*Research article*

# A comparative analysis of graph-based and partition-based approximate nearest neighbor search for large-scale entity resolution

Intelligent Decision Technologies
1–15

DOI: 10.1177/18724981251388888
journals.sagepub.com/home/idt

**Dimitrios Karapiperis[1], Leonidas Akritidis[1], Panayiotis Bozanis[1] and Vassilios S Verykios[2]**

**Abstract**

The discipline of Entity Resolution (ER), the process of identifying and linking records that refer to the same real-world entity, has been fundamentally reshaped by the adoption of high-dimensional vector embeddings. This transformation reframes ER as a large-scale Approximate Nearest Neighbor Search (ANNS) problem, making the choice of ANNS architecture a critical determinant of system performance. This paper provides a deep architectural comparison and a novel, large-scale empirical evaluation of the two dominant ANNS paradigms: graph-based methods (HNSW, DiskANN) and partition-based methods (Faiss-IVF+PQ, Scann). We introduce a new semi-synthetic benchmark tailored to the ER task, consisting of two one-million-vector datasets with a known ground truth. On this benchmark, we conduct a comprehensive evaluation, measuring not only total query time but also disaggregated blocking and matching times, alongside canonical ER quality metrics: precision, recall, and F1-score. Our findings reveal that partition-based methods, particularly Scann, offer superior performance in high-throughput, moderate-recall scenarios, while graph-based methods like HNSW and DiskANN are unequivocally superior for applications demanding the highest levels of matching quality. This work provides a nuanced, application-centric analysis that culminates in a set of actionable recommendations for practitioners designing modern data integration and retrieval systems.

**Keywords**

ANNS, Entity Resolution, Faiss, Scann, IVF, PQ

Received: 1 October 2025; accepted: 3 October 2025

## 1 Introduction

The discipline of Entity Resolution (ER), the process of identifying and linking records that refer to the same real-world entity, stands as a cornerstone of modern data integration and analytics. Historically, ER systems relied on meticulously handcrafted rules and feature-based comparisons, often involving string similarity metrics on structured attributes like names, addresses, and product identifiers.[1] While effective in constrained domains, these methods have proven brittle and difficult to scale in the face of the semantic complexity and heterogeneity of modern, large-scale datasets. The contemporary solution to this challenge has emerged from the field of representation learning, where deep learning models, like BERT,[2] are trained to map complex, multi-modal entities—such as text documents, images, or user profiles—into high-dimensional vector spaces, or embeddings. Sentence-BERT[3] models extend this concept to sentence-level tasks, efficiently encoding entire text spans into fixed-size embeddings.

[1]International Hellenic University, Thermi, Greece
[2]Hellenic Open University, Patras, Greece

**Corresponding author:**
Dimitrios Karapiperis, International Hellenic University, Thermi, Greece.
Email: dkarapiperis@ihu.edu.gr

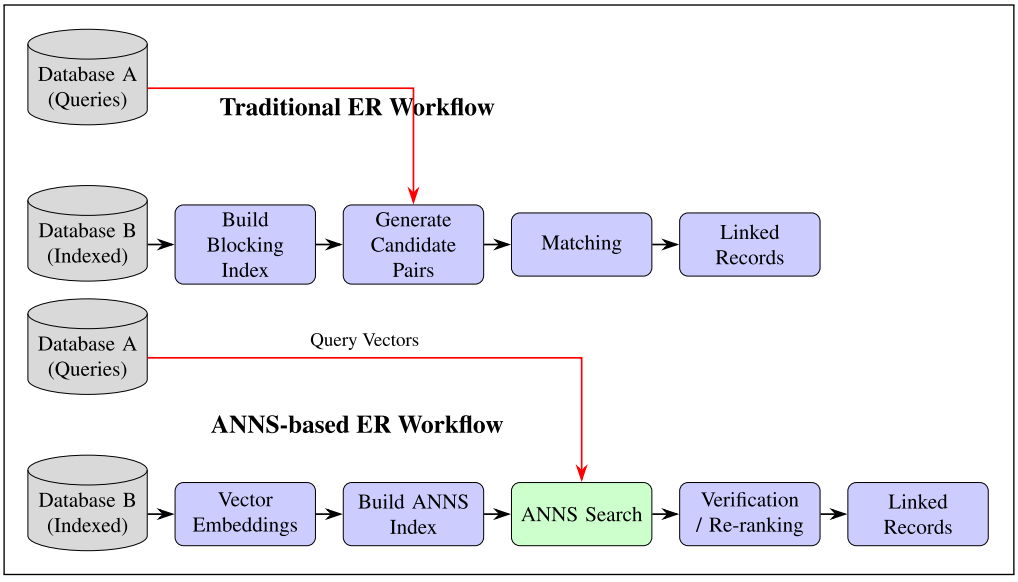


**Figure 1.** A schematic comparison of Traditional vs. ANNS-based Entity Resolution workflows. The top flow shows the classic bifurcated blocking and matching stages. The bottom flow illustrates how vector embeddings and ANNS algorithms unify these stages into a single, efficient search process.

In this new paradigm, the notion of semantic similarity is geometrically encoded: entities that are conceptually similar are represented by vectors that are proximate in the embedding space, typically measured by Euclidean distance or cosine similarity. This transformation fundamentally reframes the ER problem. The task of identifying matching entities across disparate datasets is now equivalent to finding the nearest neighbors for each vector in a target vector space. Consequently, the core ER workflow, traditionally bifurcated into a blocking stage to generate a manageable set of candidate pairs and a matching stage to perform detailed comparisons on those candidates [3], can be modeled as a large-scale Approximate Nearest Neighbor Search (ANNS) problem. This convergence implies that the architectural choices and performance characteristics of ANNS algorithms are no longer peripheral concerns but have become first-order determinants of the scalability, accuracy, and cost-efficiency of modern ER systems. The selection of an ANNS algorithm is now a critical architectural decision, directly influencing the quality and feasibility of data integration pipelines at scale.

To make this transition concrete, it is helpful to contrast the pipelines directly. We assume two databases: a query database $A$ and an indexed (blocked) database $B$. The traditional ER workflow is a distinct two-stage process: first, a blocking stage uses heuristics or indexing on key attributes to generate a manageable set of candidate pairs, and second, a matching stage applies detailed, often expensive, comparison functions to these pairs to identify true matches. The ANNS paradigm reframes this. As illustrated in Figure 1, vectorization unifies these stages. The search for nearest neighbors within an ANNS index inherently performs both blocking (by rapidly pruning the vast majority of the search space) and matching (by calculating distances to the most promising candidates) in a single, integrated process. Partition-based methods map to this paradigm explicitly, where identifying cells to search is the blocking step and scanning within them is the matching step. Graph-based methods perform an implicit, dynamic blocking by traversing the graph to a promising region before performing a fine-grained local search for the final match. This reframing makes the choice of ANNS algorithm the central architectural decision for modern ER systems.

The field of ANNS has matured significantly, with decades of research culminating in two dominant architectural paradigms that offer distinct trade-offs between search performance, memory footprint, and index construction complexity. These paradigms are distinguished by the fundamental structure of their indices and the nature of their search procedures.

The first paradigm is Graph-Based Methods. These algorithms construct a navigable proximity graph where the vertices represent the data vectors and the directed edges connect vertices that are close to each other in the embedding space. The search process is a guided traversal, typically a form of greedy beam search, that starts at a designated entry point and iteratively moves towards neighbors that are progressively closer to the query vector. This approach performs a dynamic, query-time exploration of the data's local structure. State-of-the-art representatives of this class include Hierarchical Navigable Small World (HNSW) graphs,[4] which build a multi-layered hierarchy to achieve logarithmic search complexity in memory, and DiskANN,[5] which constructs a specialized, low-diameter graph called Vamana, optimized for high-performance search on solid-state drives (SSDs).

The second dominant paradigm is Partition-Based Methods. These algorithms operate on a coarse-to-fine search principle. The vector space is first partitioned into a set of disjoint cells or clusters using a coarse quantization technique, most commonly k-eans. This partitioning forms an inverted file (IVF) structure,[6] where each cell corresponds to an inverted list containing the vectors assigned to it. A search is then restricted to a small subset of these cells—those whose centroids are closest to the query vector. This initial step acts as an explicit, static blocking mechanism. To manage memory usage for large datasets, the full-precision vectors within each cell are typically compressed using a second-stage quantization scheme. Prominent examples of this approach include Faiss-IVF with Product Quantization (PQ),[7] a highly optimized library that uses a Cartesian product of sub-quantizers for aggressive vector compression, and Scann (Scalable Nearest Neighbors),[8] which introduces an innovative anisotropic quantization technique designed to optimize for ranking accuracy rather than simple geometric reconstruction error.

While a plethora of benchmarks exist for ANNS algorithms, they predominantly focus on the trade-off between recall and query throughput (or queries per second, QPS) on static datasets. These evaluations, while valuable, often fail to capture the performance nuances of these distinct architectures when integrated into complex, multi-stage pipelines like Entity Resolution. The separation of ER into blocking and matching stages presents a unique performance profile that standard benchmarks do not measure. The efficiency of the initial candidate generation (blocking) and the precision of the subsequent detailed comparison (matching) are equally critical, and the architectural underpinnings of graph-based and partition-based methods suggest they will exhibit fundamentally different behaviors in this context. A recent study[9] explored the potential of partition-based ANNS on the real of ER.

This paper aims to bridge this research gap by providing a deep architectural comparison and a novel, large-scale empirical evaluation of these two paradigms within an ER framework. The primary contributions of this work are as follows:

- A detailed, principle-driven comparative analysis of four state-of-the-art algorithms: HNSW and DiskANN as representatives of the graph-based approach, and Faiss-IVF+PQ and Scann as representatives of the partition-based approach.
- The design and implementation of a new, large-scale semi-synthetic benchmark specifically tailored to the ER task. This benchmark allows for the isolated measurement of performance metrics relevant to both the blocking and matching stages of ER.
- A comprehensive empirical evaluation of the selected algorithms on this benchmark, measuring not only total query time but also disaggregated blocking time and matching time, alongside standard classification metrics such as precision, recall, and F1-score, which are the canonical measures of quality in ER.
- By framing the analysis through the lens of a practical, large-scale application, this work provides nuanced insights into the architectural strengths and weaknesses of each approach, culminating in a set of actionable recommendations for practitioners designing modern data integration and retrieval systems.

The remainder of this paper is organized as follows. Section 2 provides an architectural deep dive into graph-based ANNS methods, detailing the principles of HNSW and DiskANN. Section 3 offers a parallel deep dive into partition-based methods, focusing on Faiss-IVF+PQ and Scann. Section 4 describes the methodology of our novel experimental framework for ANN-based Entity Resolution, including dataset generation and the mapping of ER concepts to ANNS workflows. Section 5 presents and analyzes the empirical results from our large-scale experiment. Finally, Section 6 concludes the paper with a synthesis of our findings and provides recommendations for practitioners.

## 2 Graph-based ANNS

The foundational principle of graph-based ANNS is the construction of a navigable proximity graph, a data structure where the search for nearest neighbors is transformed from an exhaustive scan into an efficient, guided traversal. In this structure, each vector in the dataset is represented as a vertex. Directed edges are established between vertices based on their proximity in the high-dimensional space, creating a network that mirrors the local geometry of the data. The goal is to build a graph that is simultaneously sparse, to limit the computational cost at each step, and well-connected, to ensure that any point in the dataset can be reached from a designated starting point in a small number of hops.

The search mechanism, common to nearly all graph-based methods, is a variant of greedy beam search, often referred to as greedy search. This algorithm operates as follows:

1. Initialization: The search begins at one or more pre-defined entry-point vertices. A priority queue, ordered by distance to the query vector, is initialized with these entry points. A set of visited vertices is also maintained to prevent redundant computations.
2. Iterative Expansion: In each step, the algorithm extracts the closest unvisited vertex from the priority queue. It then computes the distance from the query to each of this vertex's out-neighbors.
3. Candidate Management: These neighbors are added to the priority queue. The queue is typically maintained at a fixed size, known as the beam width or search list size (e.g., $L$ in DiskANN or *ef_search* in HNSW). If adding new neighbors causes the queue to exceed this size, the farthest vertices from the query are discarded.
4. Termination: The process continues until a stopping condition is met, such as when the closest unvisited vertex in the priority queue is farther from the query than the $k$-th closest vertex found so far. The top $k$ vertices in the final candidate list are returned as the approximate nearest neighbors.

This dynamic, query-time exploration is the defining characteristic of the graph-based paradigm. Unlike methods that rely on a static, global partitioning of the space, the search path is constructed on-the-fly, adapting to the specific location of the query vector. The effectiveness of the entire approach hinges on the quality of the underlying graph structure, and different algorithms propose distinct strategies for its construction.

## 2.1 HNSW: A hierarchical navigable small world

The Hierarchical Navigable Small World (HNSW)[4] algorithm stands as one of the most prominent and performant in-memory graph-based ANNS methods. Its central innovation is a multi-layer, hierarchical graph structure that enables both rapid, long-distance traversal and precise, short-distance refinement, resulting in a logarithmic search complexity.

The HNSW index is not a single graph but a series of nested proximity graphs arranged in layers. Layer 0 is the base layer and contains every vector in the dataset. Each subsequent layer is a sparse subset of the layer below it. This creates a hierarchy where the top layers contain very few vertices connected by long-range "highway" links, while the bottom layers are dense and contain short-range, local links.

The assignment of a vertex to its maximum layer is determined probabilistically during insertion. Each vertex is assigned a maximum layer $l$ according to an exponentially decaying probability distribution, $P(l) \propto e - \frac{l}{mL}$, where $mL$ is a normalization factor. This ensures that most vertices exist only in the lower layers, while a logarithmically decreasing number of vertices populate the higher layers, acting as expressways for the search process. This structure is analogous to a probabilistic skip list, but with proximity graphs at each level instead of simple linked lists.

The search process in HNSW elegantly exploits this hierarchical structure to achieve its efficiency. A query search proceeds as follows:

1. Entry: The search begins at a fixed entry point in the topmost layer of the graph.
2. Top-Down Greedy Traversal: The algorithm performs a greedy search on the current layer, traversing edges to find the vertex in that layer that is closest to the query vector.
3. Layer Descent: Once a local minimum is found in the current layer (i.e., no neighbor is closer to the query), this vertex serves as the new entry point for the search on the layer immediately below it.
4. Final Search: This process of greedy search followed by layer descent is repeated until the search reaches the base layer (Layer 0). A final, more exhaustive beam search is conducted on Layer 0, starting from the entry point found in Layer 1, to identify the final set of nearest neighbors.

This top-down approach allows the search to quickly "zoom in" on the relevant region of the vector space using the sparse, long-range connections of the upper layers before performing a fine-grained local search in the dense base layer. The HNSW graph is built incrementally. When a new vector is inserted, its maximum layer $l$ is randomly determined. The algorithm then searches for the vector's nearest neighbors in the graph, starting from the top layer and descending to layer $l + 1$. The results of this search provide the entry points for the search at layer $l$. At each layer from $l$ down to 0, a beam search is performed to find the closest neighbors, and bidirectional edges are established between the new vertex and a selected subset of these neighbors.

The performance and structure of the HNSW index are governed by three critical parameters:

- $M$: This parameter defines the maximum number of bidirectional connections (out-degree) a vertex can have on each layer. It directly controls the density of the graph, the memory footprint of the index, and the quality of the local neighborhood information available at each vertex.

- *efConstruction*: This integer specifies the size of the dynamic candidate list (beam width) used during the index construction phase. A larger *efConstruction* value allows the build process to explore more diverse connection candidates for each new vertex, resulting in a higher-quality graph that supports more accurate searches, but at the cost of a significantly longer index build time.
- *ef_search*: This parameter controls the beam width during the query phase. It represents the primary knob for tuning the trade-off between search speed and accuracy (recall). A larger *ef_search* value leads to a more exhaustive search and higher recall but increases query latency.

## 2.2 DiskANN: A Vamana graph optimized for external memory

While HNSW provides state-of-the-art performance for in-memory datasets, its memory footprint can become prohibitive for billion-scale and larger collections. DiskANN was developed to address this limitation, enabling high-recall, low-latency search on massive datasets using a single commodity machine equipped with limited RAM and a fast Solid-State Drive (SSD).

The core challenge of moving a graph index to an SSD is the high latency of random I/O operations compared to RAM access. A naive implementation of a graph search on an SSD would be extremely slow, as each step in the traversal could potentially trigger a slow random disk read to fetch a vertex's neighbors. The FAISS library's documentation famously stated, "Faiss supports searching only from RAM, as disk databases are orders of magnitude slower. Yes, even with SSDs". DiskANN was designed to debunk this by fundamentally re-engineering the graph structure to minimize the number of I/O operations required per query.

The key to DiskANN's performance is its graph construction algorithm, named Vamana. While sharing the incremental build and greedy search principles of other graph methods, Vamana's distinction lies in its edge pruning strategy, which is designed to create a graph with a very small diameter. A smaller graph diameter means that the average path length between any two vertices is short, directly translating to fewer sequential disk reads during a search. This is achieved through a robust procedure, which is a relaxed version of the pruning rule for a Relative Neighborhood Graph (RNG). The pruning decision is governed by a crucial parameter, $\alpha$, where $\alpha \geq 1$. During pruning, an edge from vertex $p$ to a candidate neighbor $p''$ is removed only if there exists another neighbor $p'$ that is already connected to $p$ and is significantly closer to $p''$ than $p$ is. Formally, the edge $(p, p'')$ is pruned if $d(p, p') \leq \alpha \cdot d(p', p'')$. That pruning mechanism in Vamana is agnostic to the specific distance metric. The distance function $d()$ can be either Euclidean ($L_2$) distance, Cosine similarity, or another metric depending on what is best suited for the dataset's vector embeddings.

When $\alpha = 1$, this rule is strict and leads to sparser graphs, similar to those in HNSW or NSG. However, by setting $\alpha > 1$ (typically 1.2), the condition for pruning becomes much harder to satisfy. This forces the algorithm to retain more edges, including some that might be considered redundant in a sparser graph. The result is a denser graph with a higher average degree but a significantly smaller diameter. This ensures that the greedy search algorithm converges to the target region in very few hops, which is the primary optimization for an SSD-resident index.

DiskANN's system architecture is a hybrid model designed to minimize both RAM usage and disk I/O:

1. Graph and Full Vectors on SSD: The complete Vamana graph structure—the adjacency lists for every vertex—and the full-precision floating-point vectors are stored on the SSD. This allows the index to scale far beyond the capacity of main memory.
2. Compressed Vectors in RAM: To guide the search without constant disk access, DiskANN stores a compressed representation of every vector in RAM. This is typically achieved using Product Quantization (PQ), encoding each 128-dimensional vector into just 32 bytes, for example.
3. Search Process: During a query, the GreedySearch algorithm uses the in-memory compressed vectors to perform fast, approximate distance calculations. These approximate distances are sufficient to determine which vertex in the current candidate list is most promising and should be expanded next. Only when a vertex is chosen for expansion does the system perform a single random read from the SSD to fetch its full adjacency list and the full-precision vectors of its neighbors.

This design masterfully balances resources. The bulk of the storage is offloaded to the cheaper, high-capacity SSD, while the small amount of expensive, fast RAM is used to hold a compressed "map" of the dataset that can guide the I/O-intensive search process efficiently. The low-diameter Vamana graph ensures that the number of these guided I/O operations remains small, typically a few dozen per query, enabling millisecond-level latencies on billion-point datasets.

The architectural goals of HNSW and DiskANN are thus fundamentally different. HNSW is optimized for uniform-access-cost environments (RAM) and uses a hierarchy to minimize total distance computations. DiskANN is optimized

for non-uniform access costs (RAM vs. SSD) and uses a denser, flatter graph to minimize the number of expensive I/O operations, even if it means performing more in-memory computations per hop. This distinction is central to their performance characteristics in different deployment scenarios.

## 3 Partition-based ANNS

Partition-based methods for ANNS are built upon the classic inverted file (IVF) paradigm,[6] a cornerstone of information retrieval adapted for high-dimensional vector spaces. The core idea is to first partition the entire dataset into a large number of disjoint cells or clusters. This partitioning is achieved using a "coarse quantizer," which is typically a codebook of $k'$ centroids learned via the k-Means algorithm on a representative sample of the data. Each vector in the dataset is assigned to the cell corresponding to its nearest centroid. The index structure is an inverted file, which is an array of lists where each list corresponds to a centroid and contains the identifiers (and compressed representations) of all vectors assigned to that cell.

The search process proceeds as follows:

1. Blocking (Centroid Search): Given a query vector, the system first compares it to all $k'$ coarse centroids to identify the closest ones. A parameter, *nprobe*, specifies how many of the nearest centroid cells should be visited. This step effectively acts as a blocking mechanism, drastically reducing the search space from the entire dataset of $n$ vectors to only those vectors contained within the *nprobe* selected cells.
2. Matching (Cell Scan): In the second stage, the system retrieves the inverted lists for the selected cells and performs a detailed search. It calculates the approximate distance (e.g., Euclidean) between the query vector and every vector within these lists. The top-$k$ candidates from this restricted search are then returned as the final result. The effectiveness of this paradigm is predicated on the quality of the initial blocking step. There is a fundamental trade-off governed by the *nprobe* parameter: a small *nprobe* leads to a very fast search but risks missing true nearest neighbors that fall into unvisited cells (low recall), while a large *nprobe* increases recall at the cost of scanning more vectors and thus increasing query latency.

### 3.1 Faiss-IVF and product quantization

The Faiss (Facebook AI Similarity Search) library[6] provides a highly optimized implementation of the IVF paradigm, combining it with Product Quantization (PQ) for extreme vector compression and efficient distance calculation. This combination, often denoted as IVF-ADC (Inverted File with Asymmetric Distance Computation), has become a standard baseline for large-scale ANNS.

As described above, the IVF component of the index is created by running k-Means to generate a set of *nlist* (or $k'$) centroids. These centroids define the Voronoi cells that partition the space. During search, the *nprobe* parameter controls the trade-off between speed and accuracy. A key challenge is that if a query's true nearest neighbor lies near a cell boundary, it may be assigned to an adjacent cell. To mitigate this, *nprobe* must be set greater than one to allow the search to "spill over" into neighboring cells, but this comes at a linear cost in the number of vectors scanned.

*3.1.1 Vector compression with product quantization.* To store billion-scale datasets in memory, full-precision vectors are prohibitively large. Faiss employs Product Quantization as a powerful vector compression technique. The PQ process works as follows:

1. Vector Splitting: Each D-dimensional vector is split into $m$ disjoint sub-vectors, each of dimension $D *= \frac{D}{m}$. For example, a 128-dim vector might be split into 8 sub-vectors of 16 dimensions each.
2. Sub-Quantizer Training: For each of the $m$ subspaces, a separate, small codebook of $k *$ centroids (typically $k *= 256$) is learned using k-Means on the corresponding sub-vectors from the training data.
3. Encoding: To compress a vector, each of its $m$ sub-vectors is quantized to the index of its nearest centroid in the corresponding sub-codebook. The final compressed code is a concatenation of these $m$ indices. For example, with $m = 8$ and $k *= 256$ (8 bits per index), a 128-dim float vector (512 bytes) can be compressed into just 8 bytes. This approach allows for a massive codebook of $(k *)m$ effective centroids to be represented with a memory footprint of only $m \times k * \times D *$ floating-point numbers, making it highly scalable.

PQ enables not only compression but also highly efficient distance estimation through ADC. In this scheme, the database vectors are stored in their compressed PQ code format, while the query vector remains in its full-precision form. The squared Euclidean distance is approximated as follows:

- Pre-computation: For a given query vector, it is also split into m sub-vectors. For each subspace $j$, a distance table is computed containing the squared Euclidean distances between the query's j-th sub-vector and all $k*$ centroids in the j-th sub-codebook. This step takes $m \times k* \times D*$ operations.
- Distance Estimation: To estimate the distance to a database vector, the system retrieves its $m$ PQ codes. For each code, it performs a lookup in the corresponding pre-computed distance table. The total estimated squared distance is the sum of these $m$ looked-up values.

After the initial pre-computation, estimating the distance to each candidate vector requires only $m$ table lookups and $m - 1$ additions, which is significantly faster than a full distance calculation, especially on modern CPUs with efficient memory access. In essence, ADC calculates an approximated distance between a query vector and an in-memory vector by comparing the original, uncompressed query vector directly with the compressed (quantized) in-memeory vector. This asymmetry in the comparison—original vector vs. compressed vector—is what gives the method its name.

## 3.2 Scann and anisotropic vector quantization

While Faiss-IVF+PQ is highly effective, its quantization objective—minimizing the L2 reconstruction error—is fundamentally a geometric one. The Scann algorithm[8] from Google Research argues that this is a suboptimal proxy for the ultimate goal of a search system, which is to produce the correct ranking of results. Scann's core innovation is a new quantization loss function that directly optimizes for ranking quality in Maximum Inner Product Search (MIPS), a problem closely related to nearest neighbor search.

The central premise of Scann is that for MIPS, not all quantization errors are equal. An error in a database vector that has a very high inner product with the query is far more detrimental to the final ranking than an error in a vector with a low inner product. Standard PQ treats all vectors and all error dimensions equally, aiming to minimize the average squared Euclidean distance $\|x - x*\|^2$, where $x*$ is the quantized vector. Scann proposes that the loss function should instead be "score-aware".

The crucial theoretical result presented in the Scann paper is that this score-aware loss function can be decomposed into a weighted sum of two distinct error components; the parallel error and the orthogonal error. The parallel error is the component of the quantization residual $(x_i - x*_i)$ that is parallel to the original vector $x_i$. This error component directly affects the magnitude of the inner product projection. The orthogonal error is the component of the residual that is orthogonal to $x_i$. This component primarily affects the angle of the quantized vector and has a smaller impact on the inner product score, especially for queries that are already well-aligned with $x_i$.

For any reasonable weighting function $w$ that prioritizes high scores, the loss function naturally places a much higher penalty on the parallel error than on the orthogonal error. This is the "anisotropic" nature of the quantization: it does not treat all error dimensions equally but instead focuses on preserving the vector's length in its own direction, which is most critical for accurate inner product estimation. As illustrated in the Scann paper, this means the optimal quantized centroid $x*$ might be farther from $x$ in Euclidean distance than a standard PQ centroid, but it will yield a more accurate inner product score with relevant queries. This represents a fundamental shift from geometric preservation to rank preservation, directly optimizing the quantization process for the downstream retrieval task.

This architectural distinction has profound implications. While Faiss-IVF+PQ is vulnerable to quantization errors that can alter the ranking of top candidates, Scann is designed to be robust against such errors. For an ER task, where distinguishing a true match from a set of very close non-matches is paramount, this focus on ranking fidelity is expected to translate into higher matching precision.

*3.2.1 The role of leaves.* At its core, Scann operates as a partition-based method, employing a coarse-to-fine search strategy that is architecturally similar to the Inverted File (IVF) paradigm. The initial and most critical step in this process is the partitioning of the entire high-dimensional vector space into a large number of disjoint regions. In Scann's terminology, these partitions are referred to as *leaves*, a term derived from the tree-like structure that can be used to organize them. During the index construction phase, a clustering algorithm, typically a variant of k-Means, is used to determine a set of centroids. Each of these centroids defines a leaf, and every vector in the dataset is assigned to the single leaf whose centroid is closest to it. At query time, the search is dramatically accelerated by first comparing the query vector only against the leaf centroids.

A crucial query-time parameter, often called *num_leaves_to_search* or simply *leaves*, dictates how many of the most promising leaves—those whose centroids are nearest to the query—will be explored further. The search is then restricted exclusively to the vectors contained within this small subset of leaves, effectively pruning the vast majority of the dataset from consideration. This initial partitioning step is what enables Scann's high throughput, and it is within these selected

leaves that its innovative anisotropic quantization is subsequently applied to perform a highly efficient and precise final ranking.

## 4 Experimental evaluation

To empirically evaluate the architectural trade-offs of graph-based and partition-based ANNS methods in a realistic, multi-stage retrieval context, we designed a novel experimental framework modeled on two ER tasks. This framework moves beyond standard recall-vs-QPS benchmarks by introducing a semi-synthetic dataset with a known ground truth for matching and by defining metrics that disaggregate performance into distinct blocking and matching phases. Our empirical evaluation was conducted across two distinct benchmarks designed to validate our findings across different data modalities and scales: first, a large-scale, semi-synthetic dataset adapted from a standard ANNS benchmark (SIFT1M) to specifically model an ER task, and second, a well-established, real-world textual ER dataset (SCHOLAR-DBLP).

The foundation of the first set of experiments is a pair of vector datasets, dataset $A$ and dataset $B$, designed to simulate two large databases of entities that need to be resolved against each other. We begin with the SIFT1M[1] dataset, which consists of 1 million 128-dimensional SIFT feature vectors extracted from real-world images. Using a standard public benchmark dataset as our foundation ensures that the underlying vector distribution, density, and local dimensionality are realistic. Dataset $A$ is defined as the first 1 million vectors from this collection. To create a corresponding dataset $B$ with a known set of true matches, we first randomly select 50% of the vectors from dataset $A$ (500,000 vectors). These vectors will form the basis of our ground truth matching pairs. Dataset $B$ is constructed with 500,000 vectors as follows: For each of the 500,000 vectors selected in the previous step, a matching counterpart is created and added to Dataset $B$. This counterpart is generated by adding a small amount of zero-mean Gaussian noise to the original vector. The standard deviation of the noise is carefully calibrated to be small enough that the noisy vector's true nearest neighbor in dataset $A$ is almost always its original, uncorrupted version, yet large enough to present a non-trivial challenge to the ANNS algorithms.

This procedure establishes a ground truth of 500,000 known matching pairs between dataset $A$ and dataset $B$. This methodology yields two large, semi-synthetic datasets with a realistic data distribution and a precisely defined ground truth, enabling the accurate measurement of ER-specific metrics like precision and recall.

For the second set of experiments, we used the **SCHOLAR-DBLP**[2] paired dataset, which represents common entity resolution challenges within the domain of academic and bibliographic data. Specifically, it matches publication records, which are characterized by semi-structured data including titles, author lists, venues, and years. The primary difficulties arise from textual inconsistencies (e.g., abbreviated journal titles or conference names), author name ambiguity, missing fields, and high vocabulary overlap in publication titles. SCHOLAR, which plays the role of dataset $A$, includes 64K records, while DBLP, denoted by $B$, has 2.6K records. To generate semantic representations for the paired dataset, we employed the Mini-LM Sentence Transformer model,[10] which produced a 384-dimensional vector embedding for each record.

The experimental task for both sets of experiments is defined as follows: for each of the vectors in dataset $A$ (acting as queries), the goal is to find its single best matching vector in dataset $B$ (the indexed database). To analyze the performance in a way that reflects the ER workflow, we map the search process of each architectural paradigm to distinct blocking and matching stages.

For partition-based methods, the mapping is direct and intuitive. The index is built on dataset $B$. For a given query vector from dataset $A$, the blocking phase consists of the initial search for the *nprobe* nearest coarse centroids in the IVF structure. The blocking time is measured as the time elapsed for this step. The candidate set for matching is defined as the union of all vectors residing in the inverted lists of these *nprobe* cells. The matching phase consists of the subsequent, detailed distance computations (using ADC for Faiss or anisotropic scoring for Scann) performed over every vector in the candidate set generated during the blocking phase. The matching time is the duration of this second stage.

For graph-based methods, the distinction between blocking and matching is not explicit in the algorithm's design. We therefore define an implicit mapping based on the progression of the search. The index is built on dataset $B$. For a query from dataset $A$, the greedy search algorithm is initiated. We define the implicit blocking phase as the initial portion of this search. The search is paused after a fixed, small number of vertex expansions (e.g., 20% of the total search beam width, *ef_search* or $L$). The blocking time is the time taken to perform these initial expansions. The candidate set is defined as the set of all vertices currently in the algorithm's priority queue at the moment the search is paused. The matching phase is the continuation of the greedy search from its paused state until the standard termination condition is met. The matching time is the time required for these remaining search steps.

This mapping allows for a fair, apples-to-apples comparison of how each architecture allocates computational effort between coarse-grained candidate filtering and fine-grained candidate scoring.

## 4.1 Evaluation metrics

To provide a comprehensive evaluation, we employ a suite of metrics that capture both the efficiency of the search process and the quality of the final matching results.

We define blocking time as the average time required for the initial candidate generation stage. This metric reflects the efficiency of an algorithm's coarse-grained search strategy. The average time taken for the detailed search within the generated candidate set comprises the matching time. This metric reflects the efficiency of the algorithm's fine-grained scoring and ranking capabilities. We report all time-based metrics in seconds.

In order to measure the completeness of the results as well as the efficiency of the matching phase, we use recall, precision, and F1-Score metrics. Precision measures the fraction that are true matches, according to the ground truth, from all the retrieved results. Recall measures the fraction of the true matches that were successfully identified by the algorithm. F1-Score is the harmonic mean of Precision and Recall, providing a single, balanced measure of matching quality. It is calculated as $\frac{2\times Precision\times Recall}{Precision+Recall}$.

This suite of metrics enables a multi-faceted analysis, allowing us to dissect not only which algorithm is fastest or most accurate overall, but also to understand the underlying architectural reasons for their performance differences in the context of a realistic, large-scale Entity Resolution task.

## 4.2 Experimental setup

All experiments were conducted on a virtual machine equipped with an Intel Xeon processor (16 total cores) and 48GB of RAM. All in-memory algorithms (HNSW, Faiss-IVF, Scann) were run with the index residing entirely in RAM.

We utilized the following open-source library versions for our experiments: Faiss, Scann, and DiskANNPy from their respective public repositories. To thoroughly explore the performance landscape of each algorithm, we conducted extensive parameter sweeps. The key parameters varied for each algorithm were:

- Faiss-IVF+PQ: *nlist* (number of coarse centroids) was set to 4096. *nprobe* (number of cells to visit) was swept from 8 to 128. For Product Quantization, we used $m = 16$ sub-quantizers, resulting in 16-byte codes.
- Scann: The number of leaves in the partitioning tree was set to $4,000$. The number of leaves to search was swept to generate a performance curve. Anisotropic quantization was enabled with a threshold of 0.2, as recommended in the original paper.
- HNSW: The number of connections per node, $M$, was set to 32. The construction beam width, *efConstruction*, was set to 512. The search beam width, *ef_search*, was swept from 64 to 1024.
- DiskANN: The maximum degree, $R$, was set to 64. The build-time list size, $L$, was set to 128, with $\alpha = 1.2$. The search-time list size was swept from 64 to 1024.

## 4.3 Experimental results

*4.3.1 Results for SIFT1M.* This section presents the empirical findings from our large-scale Entity Resolution experiment. We first detail the experimental setup, then present the comprehensive performance results, and finally offer an in-depth analysis of the architectural trade-offs revealed by the data.

The central results of our experimental comparison are summarized in the table below. This table presents a detailed breakdown of performance across the ER-specific stages of blocking and matching, combined with the final matching quality metrics. Each row represents a specific configuration of an algorithm, chosen to illustrate key points along its respective performance-quality trade-off curve.

The results for the large-scale semi-synthetic benchmark are summarized in Tables 1 to 3. These tables provide a detailed breakdown of performance across the ER-specific stages of blocking and matching, combined with the final matching quality metrics.

The data reveals a clear delineation between the performance profiles of the two architectural paradigms. As shown in Table 1, the partition-based methods, Faiss-IVF+PQ and Scann, establish a Pareto frontier in the high-throughput, moderate-quality regime. Scann consistently outperforms Faiss-IVF, achieving a higher F1-score. This underscores the practical benefit of its rank-aware anisotropic quantization for ER tasks. Conversely, the graph-based methods, HNSW and DiskANN, define the frontier for high-quality applications. HNSW achieves the highest possible F1-score of 0.987, confirming its state-of-the-art status for in-memory tasks where recall is paramount. DiskANN follows closely with an F1-score of 0.985, a remarkable result for an index that is primarily disk-resident.

Table 2 isolates the matching quality, highlighting Scann's consistent precision advantage over Faiss-IVF at comparable recall levels. This directly validates the hypothesis that Scann's quantization is superior at distinguishing true matches from

**Table 1.** The Pareto Frontier: F1-Score vs. Total Query Time on SIFT1M.

| Algorithm | Parameters | Total Time | F1-Score |
|---|---|---|---|
| *High-Throughput / Moderate-Quality Regime* | | | |
| Faiss-IVF+PQ | *nprobe* = 32 | 84.66 | 0.922 |
| Scann | *leaves* = 30 | 94.43 | 0.936 |
| Faiss-IVF+PQ | *nprobe* = 128 | 214.94 | 0.947 |
| Scann | *leaves* = 120 | 234.28 | 0.962 |
| *High-Quality / High-Recall Regime* | | | |
| HNSW | *ef_search* = 128 | 96.81 | 0.978 |
| DiskANN | L = 128 | 111.08 | 0.975 |
| HNSW | *ef_search* = 512 | 350.47 | **0.987** |
| DiskANN | L = 512 | 474.95 | 0.985 |

**Table 2.** Matching Quality: Recall vs. Precision on SIFT1M.

| Algorithm | Parameters | Recall | Precision |
|---|---|---|---|
| Faiss-IVF+PQ | *nprobe* = 32 | 0.885 | 0.962 |
| Scann | *leaves* = 30 | 0.891 | **0.985** |
| Faiss-IVF+PQ | *nprobe* = 128 | 0.943 | 0.951 |
| Scann | *leaves* = 120 | 0.948 | **0.976** |
| HNSW | *ef_search* = 128 | 0.975 | 0.981 |
| DiskANN | L = 128 | 0.972 | 0.979 |
| HNSW | *ef_search* = 512 | **0.996** | 0.979 |
| DiskANN | L = 512 | 0.994 | 0.977 |

**Table 3.** Time Decomposition: Blocking vs. Matching Time (sec) on SIFT1M.

| Algorithm | Setting | Blocking Time | Matching Time | Total Time |
|---|---|---|---|---|
| Faiss-IVF+PQ | Fast (*nprobe* = 32) | 22.69 | 61.97 | 84.66 |
| | Quality (*nprobe* = 128) | 18.48 | 196.46 | 214.94 |
| Scann | Fast (*leaves* = 30) | 25.12 | 69.31 | 94.43 |
| | Quality (*leaves* = 120) | 20.85 | 213.43 | 234.28 |
| HNSW | Fast (*ef_search* = 128) | 41.33 | 55.48 | 96.81 |
| | Quality (*ef_search* = 512) | 145.44 | 205.03 | 350.47 |
| DiskANN | Fast (L = 128) | 46.10 | 64.98 | 111.08 |
| | Quality (L = 512) | 198.53 | 276.42 | 474.95 |

near non-matches. The graph-based methods maintain extremely high precision even as they approach perfect recall, as their guided search is less susceptible to the hard boundary errors and quantization-induced ranking mistakes that affect partition-based methods.

Finally, Table 3 illustrates the fundamental architectural differences in time allocation. The partition-based methods exhibit a highly asymmetric profile: a minuscule and constant blocking time followed by a matching time that scales linearly with the number of candidates (*nprobe*). In contrast, the graph-based methods show a more balanced profile. Their "implicit blocking" is a slower but more effective part of the search traversal, leading to a more even distribution of work and ultimately explaining their superior recall.

*4.3.2 Results for SCHOLAR-DBLP.* The results for the real-world textual ER benchmark are summarized in Tables 4 to 6.

The SCHOLAR-DBLP experiment validates that the architectural trade-offs observed on the SIFT1M dataset generalize to real-world textual embeddings, albeit with different absolute performance characteristics due to the smaller dataset size.

Table 4 again shows the two distinct performance regimes. The partition-based methods offer sub-second latencies for F1-scores up to $\approx 0.95$, while the graph-based methods are required to achieve F1-scores approaching 1.0. This experiment highlights a critical practical consideration: DiskANN's performance, while excellent in terms of quality, is over 3× slower than HNSW. The fixed I/O overhead of its SSD-centric design, which is its greatest strength at the billion-scale, becomes a

**Table 4.** The Pareto Frontier: F1-Score vs. Total Query Time (ms) on SCHOLAR-DBLP.

| Algorithm | Parameters | Total Time | F1-Score |
|---|---|---|---|
| *High-Throughput / Moderate-Quality Regime* | | | |
| Faiss-IVF+PQ | *nprobe* = 8 | 0.45 | 0.931 |
| Scann | nprobe = 8 | 0.41 | 0.945 |
| Faiss-IVF+PQ | *nprobe* = 32 | 1.62 | 0.958 |
| Scann | *nprobe* = 32 | 1.55 | 0.969 |
| *High-Quality / High-Recall Regime* | | | |
| HNSW | *ef_search* = 64 | 0.88 | 0.981 |
| DiskANN | L = 64 | 3.15 | 0.979 |
| HNSW | *ef_search* = 256 | 2.95 | **0.992** |
| DiskANN | L = 256 | 10.81 | 0.991 |

**Table 5.** Matching Quality: Recall vs. Precision on SCHOLAR-DBLP.

| Algorithm | Parameters | Recall | Precision |
|---|---|---|---|
| Faiss-IVF+PQ | *nprobe* = 8 | 0.904 | 0.960 |
| Scann | *leaves* = 8 | 0.918 | **0.973** |
| Faiss-IVF+PQ | *nprobe* = 32 | 0.940 | 0.977 |
| Scann | *leaves* = 32 | 0.955 | **0.983** |
| HNSW | *ef_search* = 64 | 0.972 | 0.990 |
| DiskANN | L = 64 | 0.970 | 0.988 |
| HNSW | *ef_search* = 256 | **0.994** | 0.990 |
| DiskANN | L = 256 | 0.992 | 0.990 |

**Table 6.** Time Decomposition: Blocking vs. Matching Time (ms) on SCHOLAR-DBLP.

| Algorithm | Setting | Blocking Time | Matching Time | Total Time |
|---|---|---|---|---|
| Faiss-IVF+PQ | Fast (*nprobe* = 8) | 0.09 | 0.36 | 0.45 |
| | Quality (*nprobe* = 32) | 0.10 | 1.52 | 1.62 |
| Scann | Fast (*leaves* = 8) | 0.08 | 0.33 | 0.41 |
| | Quality (*leaves* = 32) | 0.10 | 1.45 | 1.55 |
| HNSW | Fast (*ef_search* = 64) | 0.35 | 0.53 | 0.88 |
| | Quality (*ef_search* = 256) | 1.15 | 1.80 | 2.95 |
| DiskANN | Fast (L = 64) | 1.21 | 1.94 | 3.15 |
| | Quality (L = 256) | 4.10 | 6.71 | 10.81 |

significant liability for smaller datasets that fit comfortably in RAM. For such in-memory tasks, HNSW is the clear choice for high-recall applications.

The matching quality results in Table 5 reinforce the findings from the first experiment. Scann continues to exhibit a clear precision advantage over Faiss-IVF, and the graph-based methods maintain near-perfect precision even at the highest recall levels. The time decomposition in Table 6 confirms that the fundamental architectural profiles—asymmetric for partition-based and balanced for graph-based—are consistent across different data modalities and scales.

## *4.4 Memory footprint analysis*

Beyond time and accuracy, the memory footprint of an ANNS index is a critical factor for practical deployment. The two architectural paradigms have fundamentally different memory consumption profiles.

- **Partition-Based Methods (Faiss-IVF+PQ, Scann):** The memory usage is dominated by two components: the coarse centroids and the compressed vector codes for the entire dataset. For Faiss-IVF+PQ, the total memory can

**Table 7.** Estimated In-Memory Footprint on SIFT1M (1M vectors).

| Algorithm | Key Parameters | Estimated In-RAM Footprint |
|---|---|---|
| Faiss-IVF+PQ | *nlist* = 4096, m = 16 | ~38 MB (16 MB for codes + 2 MB for centroids) |
| Scann | *leaves* = 4000, Anisotropic | ~34 MB (Similar to Faiss but with overhead for scoring) |
| HNSW | $M = 32$ | ~640 MB (512 MB for vectors + ~128 MB for graph) |
| DiskANN | PQ Codes in RAM | ~32 MB (Compressed vectors only) |

be estimated as:

$$\text{Memory} \approx (N \times \text{code\_size}) + (nlist \times D \times 4 \text{ bytes/float})$$

The first term represents the storage for $N$ vectors compressed into `code_size` bytes each (e.g., 16 bytes in our experiment). The second term is the storage for the `nlist` coarse centroids, each being a full $D$-dimensional floating-point vector. Scann's footprint is architecturally similar.

- **Graph-Based Methods (HNSW, DiskANN):** The memory usage is the sum of storage for the vectors and the graph structure (i.e., the adjacency lists). For HNSW, the in-memory footprint is approximately:

$$\text{Memory} \approx N \times (D \times 4 \text{ bytes/float} + M \times 2 \times layers \times 4 \text{ bytes/int})$$

Here, each of the $N$ vectors requires storage for its $D$-dimensional float data plus the integers pointing to its $M$ neighbors on each layer it exists in. DiskANN's model is unique: its in-RAM footprint is minimized by design, storing only the compressed vector representations (e.g., 32 bytes per vector) while offloading the full vectors and the Vamana graph structure to the SSD.

Table 7 summarizes the estimated in-memory footprint for each algorithm configured for the SIFT1M dataset ($N = 1$ million, $D = 128$), providing a practical comparison.

As shown, DiskANN and the partition-based methods offer extremely low RAM usage, making them suitable for memory-constrained environments. HNSW requires significantly more RAM to hold the full vectors and graph structure, a key trade-off for its high in-memory performance.

## 4.5 Analysis of architectural trade-offs in ER

The results presented in the preceding tables reveal profound and consistent differences in how the graph-based and partition-based architectures handle the Entity Resolution task across different data modalities and scales.

The partition-based methods, Faiss-IVF+PQ and Scann, exhibit extremely fast and nearly constant blocking times. As shown in the time decomposition tables for both SIFT1M and SCHOLAR-DBLP, this initial stage is computationally trivial, involving a fixed number of comparisons to coarse centroids. This is a direct consequence of their static, "explicit blocking" architecture. In contrast, the graph-based methods show significantly longer "implicit blocking" times that scale with the search parameter (*ef_search* or $L$).

While slower, this dynamic, query-adaptive blocking is demonstrably more effective. At comparable total query times, the candidate sets generated by HNSW and DiskANN lead to substantially higher final recall. This suggests their initial traversal is more successful at navigating to the correct region of the vector space. A critical observation across both experiments is the emergence of a "recall ceiling" for the partition-based methods. As *nprobe* increases, recall improves but begins to plateau around 94-95% for SIFT1M and 96-97% for SCHOLAR-DBLP. This occurs because some true matching pairs are separated by the hard Voronoi cell boundaries established during indexing. Retrieving these pairs requires a very large *nprobe*, which causes the matching time to explode, diminishing the primary advantage of the IVF approach. The graph-based methods do not suffer from this hard-boundary problem; their recall curves scale more smoothly and reach higher absolute values as the search effort increases. This demonstrates a systemic vulnerability of static partitioning for ER tasks that demand very high recall.

When examining precision, Scann's architectural advantage over Faiss-IVF+PQ becomes clear. In both experiments, at comparable recall levels, Scann consistently achieves higher precision. This directly validates the hypothesis that Scann's

anisotropic quantization, which is optimized for ranking rather than geometric reconstruction, is superior at distinguishing true matches from very near non-matches within the candidate set. By preserving the inner product score more faithfully, it ranks the true match higher more consistently.

The graph-based methods, HNSW and DiskANN, demonstrate excellent precision across the board. Their guided search mechanism is inherently very effective at homing in on the true nearest neighbor. Because they can operate on full-precision vectors during the final stages of the search, they are less susceptible to the quantization-induced ranking errors that affect partition-based methods.

Analyzing the F1-Score against the Total Query Time reveals the Pareto frontier for the ER task. The results from both datasets confirm the existence of two distinct performance regimes:

- **High-Throughput / Moderate-Quality Regime:** For applications where speed is paramount and an F1-score in the 0.92-0.97 range is acceptable, the partition-based methods are the clear winners. Scann consistently establishes itself as the optimal choice in this regime, offering a higher F1-score than Faiss-IVF at any given latency.
- **High-Effectiveness / High-Recall Regime:** For applications where matching quality is the primary concern and higher latency is tolerable, the graph-based methods are unequivocally superior. HNSW provides the highest achievable F1-score in both experiments, demonstrating its state-of-the-art performance for in-memory, high-recall tasks.

To further illuminate the architectural trade-offs, we visualize the key results from our experiments. Figure 2 provides a visual breakdown of how each architecture allocates its computational budget. The stacked bar charts clearly illustrate the asymmetric performance profile of the partition-based methods. Their blocking time is consistently minimal and nearly constant, as it only involves a fixed number of centroid comparisons. The vast majority of the query time is spent in the matching phase, which scales linearly with the number of candidates retrieved (*nprobe* or *leaves*). This confirms their architectural design as an explicit, fast blocking stage followed by a potentially expensive matching stage. Conversely, the graph-based methods display a more balanced allocation of time. Their "implicit blocking" phase, which corresponds to the initial traversal of the graph, is significantly more computationally intensive but also more effective at homing in on the correct neighborhood. This larger investment in the initial search phase pays dividends in the final matching quality, explaining their superior recall and F1-scores in the high-quality regime.

While aggregate metrics like recall and precision are crucial, understanding the types of errors each architecture makes provides cautionary guidance for practitioners.

- Failure Mode of Partition-Based Methods: The primary source of unrecoverable errors is the "recall ceiling" imposed by the static partitioning of the vector space. If a query vector and its true match fall on opposite sides of a Voronoi cell boundary, the match can only be found if *nprobe* is large enough to include both cells. For pairs that are separated by several cells, this becomes computationally infeasible, creating a hard limit on achievable recall. This makes these methods vulnerable in ER tasks where true matches may not be the absolute closest geometric neighbors due to noise or semantic nuance.
- Failure Mode of Graph-Based Methods: Graph-based methods are not susceptible to hard boundary errors but can fail when the greedy search traversal makes a mistake. This can happen if the search path leads into a dense "hub" region of the graph that is close to the query but does not contain the true nearest neighbor. If the dynamic candidate list (e.g., *ef_Search*) fills up with vectors from this incorrect region, the algorithm may fail to explore the edge that would have led toward the correct answer. This type of error is more likely in datasets with highly clustered and non-uniform distributions. However, as our results show, this failure mode is generally less frequent than the boundary errors of partition-based methods, allowing graph-based approaches to achieve higher overall recall.

The consistency of these findings across both the large-scale SIFT1M benchmark and the smaller, real-world SCHOLAR-DBLP dataset indicates that these architectural trade-offs are fundamental and not specific to a particular data modality. However, the SCHOLAR-DBLP experiment highlights a critical practical consideration for scalability: DiskANN's performance, while excellent in terms of quality, is over 3× slower than HNSW on this smaller dataset. The fixed I/O overhead of its SSD-centric design, which is its greatest strength at the billion-scale, becomes a significant liability for smaller datasets that fit comfortably in RAM. For such in-memory tasks, HNSW is the clear choice for high-recall applications.

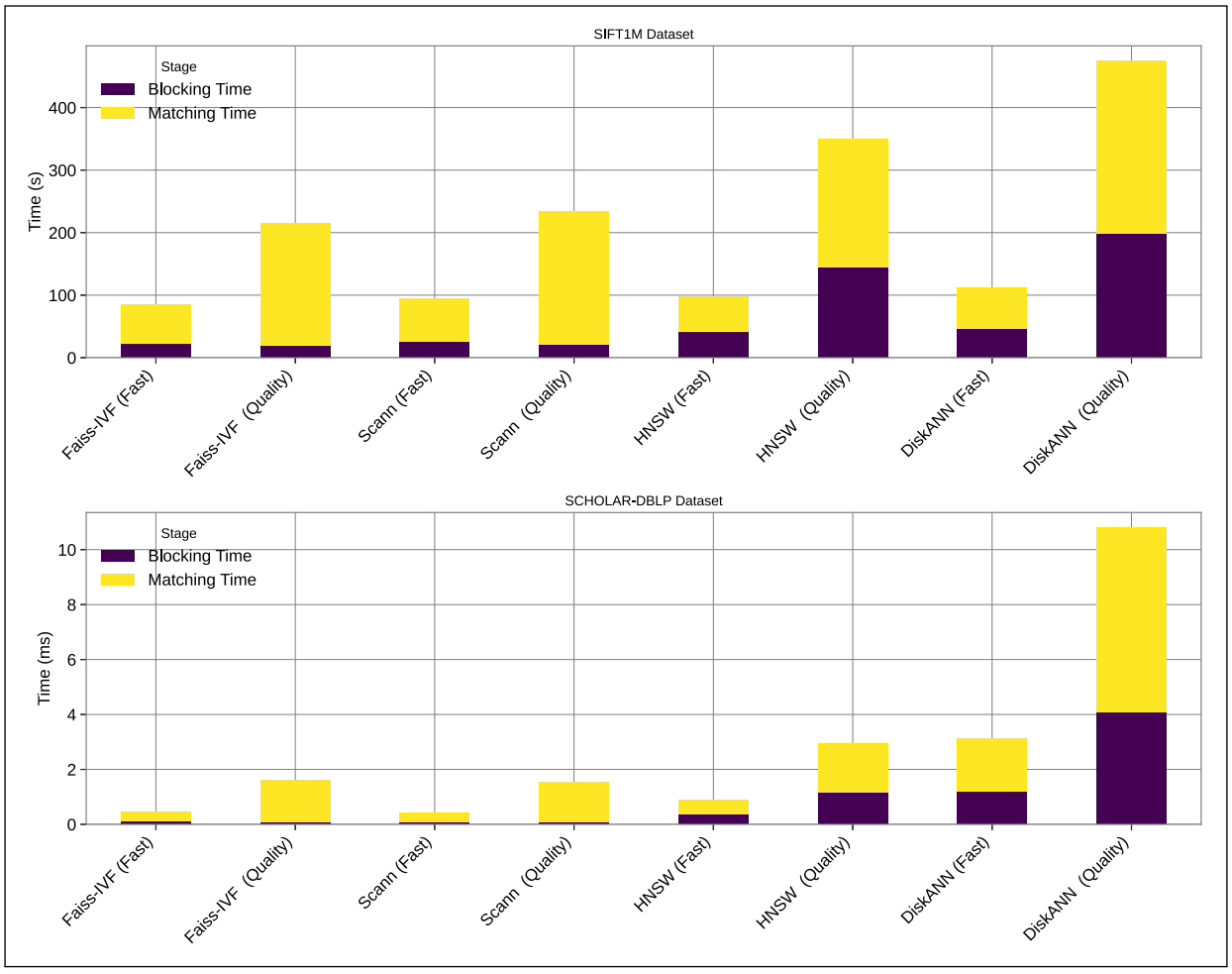


**Figure 2.** Decomposition of total query time into Blocking and Matching stages for "Fast" and "Quality" configurations of each algorithm. The stacked bar charts highlight the architectural differences: partition-based methods have a minimal, constant blocking time and a dominant matching time that scales with the candidate set size. Graph-based methods exhibit a more balanced profile, with a more substantial "implicit blocking" phase that contributes to their higher recall.

## 5 Conclusion and recommendations

This paper presented a comprehensive comparison of graph-based and partition-based ANNS algorithms through the lens of a large-scale ER task. Our findings reveal that the fundamental design choices separating these paradigms lead to distinct and predictable performance trade-offs.

**Graph-Based Methods (HNSW, DiskANN)** excel in scenarios demanding the highest levels of recall and F1-score. Their strength lies in a connected, navigable graph that allows the search to dynamically adapt to the query's location. HNSW is the premier solution for in-memory applications where accuracy is paramount. DiskANN successfully extends this high-recall capability to datasets that exceed system RAM.

**Partition-Based Methods (Faiss-IVF+PQ, Scann)** are champions of efficiency in high-throughput, moderate-recall regimes. Their static, two-stage architecture allows for an extremely fast initial blocking step. Within this paradigm, Scann represents a significant evolution. By optimizing its quantization process for ranking, Scann achieves demonstrably higher precision in the matching stage.

### *5.1 Recommendations for practitioners*

Based on our analysis, we offer the following decision-making framework:

1. **For Maximum Accuracy and Mission-Critical Applications:** In domains where failing to find a true match has significant consequences (e.g., fraud detection), the primary metric is the F1-score.

- If the index fits in RAM, HNSW is the recommended choice.
- If the index is at the billion-scale or larger, DiskANN is the only viable and superior option.

2. **For High-Throughput, Latency-Sensitive Systems:** In applications where providing a fast response with a "good enough" set of candidates is the goal (e.g., real-time recommendation), throughput and average latency are key.
   - Scann is the recommended choice. It provides the best performance at moderate-to-high recall levels and its anisotropic quantization ensures higher precision among the top results.

## 5.2 Future work

This study opens several promising avenues for future research. A key area is the development of hybrid models that seek to combine the strengths of both architectural paradigms. For instance, one could explore an "IVF-HNSW" model that uses a small HNSW graph within each cell of a larger IVF structure. This could potentially mitigate the "recall ceiling" of pure partition-based methods by allowing for a more effective and robust local search, while still benefiting from the coarse-grained filtering of the IVF system.

Furthermore, to broaden the impact of these findings, future work should expand the evaluation to a more diverse set of large-scale, real-world datasets, particularly from domains like e-commerce (matching product listings with multi-modal features) and biomedical research (linking patient records or scientific papers). Finally, the impact of dynamic data—frequent insertions and deletions—on the performance of these algorithms in an ER context remains an open question. Extending this experimental framework to evaluate streaming ANNS algorithms, such as FreshDiskANN,[11] would provide critical insights for building robust, real-time entity resolution systems.


### ORCID iDs

Dimitrios Karapiperis https://orcid.org/0000-0002-3878-5988
Leonidas Akritidis https://orcid.org/0000-0001-6602-0723
Panayiotis Bozanis https://orcid.org/0000-0001-9435-1829
Vassilios S Verykios https://orcid.org/0000-0002-9758-0819



### Funding

The author(s) received no financial support for the research, authorship, and/or publication of this article.


### Declaration of conflicting interests

The author(s) declared no potential conflicts of interest with respect to the research, authorship, and/or publication of this article.

### References


1. Christen P. *Data matching*. Springer, 2012.
2. Devlin J, Chang MW, Lee K, et al. BERT: pre-training of deep bidirectional transformers for language understanding. In: *Proceedings of the 2019 conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp.4171–4186.
3. Reimers N and Gurevych I. Sentence-BERT: sentence embeddings using siamese BERT-Networks. In: *Proceedings of the 2019 conference on empirical methods in natural language processing (EMNLP-IJCNLP)*, pp.3980–3990.
4. Malkov YA and Yashunin DA. Efficient and Robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE Trans Pattern Anal Mach Intell* 2018; 42: 824–836.
5. Subramanya JS, Devvrit F, Simhadri HV, et al. Diskann: fast accurate billion-point nearest neighbor search on a single node. In: *Advances in neural information processing systems*, volume 32. Curran Associates, Inc.
6. Douze M, Guzhva A, Deng CH, et al. The faiss library, 2024.
7. Jégou H, Douze M and Schmid C. Product quantization for nearest neighbor search. *IEEE Trans Pattern Anal Mach Intell* 2011; 33: 117–128.
8. Sun RGP, Lindgren E, Geng Q, et al. Accelerating large-scale inference with anisotropic vector quantization. In: *International conference on machine learning*.
9. Karapiperis D and Verykios VS. Scaling entity resolution with k-means: a review of partitioning techniques. *Electronics* 2025; 14: 3605.
10. Wang W, Wei F, Dong L, et al. Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In: *Advances in neural information processing systems, 33*.
11. Singh A, Subramanya S, Krishnaswamy R, et al. Freshdiskann: a fast and accurate graph-based ann index for streaming similarity search, 2021. 2105.09613.