang, Pengcheng Zhang, Bin Yao, Zhongpu Chen, Yitong Song, and Guangxu Cheng. 2024. UNIFY: Unified Index for Range Filtered Approximate Nearest Neighbors Search. *arXiv preprint arXiv:2412.02448* (2024).
- [86] Haomiao Liu, Ruiping Wang, Shiguang Shan, and Xilin Chen. 2016. Deep supervised hashing for fast image retrieval. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2064–2072.
- [87] Wanqi Liu, Hanchen Wang, Ying Zhang, Wei Wang, Lu Qin, and Xuemin Lin. 2021. EI-LSH: An early-termination driven I/O efficient incremental c-approximate nearest neighbor search. *The VLDB Journal* 30 (2021), 215–235.
- [88] Xinchun Liu, Wu Liu, Huadong Ma, and Huiyuan Fu. 2016. Large-scale vehicle re-identification in urban surveillance videos. In *2016 IEEE international conference on multimedia and expo (ICME)*. IEEE, 1–6.
- [89] Yi Liu, Minghui Wang, and Changxin Li. 2024. Research on High-Accuracy Indoor Visual Positioning Technology Using an Optimized SE-ResNeXt Architecture. In *Proceedings of the 2024 7th International Conference on Signal Processing and Machine Learning*. 313–320.
- [90] Kejing Lu and Mineichi Kudo. 2020. R2LSH: A nearest neighbor search scheme based on two-dimensional projected spaces. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1045–1056.
- [91] Kejing Lu, Hongya Wang, Wei Wang, and Mineichi Kudo. 2020. VHP: approximate nearest neighbor search via virtual hypersphere partitioning. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1443–1455.
- [92] Qin Lv, William Josephson, Zhe Wang, Moses Charikar, and Kai Li. 2017. Intelligent probing for locality sensitive hashing: Multi-probe LSH and beyond. *Proceedings of the VLDB Endowment* (2017).
- [93] Yury Malkov, Alexander Ponomarenko, Andrey Logvinov, and Vladimir Krylov. 2014. Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems* 45 (2014), 61–68.
- [94] 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.
- [95] Rosalind B Marimont and Marvin B Shapiro. 1979. Nearest neighbour searches and the curse of dimensionality. *IMA Journal of Applied Mathematics* 24, 1 (1979), 59–70.
- [96] Yusuke Matsui. 2025. Reconfigurable Inverted Index (Rii): IVFPQ-based fast and memory efficient approximate nearest neighbor search method with a subset-search functionality. <https://github.com/matsui528/rii>. Accessed: 2025-04-23.
- [97] Yusuke Matsui, Ryota Hinami, and Shin'ichi Satoh. 2018. Reconfigurable inverted index. In *Proceedings of the 26th ACM international conference on Multimedia*. 1715–1723.
- [98] Yusuke Matsui, Toshihiko Yamasaki, and Kiyoharu Aizawa. 2015. Pqtable: Fast exact asymmetric distance neighbor search for product quantization using hash tables. In *Proceedings of the IEEE International Conference on Computer Vision*. 1940–1948.
- [99] Microsoft. 2025. DiskANN. <https://github.com/microsoft/DiskANN>. Accessed: 2025-02-26.
- [100] Microsoft. 2025. <https://github.com/microsoft/MSVBASE>. <https://github.com/microsoft/MSVBASE>. Accessed: 2025-02-26.
- [101] Jason Mohoney, Anil Pacaci, Shihabur Rahman Chowdhury, Ali Mousavi, Ihab F Ilyas, Umar Farooq Minhas, Jeffrey Pound, and Theodoros Rekatsinas. 2023. High-throughput vector similarity search in knowledge graphs. *Proceedings of the ACM on Management of Data* 1, 2 (2023), 1–25.
- [102] Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. MTEB: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316* (2022).
- [103] Marius Muja and David G Lowe. 2014. Scalable nearest neighbor algorithms for high dimensional data. *IEEE transactions on pattern analysis and machine*

- intelligence 36, 11 (2014), 2227–2240.
- [104] NovaSearch. 2025. Stella\_em\_400M\_v5. [https://huggingface.co/NovaSearch/stella\\_en\\_400M\\_v5](https://huggingface.co/NovaSearch/stella_en_400M_v5). Accessed: 2025-03-11.
- [105] NVIDIA. 2025. NV-Embed-v2. <https://huggingface.co/nvidia/NV-Embed-v2>. Accessed: 2025-01-22.
- [106] Beijing Academy of Artificial Intelligence. 2025. bge-en-icl. <https://huggingface.co/BAAI/bge-en-icl>. Accessed: 2025-01-22.
- [107] Stephen M Omohundro. 1989. Five balltree construction algorithms. (1989).
- [108] Yongjoo Park, Michael Cafarella, and Barzan Mozafari. 2015. Neighbor-sensitive hashing. *Proceedings of the VLDB Endowment* 9, 3 (2015), 144–155.
- [109] Liana Patel, Peter Kraft, Carlos Guestrin, and Matei Zaharia. 2024. ACORN: Formant and Predicate-Agnostic Search Over Vector Embeddings and Structured Data. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.
- [110] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2025. glove100\_angular. [https://www.tensorflow.org/datasets/catalog/glove100\\_angular](https://www.tensorflow.org/datasets/catalog/glove100_angular). Accessed: 2025-03-06.
- [111] Pinecone. 2025. Pinecone: The vector database to build knowledgeable AI. <https://www.pinecone.io/>. Accessed: 2025-03-11.
- [112] The Milvus Project. 2025. Milvus. <https://github.com/milvus-io/milvus>. Accessed: 2025-02-27.
- [113] Navid Rekabsaz, Oleg Lesota, Markus Schedl, Jon Brassey, and Carsten Eickhoff. 2025. TripClick. <https://tripdatabase.github.io/tripclick/>. Accessed: 2025-07-28.
- [114] Meta Research. 2025. Faiss. <https://github.com/facebookresearch/faiss>. Accessed: 2025-01-17.
- [115] Fred Richardson, Douglas Reynolds, and Najim Dehak. 2015. A unified deep neural network for speaker and language recognition. *arXiv preprint arXiv:1504.00923* (2015).
- [116] Christoph Schuhmann. 2025. LAION-400-MILLION OPEN DATASET. <https://laion.ai/blog/laion-400-open-dataset/>. Accessed: 2025-03-06.
- [117] Anton Shapkin, Denis Litvinov, Yaroslav Zharov, Egor Bogomolov, Timur Galimzyanov, and Timofey Bryksin. 2023. Dynamic Retrieval-Augmented Generation. *arXiv preprint arXiv:2312.08976* (2023).
- [118] Chanop Silpa-Anan and Richard Hartley. 2008. Optimised KD-trees for fast image descriptor matching. In *2008 IEEE conference on computer vision and pattern recognition*. IEEE, 1–8.
- [119] Harsha Vardhan Simhadri, George Williams, Martin Aumüller, Artem Babenko, Dmitry Baranchuk, Qi Chen, Matthijs Douze, Lucas Hosseini, Ravishankar Krishnaswamy, Gopal Srinivasa, Suhas Jayaram Subramanya, and Jingdong Wang. 2025. Billion-Scale Approximate Nearest Neighbor Search Challenge: NeurIPS’21 competition track. <https://big-ann-benchmarks.com/neurips21.html>. Accessed: 2025-03-06.
- [120] Aditi Singh, Suhas Jayaram Subramanya, Ravishankar Krishnaswamy, and Harsha Vardhan Simhadri. 2021. Freshdiskann: A fast and accurate graph-based ann index for streaming similarity search. *arXiv preprint arXiv:2105.09613* (2021).
- [121] Sivic and Zisserman. 2003. Video Google: A text retrieval approach to object matching in videos. In *Proceedings ninth IEEE international conference on computer vision*. IEEE, 1470–1477.
- [122] Sivic and Zisserman. 2003. Video Google: A text retrieval approach to object matching in videos. In *Proceedings ninth IEEE international conference on computer vision*. IEEE, 1470–1477.
- [123] Ján Suchal and Pavol Návrát. 2010. Full text search engine as scalable k-nearest neighbor recommendation system. In *Artificial Intelligence in Theory and Practice III: Third IFIP TC 12 International Conference on Artificial Intelligence, IFIP AI 2010, Held as Part of WCC 2010, Brisbane, Australia, September 20-23, 2010. Proceedings* 3. Springer, 165–173.
- [124] Narayanan Sundaram, Aizana Turmukhametova, Nadathur Satish, Todd Mostak, Piotr Indyk, Samuel Madden, and Pradeep Dubey. 2013. Streaming similarity search over one billion tweets using parallel locality-sensitive hashing. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1930–1941.
- [125] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. 2015. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*. 1–9.
- [126] Pooya Tavallali, Peyman Tavallali, and Mukesh Singhal. 2021. K-means tree: an optimal clustering tree for unsupervised learning. *The journal of supercomputing* 77, 5 (2021), 5239–5266.
- [127] Trevor. 2025. Mtg Scryfall Cropped Art Embeddings. <https://huggingface.co/datasets/TrevorJS/mtg-scrryfall-cropped-art-embeddings-open-clip-ViT-SO400M-14-SigLIP-384>. Accessed: 2025-04-23.
- [128] Vectara. 2025. Vectara: The AI Agent and Assistant platform for enterprises. <https://www.vectara.com/>. Accessed: 2025-03-11.
- [129] Vespa. 2025. Vespa: We Make AI Work. <https://vespa.ai/>. Accessed: 2025-03-11.
- [130] Jingdong Wang and Shipeng Li. 2012. Query-driven iterated neighborhood graph search for large scale indexing. In *Proceedings of the 20th ACM international conference on Multimedia*. 179–188.
- [131] Jianguo Wang, Xiaomeng Yi, Rentong Guo, Hai Jin, Peng Xu, Shengjun Li, Xiangyu Wang, Xiangzhou Guo, Chengming Li, Xiaohai Xu, et al. 2021. Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 International Conference on Management of Data*. 2614–2627.
- [132] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2022. Navigable proximity graph-driven native hybrid queries with structured and unstructured constraints. *arXiv preprint arXiv:2203.13601* (2022).
- [133] Mengzhao Wang, Lingwei Lv, Xiaoliang Xu, Yuxiang Wang, Qiang Yue, and Jiongkang Ni. 2024. An efficient and robust framework for approximate nearest neighbor search with attribute constraint. *Advances in Neural Information Processing Systems* 36 (2024).
- [134] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631* (2021).
- [135] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631* (2021).
- [136] Mengzhao Wang, Xiaoliang Xu, Qiang Yue, and Yuxiang Wang. 2021. A comprehensive survey and experimental comparison of graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2101.12631* (2021).
- [137] Runhui Wang and Dong Deng. 2020. DeltaPQ: lossless product quantization code compression for high dimensional similarity search. *Proceedings of the VLDB Endowment* 13, 13 (2020), 3603–3616.
- [138] Chuangxian Wei, Bin Wu, Sheng Wang, Renjie Lou, Chaoqun Zhan, Feifei Li, and Yuanzhe Cai. 2020. AnalyticDB-V: a hybrid analytical engine towards query fusion for structured and unstructured data. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3152–3165.
- [139] Yair Weiss, Antonio Torralba, and Rob Fergus. 2008. Spectral hashing. *Advances in neural information processing systems* 21 (2008).
- [140] J Xu, A Szlam, and J Weston. 2021. Beyond goldfish memory: Long-term open-domain conversation. *arXiv 2021. arXiv preprint arXiv:2107.07567* (2021).
- [141] Xiaoliang Xu, Chang Li, Yuxiang Wang, and Yixing Xia. 2020. Multiattribute approximate nearest neighbor search based on navigable small world graph. *Concurrency and Computation: Practice and Experience* 32, 24 (2020), e5970.
- [142] Yuxuan Xu. 2025. iRangeGraph. <https://github.com/YuexuanXu7/iRangeGraph>. Accessed: 2025-02-21.
- [143] Yuxuan Xu, Jianyang Gao, Yutong Gou, Cheng Long, and Christian S Jensen. 2024. iRangeGraph: Improvising Range-dedicated Graphs for Range-filtering Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 6 (2024), 1–26.
- [144] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 2241–2253.
- [145] Wen Yang, Tao Li, Gai Fang, and Hong Wei. 2020. Pase: Postgresql ultra-high-dimensional approximate nearest neighbor search extension. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 2241–2253.
- [146] Zongheng Yang, Badrish Chandramouli, Chi Wang, Johannes Gehrke, Yinan Li, Umar Farooq Minhas, Per-Ake Larson, Donald Kossmann, and Rajeev Acharya. 2020. Qd-tree: Learning data layouts for big data analytics. In *Proceedings of the 2020 ACM SIGMOD international conference on management of data*. 193–208.
- [147] Chaoqun Zhan, Maomeng Su, Chuangxian Wei, Xiaoqiang Peng, Liang Lin, Sheng Wang, Zhe Chen, Feifei Li, Yue Pan, Fang Zheng, et al. 2019. AnalyticDB: real-time OLAP database system at Alibaba cloud. *Proceedings of the VLDB Endowment* 12, 12 (2019), 2059–2070.
- [148] Dun Zhang, Jiacheng Li, Ziyang Zeng, and Fulong Wang. 2024. Jasper and Stella: distillation of SOTA embedding models. *arXiv preprint arXiv:2412.19048* (2024).
- [149] Qianxi Zhang, Shuotao Xu, Qi Chen, Guoxin Sui, Jiadong Xie, Zhizhen Cai, Yaoqi Chen, Yinxuan He, Yuqing Yang, Fan Yang, et al. 2023. {VBASE}: Unifying Online Vector Similarity Search and Relational Queries via Relaxed Monotonicity. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 377–395.
- [150] Weijie Zhao, Shulong Tan, and Ping Li. 2020. Song: Approximate nearest neighbor search on gpu. In *2020 IEEE 36th International Conference on Data Engineering (ICDE)*. IEEE, 1033–1044.
- [151] Weijie Zhao, Shulong Tan, and Ping Li. 2022. Constrained approximate similarity search on proximity graph. *arXiv preprint arXiv:2210.14958* (2022).
- [152] Bolong Zheng, Zhao Xi, Lianggui Weng, Nguyen Quoc Viet Hung, Hang Liu, and Christian S Jensen. 2020. PM-LSH: A fast and accurate LSH framework for high-dimensional approximate NN search. *Proceedings of the VLDB Endowment* 13, 5 (2020), 643–655.
- [153] Chaoji Zuo and Dong Deng. 2023. ARKGraph: All-Range Approximate K-Nearest-Neighbor Graph. *Proceedings of the VLDB Endowment* 16, 10 (2023), 2645–2658.
- [154] Chaoji Zuo, Miao Qiao, Wenchao Zhou, Feifei Li, and Dong Deng. 2024. SeRF: Segment Graph for Range-Filtering Approximate Nearest Neighbor Search. *Proceedings of the ACM on Management of Data* 2, 1 (2024), 1–26.