

Review Article

Open Access Journal of Disease and Global Health

Vector Index Performance across Synthetic Distributions for Disease Surveillance and Global Health Retrieval

Dukhik Hovsepyan*^{ORCID}

PhD Researcher, European University of Armenia, Chair of Information Technologies and Applied Mathematics, Yerevan, Armenia

***Corresponding Author**

Dukhik Hovsepyan, PhD Researcher, European University of Armenia, Chair of Information Technologies and Applied Mathematics, Yerevan, Armenia.

Submitted: 2026, Sep 10; Accepted: 2026, Oct 14; Published: 2026, Oct 23

Citation: Hovsepyan, D. (2026). Vector Index Performance across Synthetic Distributions for Disease Surveillance and Global Health Retrieval. *Ope Acce Jou Dis Glo Heal*, 4(3), 01-10.

Abstract

Vector similarity search is increasingly relevant to information retrieval systems used in disease surveillance, biomedical knowledge management, and global-health information retrieval, where efficient access to semantically related records can support the analysis of large collections of multidimensional representations. However, selecting an appropriate vector index remains dependent on workload characteristics, including recall, query cost, construction cost, memory requirements, and vector distribution. This paper presents a reproducible, controlled comparison of four indexing approaches: exhaustive flat search, inverted-file search with exact reranking (IVF-Flat), random-hyperplane locality-sensitive hashing (LSH), and product quantization (PQ). Synthetic vector corpora are generated from isotropic Gaussian, clustered Gaussian-mixture, and anisotropic Gaussian distributions at three corpus sizes using a fixed random seed. For each size, 72 queries are evaluated against exhaustive squared-Euclidean top-10 ground truth. The experiments vary search breadth or representation size and measure Recall@10, query latency, candidate work, build time, and estimated index payload. At the largest corpus size of 1,800 vectors with 32 dimensions, IVF-Flat achieves exact recall on clustered data while evaluating approximately 596 candidates per query, whereas the best tested LSH configuration also achieves exact recall while evaluating approximately 347 candidates. On isotropic data, the tested LSH configuration reaches 0.742 Recall@10, while PQ remains between 0.14 and 0.18 across the evaluated distributions despite substantially reducing the estimated code payload. Exhaustive flat search is the fastest measured query path in this small-scale Python/NumPy implementation, demonstrating that approximate indexing does not necessarily provide a latency advantage for small collections and dimensionalities. The results are descriptive rather than inferential and should not be interpreted as production-scale rankings. Nevertheless, the study provides transparent evidence for distribution-conditioned index selection and establishes an experimental basis for future evaluation on larger biomedical and global-health datasets, including disease-surveillance records, health-related documents, and real embedding workloads.

Keywords: Vector Search, Approximate Nearest Neighbors, IVF, Locality-Sensitive Hashing, Product Quantization

1. Introduction

A vector index maps a collection of high-dimensional vectors to a structure intended to answer similarity queries with lower cost than scanning every item. Such collections arise from image descriptors, text embeddings, recommender systems, and scientific representations. Vector retrieval is also increasingly embedded in broader data systems: similarity search may be combined with relational predicates, graph traversal, or retrieval-augmented generation. The relevant decision is therefore not simply whether

an approximate algorithm has a favorable asymptotic bound. It is whether its accuracy, latency, construction overhead, storage footprint, update path, and filtering semantics fit a particular workload and deployment environment. Research systems range from hashing and quantization to inverted lists and proximity graphs; optimized libraries such as FAISS and integrated platforms such as Milvus make indexing one component of a larger data-management contract [1-9].

A familiar engineering shortcut is to compare one index's reported query time with another's and select the fastest. That comparison can be misleading when datasets, distance metrics, query distributions, implementation language, hardware, parameter settings, and recall targets differ. Benchmarking methodology itself can determine the result: ANN-Benchmarks formalized a repeatable framework, and subsequent work has emphasized that a comparison may reflect implementation choices rather than algorithmic properties [10-12]. In particular, exact search is both the natural reference for top-k quality and a competitive implementation option on small corpora or highly optimized hardware. Practitioner-focused experiments likewise find that flat search can remain viable where index construction and traversal overhead are not amortized [13].

Data geometry makes the problem harder. A clustered corpus offers opportunities for coarse partitioning, whereas diffuse or highly anisotropic data may stress the routing and candidates election rules of a given method. Intrinsic dimensionality, insertion order, and query-to-corpus distribution shift have been observed to affect graph-index recall [14]. Graph routing, adaptive partitioning, quantization, and filtering continue to evolve; recent studies examine probabilistic routing, streaming quantization, adaptive IVF maintenance, dynamic indexes, and graph search at large scale [15-20]. These results motivate controlled experiments that make both the data distribution and the measurement boundary explicit.

This paper studies a narrow but foundational question: how do four representative index families trade retrieval quality, measured latency, candidate work, construction time, and payload size across three known synthetic distributions? The analysis is intentionally small-scale and reproducible. It does not claim to reproduce billion-vector results, compare commercial services, or certify any implementation as suitable for deployment. Instead, the experiment exposes how choices interact under a known metric and controlled data generation, and it gives readers executable code and the complete generated JSON inputs and results.

The study makes four contributions. First, it supplies a seeded generator for three distributions with full vector and query arrays in JSON. Second, it implements a common exact-ground-truth evaluation for flat search, IVF-Flat, random-hyperplane LSH, and PQ, with explicit parameter sweeps. Third, it reports quality and systems-facing measurements together rather than reducing the outcome to a single latency number. Fourth, it interprets the observed behavior in light of recent ANN evaluation and indexing research and specifies what must change before these prototype results could support broad system-level claims.

The remainder of the paper is organized as follows. Section 2 introduces the index families and evaluation principles. Section 3 states the research questions. Section 4 details the data, implementations, metrics, and execution protocol. Section 5 reports the measurements. Sections 6 and 8 interpret the evidence and delimit its validity, followed by conclusions and reproducibility details.

2. Background and Related Work

2.1. Problem Definition and Exact Reference

Let $X = \{x_1, \dots, x_N\}$ be a database of vectors in $\mathbb{R}^d$ and let $q \in \mathbb{R}^d$ be a query. For a distance function D , exact top- k search returns the k elements of X with the smallest distance to q .

This paper uses squared Euclidean distance:

$$D(q, x) = \|q - x\|_2^2 = \sum_{j=1}^d (q_j - x_j)^2$$

which preserves Euclidean ranking and avoids an unnecessary square root. Approximate search returns a candidate set or ordered list that may omit one or more exact neighbors. We evaluate its item overlap with exact top- k ground truth, not distance-ratio approximation. This choice is common in empirical vector retrieval, but it should not be confused with application relevance, range-query quality, or end-to-end task success [10,12].

The classical search literature includes multidimensional tree structures and randomized hashing. A k - d tree organizes points by recursive coordinate partitions, while locality-sensitive hashing (LSH) maps nearby points to coincident buckets with a probability that depends on their similarity [21,2,4]. For angular proximity, random hyperplane hashes provide a compact, simple mechanism; more advanced LSH constructions improve theoretical or empirical behavior under selected metrics [1]. These methods trade table count, signature length, and bucket probing against collision probability and candidate volume. Recent indexing research also revisits graph pruning and multi-representation partitioning as complementary ways to improve approximate recall [22].

2.2. Coarse Partitions and Compressed Representations

An inverted-file index first assigns each vector to one or more coarse cells, commonly using centroids, and stores vectors by cell. At query time, the nearest coarse cells are selected and their contents searched. The parameter n_{probe} controls how many cells are visited: small values reduce work but risk missing true neighbors, while probing all cells approaches exhaustive search. Multi-index approaches subdivide the representation further to manage high-dimensional spaces [23]. This family is attractive because candidate work and storage access can be controlled explicitly, though centroid training, skewed list sizes, and data drift matter in practice. Recent work on adaptive IVF maintenance highlights that a static partition can degrade as a stream evolves [19].

Product quantization divides each vector into subspaces and learns a codebook per subspace. A vector is represented by a short sequence of centroid identifiers; a query can be scored through lookup tables that approximate distances to the encoded vector. PQ's purpose is primarily to lower representation and memory cost, not to preserve exact ranks for free. Accuracy depends on code length, subspace decomposition, training data, and distribution. Product quantization is foundational in large-scale similarity search, while modern alternatives study anisotropic or locally adaptive quantization and streaming conditions [15,24,5].

2.3. Proximity Graphs and Systems Context

Graph-based indexes connect vectors using proximity relationships and navigate the graph at query time. Navigable small-world methods and hierarchical navigable small-world (HNSW) graphs use local links and multiple navigation scales; Disk ANN demonstrates that graph search can be adapted to large, disk-resident collections [7,25,8]. Graph indexes commonly deliver strong recall-latency trade-offs, but their behavior depends on graph construction, search breadth, memory layout, and update strategy. The relationship between data order, intrinsic dimensionality, and recall has motivated recent empirical analysis of HNSW itself [14]. Other recent work evaluates graph-based designs systematically or improves their routing, filtering, and storage behavior [26,16-18].

A vector database adds concerns beyond nearest-neighbor traversal: metadata filtering, persistence, concurrency, consistency, updates, replication, resource isolation, and integration with structured queries. Milvus and TigerVector illustrate different system-level ways to incorporate vector search into database architectures; recent work also studies graph-database integration and adaptive indexing [27-29,9,20]. The present paper does not implement a complete database system. It focuses on the index-level retrieval operation and includes the surrounding system dimensions only as interpretation and future work.

2.4. Research Questions

We examine three research questions (RQs):

- How does Recall@10 vary across the three controlled data distributions and the tested operating points of IVF-Flat, LSH, and PQ?
- At these corpus sizes, do reductions in vector candidates translate into lower measured query latency than a vectorized exhaustive baseline?
- How do build time and estimated payload size modify the interpretation of the recall-latency trade-off?

We expect coarse partitions to benefit most from the clustered mixture, where cells can preserve local neighborhoods. We expect random-hyperplane LSH to be sensitive to distribution and to its number of tables. We expect PQ to produce the smallest representation but to lose top-k overlap at the short code lengths used here. We do not assume approximate methods will beat flat search at small N: vectorized exhaustive distance computation has regular memory access and no routing overhead. These are directional hypotheses, not guarantees, and the experiment is designed to expose counterexamples rather than tune the data until every hypothesis appears true.

3. Methods

3.1. Synthetic Data

The generator creates three independent corpora, each with $N_{max} = 1,800$ vectors, $d = 32$, and 72 queries. A fixed master seed of 20260930 is offset per distribution. The isotropic Gaussian corpus and queries have independent standard-normal coordinates. The clustered mixture uses 12 randomly sampled centers in R^{32} , with corpus and query points drawn from the same centers

plus independent Gaussian noise of standard deviation 0.45. The anisotropic Gaussian corpus and queries use independent coordinates with standard deviations geometrically decreasing from 1.0 to 0.025. All vectors are stored as 32-bit floats. Queries are generated independently of corpus rows but from the same distribution family; the clustered queries use independently drawn component labels.

For each distribution, the experiment evaluates prefixes of 500, 1,000, and 1,800 corpus vectors while retaining the same generated queries. This prefix design makes the size trend deterministic but does not constitute independent resampling at each N . A single realization per distribution is used. The full arrays, seed, distance definition, dimensions, and short distribution descriptions are included in `data/\*.json`. There are no external datasets, learned embeddings, demographic attributes, or metadata filters in this experiment.

3.2. Index Implementations

The four implementations are deliberately transparent NumPy prototypes. They share the same vector arrays, query vectors, squared-Euclidean metric, output size $k = 10$, exact ground truth, and timing wrapper. They are not bindings to optimized native libraries.

- **Flat exact search:** The baseline computes all N squared distances for a query, selects the ten smallest, and orders them. Its Recall@10 is 1.0 by construction. The measured index payload is the raw float32 vector matrix. This method has no training or graph construction stage.
- **IVF-Flat:** A Lloyd-style k -means routine trains $n_{list} = \min(32, \max(8, \lceil \sqrt{N} \rceil))$ centroids, assigns each vector to its nearest centroid, and stores lists of row identifiers. Search ranks centroids for the query, scans selected lists using exact vector distances, and returns the best ten. We sweep $n_{probe} \in \{1, 2, 4, 8, n_{list}\}$ after removing duplicate values. At $n_{probe} = n_{list}$, all assigned vectors are candidates and recall should match exact search, modulo implementation ties. The payload estimate includes raw vectors, float32 centroids, and 32-bit identifiers.
- **Random-hyperplane LSH:** Each table uses ten independently sampled Gaussian hyperplanes to form a 10-bit sign signature. The sweep uses 2, 4, 8, or 16 tables. Candidate IDs from the query's exact-signature and one-bit-neighbor buckets are unioned; if fewer than ten candidates are found, radius-two buckets are added. Candidates are reranked by exact squared-Euclidean distance. This is a simple angular-projection hash, not a tuned Euclidean LSH family. The measured payload estimate includes raw vectors, hyperplane coefficients, and table identifiers. It excludes Python dictionary/object overhead.
- **Product quantization:** Vectors are split into four equal subspaces; an independent k -means codebook with 8, 16, or 32 centroids is trained per subspace. Each vector is represented by four byte-valued code identifiers. At query time, lookup tables estimate distances from the query sub-

vectors to codebook centroids, and the ten lowest estimated-distance codes are returned without full-precision reranking. The payload estimate counts bit-packed identifiers at the nominal code width plus float32 codebooks; it excludes raw vectors because query ranking operates on codes. It also excludes the Python array/object overhead.

The implementations permit useful structural comparisons, but important native optimizations are absent: SIMD kernels, multithreaded construction/search, cache-aware graph layouts, compact production serialization, memory mapping, disk I/O, GPU kernels, and batched query execution. Build times and latencies consequently characterize these scripts and the recorded host, not the asymptotic or production performance of a method family.

3.3. Ground Truth and Metrics

For each query q_i , exhaustive search over the relevant corpus prefix produces the exact neighbor set G_i of size $k = 10$. If an index returns A_i , the per-query recall is

$$\text{Recall}@k(q_i) = \frac{|A_i \cap G_i|}{k}, \quad \overline{\text{Recall}@k} = \frac{1}{Q} \sum_{i=1}^Q \text{Recall}@k(q_i)$$

Mean and 95th-percentile query latency are computed from per-query medians over three timed repetitions after one warm-up search. The 72 query-level medians are then summarized. The experiment also reports average candidates scored, index build time, and a logical payload estimate in bytes. Candidate counts do not include IVF centroid-routing comparisons or LSH projection/hash work; for PQ, one scored candidate means one compressed code, not one full float32 distance. They are therefore algorithm-specific work indicators, not interchangeable counts of equal-cost operations.

Build time is measured with `time.perf_counter` around training and structure construction, excluding JSON loading. Query time includes routing, candidate formation, score computation, and top- k selection for the relevant prototype. It excludes process startup, network transit, index deserialization, query parsing outside the function, and persistence. No hardware counters or process-resident memory snapshots are collected. We use no hypothesis tests or confidence intervals because each configuration has a single fixed corpus/query realization; query-level variation is descriptive and does not substitute for independent data seeds.

3.4. Execution and Configurations

The full run was executed on Windows Server 2025, Python 3.14.7, and NumPy 2.4.6 using the supplied selected interpreter; Matplotlib 3.11.2 produced the figures. The processor identifier is recorded by the script when exposed by Python, although it is not a marketing model name. The full result summary stores Python, NumPy, processor, and platform fields so that the environment can be checked. The script uses no GPU and no external vector-index package.

At each distribution-size pair, one IVF structure is built and reused across its probe counts; each LSH table-count setting and PQ codebook size is built separately. The resulting 13 search operating points per pair comprise one flat baseline, five IVF settings, four LSH settings, and three PQ settings. Three distributions times three sizes yield 117 measurement records. A reduced `--quick` mode is supplied for pipeline checks, but only `results/benchmark\_results.json` and the full-run figures are used in the findings below.

4. Results

4.1. Recall Depends on both Distribution and Operating Point

At $N=1,800$, flat search obtains exact Recall@10 for every distribution. IVF-Flat also reaches 1.0 recall on the clustered mixture with eight of 32 lists probed, scanning an average of 596 vectors per query. Its average candidate count is consistent with the 12-component structure: when the coarse quantizer routes a query to a few relevant cells, it can avoid most of the corpus scan while preserving the exact top ten for this fixed query set. On isotropic and anisotropic data, the best-recall IVF operating point probes all 32 lists, returning to a full candidate scan. In these two distributions the partition structure offers no tested reduction in work at perfect recall.

LSH is also distribution-sensitive. On the clustered mixture, eight tables reach 1.0 recall while scoring about 347 candidates on average. On anisotropic Gaussian data, 16 tables reach 0.961 recall with roughly 474 candidates. On isotropic Gaussian data, the best tested setting reaches 0.742 recall at 16 tables, with roughly 357 candidates. More hash tables increase the probability that neighbors collide, but also increase construction and storage costs; the results show that more tables do not erase distribution-dependent misses at this parameter range.

PQ yields a compact code representation but comparatively low exact-neighbor overlap. With the largest tested codebook (32 centroids per subspace), Recall@10 is 0.183 on isotropic, 0.179 on clustered, and 0.143 on anisotropic data. The estimated payload is approximately 9.5 kB, compared with 230.4 kB for flat float32 vectors. This is a roughly 24-fold payload reduction under the stated accounting, not a measured resident-memory reduction. The code has no reranking stage, so quantization error directly determines the returned ordering. The result motivates additional code lengths, subspace counts, residual encoding, and reranking rather than a general conclusion that PQ cannot achieve high recall.

4.2. Recall-Latency Frontier and Scale

Figure 1 plots the tested recall-latency operating points at the largest corpus size. On this machine and at this small N , flat search is the lowest-latency exact method in all three panels. IVF-Flat only becomes competitive on the clustered mixture at the eight-list operating point; LSH saves candidate vector work there but remains slower than the vectorized exhaustive baseline. The gap between fewer candidate distances and lower measured time is not contradictory: these Python prototypes include hash-bucket assembly, centroid routing, temporary array allocation, and top- k operations, while flat distance computation is a compact vectorized

kernel.

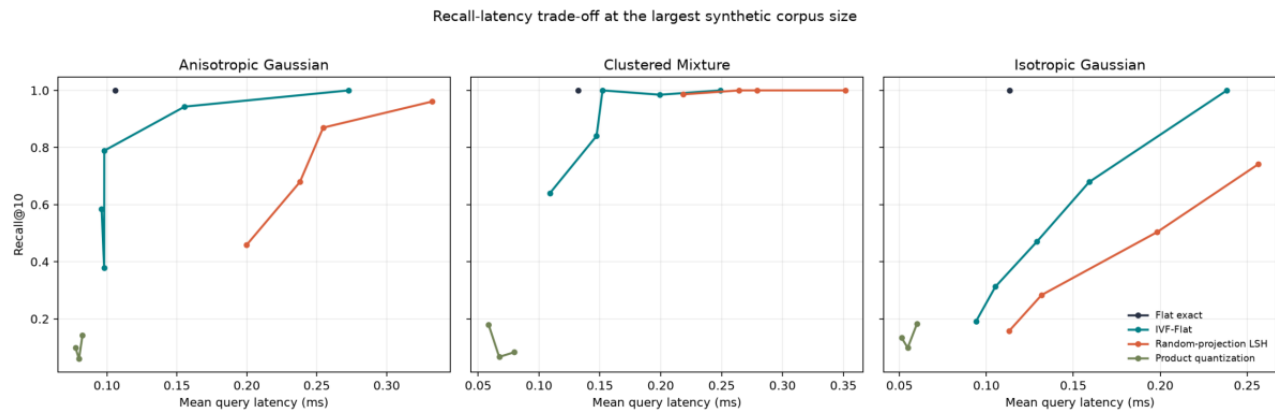


Figure 1: Recall@10 Against mean Query Latency for Tested Settings at the Largest Corpus Size. The Lines Connect Configurations within each Method; a Line is not a Pareto Frontier unless Dominated Settings are Removed

The latency-versus-size curves in Figure 2 use each method's highest-recall tested configuration at each size. They are therefore not matched-recall comparisons. The curves should be read as an illustration of measured scaling under this controlled setup, not as a universal complexity result.

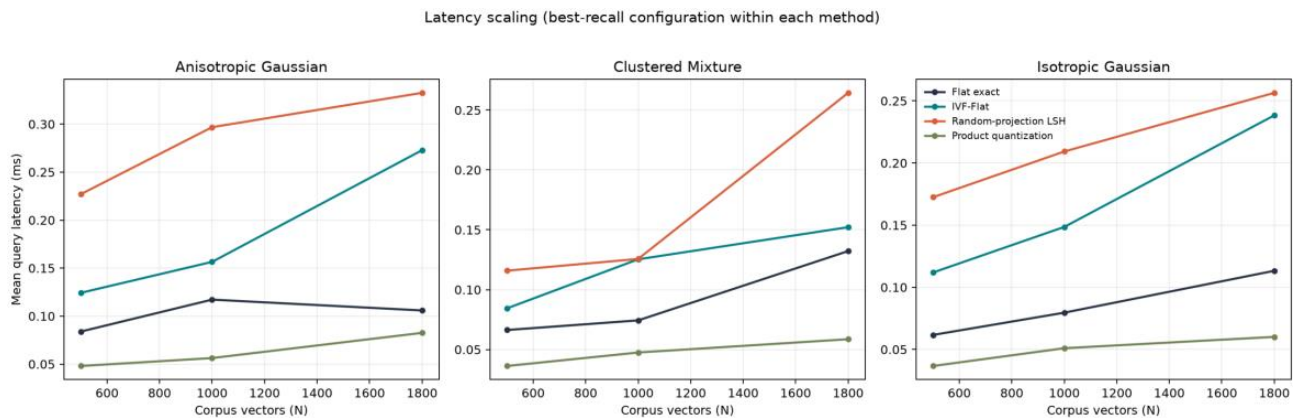


Figure 2: Measured Latency versus Corpus Size using the Highest-Recall Tested Configuration within each Method at each Size. Since Recall can Differ Across Methods and Sizes, this is Descriptive Scaling rather than a Fixed-Quality Comparison

The construction-time curves in Figure 3 show a different cost dimension: flat has no build phase, while clustering and codebook training incur nonzero setup cost. For short-lived or frequently rebuilt collections, the amortization horizon can matter more than steady-state query latency.

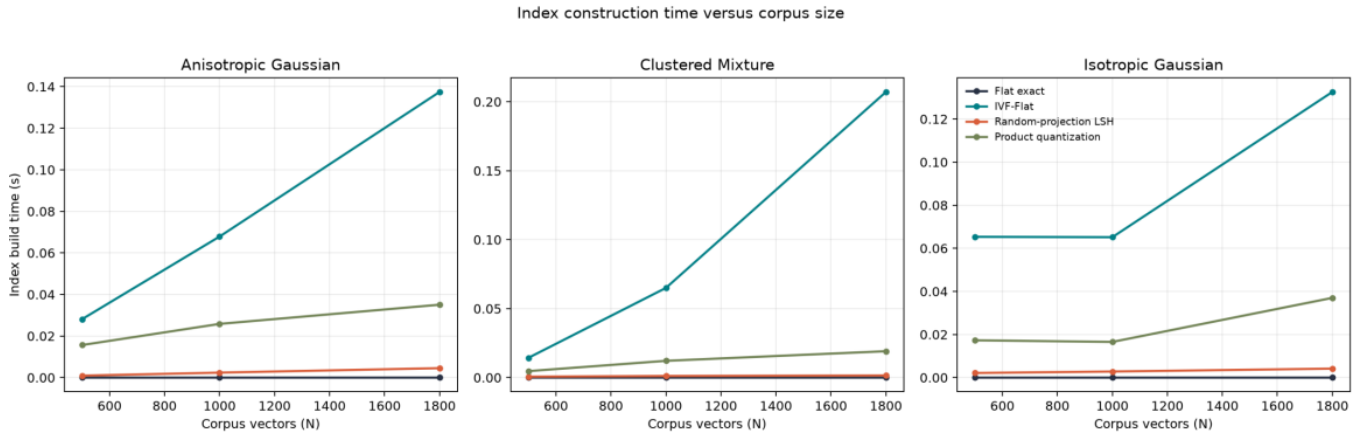


Figure 3: Measured Index Build Time Versus Corpus Size. Flat Search Requires no Index Training; other Values include the Relevant Prototype's Construction Procedure

Distribution	Method	Setting	Recall@10	Latency (ms)	Candidates	Payload (kB)	Build (s)
Isotropic	Flat exact	—	1.000	0.113	1,800	230.4	0.000
Isotropic	IVF-Flat	n_probe = 32	1.000	0.238	1,800	241.7	0.133
Isotropic	LSH	16 tables	0.742	0.256	357	366.1	0.033
Isotropic	PQ	32 codewords	0.183	0.060	1,800	9.5	0.155
Clustered	Flat exact	—	1.000	0.132	1,800	230.4	0.000
Clustered	IVF-Flat	n_probe = 8	1.000	0.152	596	241.7	0.207
Clustered	LSH	8 tables	1.000	0.264	347	298.2	0.004
Clustered	PQ	32 codewords	0.179	0.059	1,800	9.5	0.138
Anisotropic	Flat exact	—	1.000	0.106	1,800	230.4	0.000
Anisotropic	IVF-Flat	n_probe = 32	1.000	0.273	1,800	241.7	0.137
Anisotropic	LSH	16 tables	0.961	0.333	474	366.1	0.051
Anisotropic	PQ	32 codewords	0.143	0.083	1,800	9.5	0.143

Table 1: Best-Recall Tested Configuration at N = 1,800, 32 Dimensions, Q = 72, and k = 10. Latency is the mean of Query-Level Medians; Payload is a Logical Estimate

4.3. Memory and Construction Interpretation

For the largest vectors array, raw float32 payload is $1,800 \times 32 \times 4 = 230,400$ bytes. The IVF estimate adds centroid values and one 32-bit list identifier per vector, yielding about 241.7 kB. The LSH estimate is larger, about 366.1 kB at 16 tables, because the method retains raw vectors for exact reranking and adds hyperplanes and identifiers. The PQ estimate at 32 codewords is about 9.5 kB for packed four-subspace codes and codebooks. These are accounting estimates with deliberately different algorithmic assumptions. In a production service, allocator overhead, graph links, metadata, replicas, alignment, caches, file formats, and raw-vector retention can dominate or alter the comparison. Disk-resident graph systems in particular depend on page layout and I/O behavior not represented by in-memory byte [17].

The full measurements do not support a claim that approximate indexing improves every metric. Flat's measured query path is faster at this scale; its costs grow linearly with the corpus and

it stores full vectors, but its compact vectorized kernel has low fixed overhead. IVF and LSH reduce candidates in the clustered case but pay for setup and query routing. PQ gives the smallest code payload yet loses recall because the tested codebooks are intentionally short and there is no exact reranking. The expected systems trade-off emerges only when recall, memory, build effort, and actual deployment-scale throughput are considered jointly.

5. Discussion

5.1. Interpreting Distribution-Conditioned Behavior

The clearest finding is not a winner but a dependency: data geometry changes the value of routing. In the clustered mixture, IVF's coarse centroids align with the generative components and its list selection can preserve exact top-10 results while scanning about one third of the vectors. LSH also benefits, reaching exact recall at eight tables in the tested set. In the isotropic case, each local neighborhood is less cleanly separated by these particular learned or random partitions; IVF requires a full scan for perfect recall,

and the tested LSH sweep plateaus below 0.75. The anisotropic case is not equivalent to low dimensionality: the vectors still occupy 32 coordinates, but their variance is concentrated in fewer directions. LSH obtains near-exact recall at its widest tested setting, while IVF returns to full scanning for exact recall. These observations are consistent with the broader emphasis on intrinsic dimensionality and data ordering in graph search and on adaptive search for evolving or skewed workloads [14,19,20].

The data distributions here are intentionally simple. Real embeddings can be multimodal, heavy-tailed, normalized, anisotropic, clustered at multiple scales, temporally drifting, and correlated with attributes. An experiment on three distributions cannot model that diversity. It does, however, make a hidden assumption visible: results from one synthetic distribution should not be generalized to another without measurement. This is particularly important for search settings where vectors and queries are not identically distributed, a concern reflected in both graph-index studies and adaptive retrieval work [16,12,29].

5.2. Why Exact Search Wins here

The flat baseline should not be dismissed as an asymptotic straw man. At $N \leq 1,800$ and $d = 32$, a single vectorized matrix operation can evaluate all distances efficiently. An index must pay for routing, candidate gathering, and ranking, with a small work reduction that may not cover those costs. This result is also a warning about the implementation boundary: native indexes can use SIMD, carefully laid out memory, multi-query batching, and optimized selection routines. The present Python code cannot establish the corpus size at which a native IVF, graph, or quantized index becomes faster. Operational comparisons in recent retrieval research similarly treat flat search as a legitimate option rather than assuming graphs dominate at every scale [27,13].

A rigorous follow-up should compare at matched recall targets, for example 0.90, 0.95, and 0.99, rather than selecting each method's maximum recall and plotting it against latency. It should also sweep substantially larger N , report query throughput under concurrency, and include build amortization over an explicit query count. A Pareto analysis could remove dominated configurations and show the memory-latency frontier. In production-oriented studies, exact search could be the fallback for small partitions or highly selective filters, while approximate search handles larger candidate universes. Filter-aware research such as Filtered-DiskANN demonstrates that even the query semantics can change index design and quality trade-offs [26].

5.3. Quantization and Recall are Separate Objectives

PQ's storage result is structurally important but not sufficient for selection. Its short codes reduce the representation by roughly a factor of 24 in the stated logical accounting, while Recall@10 remains low for this setup. A common production pattern is to retrieve a broader approximate candidate set and rerank in full precision, or to use residual/additive encodings and query-adaptive scoring. Recent work on local adaptation, reduced-rank scoring,

and streaming quantization illustrates that modern quantization systems are more sophisticated than this basic PQ prototype [15,24]. We use PQ as a representative compression family, not as a state-of-the-art implementation.

Future experiments should vary both the number of sub quantizers and the number of codewords, assess codebook training data separately from evaluation queries, and measure precision after reranking. A fair memory comparison should state whether the raw vector is retained for reranking; if so, the compressed representation alone is not the total index payload. The current script intentionally distinguishes those cases: LSH counts raw vectors because it reranks exact candidate distances, while PQ does not retain or use the raw vectors in its ranking path. This means the bytes column compares the actual prototype payloads under their respective retrieval paths, not a shared storage policy.

5.4. Implications for Vector Data Systems

An index is one component of a data system, not an isolated algorithm. Vector databases support updates, filters, consistency, persistence, and mixed queries; graph stores may combine vector search with graph neighborhoods; cloud-native deployments face memory bandwidth, storage I/O, and CPU heterogeneity. Systems such as Milvus and Tiger Vector illustrate these integration choices [28,9]. Adaptive indexing work such as Quake and recent graph evaluations address distribution change, workload skew, and construction behavior beyond the static setting tested here [16,20]. Disk-resident index layout is another independent axis, as reflected in recent storage-oriented analysis [17].

Consequently, a journal-strength extension should preserve the controlled synthetic experiments as one diagnostic tier and add established public vector benchmarks plus task-relevant embeddings. It should compare optimized, version-pinned native implementations under identical hardware and build options, report recall-latency and recall-memory Pareto fronts, characterize build and update costs, and isolate filtered and unfiltered workloads. Repeated independent seeds and bootstrap intervals over queries would quantify variability; multi-run medians on isolated hardware would improve timing confidence. The submission should also archive commands, package versions, CPU/RAM details, data checksums, and index parameters. These practices are consistent with benchmark and reproducibility lessons from ANN evaluation [10-12].

5.5. Potential Applications in Disease Surveillance and Global Health

Public-health organizations may need to search outbreak situation reports, pathogen and genomic surveillance summaries, clinical guidelines, systematic reviews, health-facility reports, and epidemiological bulletins. Vector retrieval can help find semantically related evidence across changing terminology and document collections, but should be combined with structured filters for disease, location, reporting period, population, source, and document version. The index choices in Table 2 are conditional

engineering guidance based on the mechanisms discussed in this paper; this benchmark contains no clinical or public-health data and cannot establish which index performs best on those workloads.

Use case	Index that may fit	Why it may fit	Main safeguard
Small, curated outbreak or guideline collection	Flat exact search	Exact top-k is straightforward to audit and avoids index-build overhead	Keep date, location, and source filters explicit; compare semantic results with expert judgments
Large corpus with strongly clustered disease or regional reports	IVF-Flat	Coarse cells can reduce candidate scans when clusters align with disease, geography, or report type; clustered synthetic data favored IVF in this experiment	Verify cluster alignment on real embeddings and test recall across regions and outbreaks
Near-duplicate alert and situation-report discovery	LSH candidate generation with exact reranking	Hash buckets can produce a smaller candidate set for similarity-based deduplication	The tested random-hyperplane LSH is distribution-sensitive; validate recall and duplicate precision under the chosen metric
Large archive with strict memory limits or limited-connectivity deployment	PQ candidate generation with full-precision reranking	Compressed codes can reduce vector payload when retaining every full vector is costly	Tested short-code PQ had low Recall@10 without reranking; measure evidence recall after reranking
Large, latency-sensitive and frequently queried evidence collection	HNSW, subject to workload testing	Graph navigation is a plausible high-recall, low-latency option when memory is available	HNSW was not implemented or benchmarked here; measure update, memory, and filtered-query behavior

Table 2: Conditional Index Guidance for Disease Surveillance and Global-Health Retrieval. Recommendations are Inferred from Index Mechanisms and this Paper’s Synthetic Results, not Measured on Health Data

For example, searching a small regional archive for prior guidance on a newly reported pathogen can begin with flat exact search, providing a simple reference while filtering by date, location, and source. For a much larger collection of surveillance reports, IVF-Flat may be worth testing if embeddings form useful disease- or region-specific clusters; the clustered synthetic result suggests this mechanism can reduce candidate work, but does not demonstrate that real reports will have the same geometry. HNSW is another candidate when query latency is important and the memory budget permits it, though this study provides no HNSW measurements.

LSH may be useful for surfacing near-duplicate situation reports or alerts so that analysts can identify repeated or syndicated content. It should be evaluated as a candidate generator, followed by exact or learned reranking, because missed reports and false duplicate groupings can distort situational awareness. PQ may be appropriate for retaining a broad archive on memory-constrained systems, but should generally be treated as a compressed first stage: in this experiment, PQ achieved less than 0.19 Recall@10 at the largest tested codebook setting. A health deployment would need to assess whether reranking restores recall for critical evidence and whether the remaining payload savings justify the added complexity.

These retrieval systems should support, not replace, epidemiological interpretation or clinical judgment. Similarity scores are not disease diagnoses, incidence estimates, or causal evidence. Evaluation should use appropriately governed data, point-in-time splits,

source provenance, multilingual and regional coverage checks, outbreak-shift tests, and task-specific measures such as evidence recall and duplicate-detection precision. Privacy, access controls, representativeness, and potential disparities in coverage should be assessed before operational use. No patient records, health-facility records, or other health data were used in this study.

6. Conclusion

This paper provides a reproducible synthetic comparison of exact flat search, IVF-Flat, random-hyperplane LSH, and product quantization across isotropic, clustered, and anisotropic vector data. In the fixed 32-dimensional, 1,800-point experiments, clustered geometry enables IVF and LSH to achieve exact Recall@10 while evaluating substantially fewer full vectors. LSH approaches exact recall on anisotropic data but is weaker on isotropic data at the tested settings. PQ compresses the logical representation markedly, but its short codes produce low top-10 overlap without reranking. Exhaustive vectorized search remains the fastest measured query path at these small sizes. These findings provide a basis for conditional index-selection guidance for potential applications in disease surveillance and global-health information retrieval, where the choice between exact search, clustered indexing, hashing, and compression may depend on collection size, vector distribution, latency, and memory constraints.

Declarations

Funding

This research received no specific grant from any funding agency.

Conflict of Interest

The authors declare no conflict of interest.

Data Availability Statement

Data available upon reasonable request.

AI Usage Statement

State whether generative AI tools were used for language editing only or not used at all.

References

1. Andoni, A., Indyk, P., Laarhoven, T., Razenshteyn, I., & Schmidt, L. (2015). Practical and optimal LSH for angular distance. *Advances in neural information processing systems*, 28.
2. Datar, M., Immorlica, N., Indyk, P., & Mirrokni, V. S. (2004, June). Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry* (pp. 253-262).
3. Douze, M. (2024). Guzhva A Deng C Johnson J Szilvasy G Mazaré PE et al. *The Faiss library*. *arXiv*.
4. Indyk, P., & Motwani, R. (1998, May). Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of the thirtieth annual ACM symposium on Theory of computing* (pp. 604-613).
5. Jegou, H., Douze, M., & Schmid, C. (2010). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33(1), 117-128.
6. Johnson, J., Douze, M., & Jégou, H. (2019). Billion-scale similarity search with GPUs. *IEEE transactions on big data*, 7(3), 535-547.
7. Malkov, Y. A., & Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4), 824-836.
8. Jayaram Subramanya, S., Devvrit, F., Simhadri, H. V., Krishnaswamy, R., & Kadekodi, R. (2019). Diskann: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing Systems*, 32.
9. Wang, J., Yi, X., Guo, R., Jin, H., Xu, P., Li, S., ... & Xie, C. (2021, June). Milvus: A purpose-built vector data management system. In *Proceedings of the 2021 international conference on management of data* (pp. 2614-2627).
10. Aumüller, M., Bernhardsson, E., & Faithfull, A. (2020). ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems*, 87, 101374.
11. Kriegel, H. P., Schubert, E., & Zimek, A. (2017). The (black) art of runtime evaluation: Are we comparing algorithms or implementations?. *Knowledge and Information Systems*, 52(2), 341-378.
12. Simhadri, H. V., Williams, G., Aumüller, M., Douze, M., Babenko, A., Baranchuk, D., ... & Wang, J. (2022, July). Results of the NeurIPS'21 challenge on billion-scale approximate nearest neighbor search. In *NeurIPS 2021 Competitions and Demonstrations Track* (pp. 177-189). PMLR.
13. Lin, J. (2025, July). Operational advice for dense and sparse retrievers: HNSW, flat, or inverted indexes?. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 6: Industry Track)* (pp. 865-872).
14. Elliott, O. P., & Clark, J. (2024, August). The impacts of data, ordering, and intrinsic dimensionality on recall in hierarchical navigable small worlds. In *Proceedings of the 2024 ACM SIGIR International Conference on Theory of Information Retrieval* (pp. 25-33).
15. Aguerrebere, C., Hildebrand, M., Bhati, I. S., Willke, T., & Tepper, M. (2024). Locally-adaptive quantization for streaming vector search. *arXiv preprint arXiv:2402.02044*.
16. Azizi, I., Echihiabi, K., & Palpanas, T. (2025). Graph-based vector search: An experimental evaluation of the state-of-the-art. *Proceedings of the ACM on Management of Data*, 3(1), 1-31.
17. Li, L., Gong, S., Yang, Y., Wang, Y., & Wu, J. (2026). I/O Optimizations for Graph-Based Disk-Resident Approximate Nearest Neighbor Search: A Design Space Exploration. *arXiv preprint arXiv:2602.21514*.
18. Lu, K., Xiao, C., & Ishikawa, Y. (2024). Probabilistic routing for graph-based approximate nearest neighbor search. *arXiv preprint arXiv:2402.11354*.
19. Mohoney, J., Pacaci, A., Chowdhury, S. R., Minhas, U. F., Pound, J., Renggli, C., ... & Venkataraman, S. (2024). Incremental ivf index maintenance for streaming vector search. *arXiv preprint arXiv:2411.00970*.
20. Mohoney, J., Sarda, D., Tang, M., Chowdhury, S. R., Pacaci, A., Ilyas, I. F., ... & Venkataraman, S. (2025). Quake: adaptive indexing for vector search. In *19th USENIX Symposium on Operating Systems Design and Implementation (OSDI 25)* (pp. 153-169).
21. Bentley, J. L. (1975). Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9), 509-517.
22. Sun, P., Simcha, D., Dopson, D., Guo, R., & Kumar, S. (2023). SOAR: improved indexing for approximate nearest neighbor search. *Advances in Neural Information Processing Systems*, 36, 3189-3204.
23. Lempitsky, V. (2012, June). The inverted multi-index. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)* (pp. 3069-3076).
24. Jääsaari, E., Hyvönen, V., & Roos, T. (2024). Lorann: Low-rank matrix factorization for approximate nearest neighbor search. *Advances in Neural Information Processing Systems*, 37, 102121-102153.
25. Malkov, Y., Ponomarenko, A., Logvinov, A., & Krylov, V. (2014). Approximate nearest neighbor algorithm based on navigable small world graphs. *Information Systems*, 45, 61-68.
26. Gollapudi, S., Karia, N., Sivashankar, V., Krishnaswamy, R., Begwani, N., Raz, S., ... & Simhadri, H. V. (2023, April). Filtered-diskann: Graph algorithms for approximate nearest

-
- neighbor search with filters. In *Proceedings of the ACM Web Conference 2023* (pp. 3406-3416).
27. Kuffo, L., & Boncz, P. (2025, June). Bang for the Buck: Vector Search on Cloud CPUs. In *Proceedings of the 21st International Workshop on Data Management on New Hardware* (pp. 1-8).
28. Liu, S., Zeng, Z., Chen, L., Ainihaer, A., Ramasami, A., Chen, S., ... & Wang, J. (2025, June). Tigervector: Supporting vector search in graph databases for advanced rags. In *Companion of the 2025 International Conference on Management of Data* (pp. 553-565).
29. Wei, J., Xu, Q., & Yang, C. (2026). The Virtuous Cycle: AI-Powered Vector Search and Vector Search-Augmented AI. *arXiv preprint arXiv:2603.09347*.

Copyright: ©2026 Dukhik Hovsepyan. This is an open-access article distributed under the terms of the Creative Commons Attribution License, which permits unrestricted use, distribution, and reproduction in any medium, provided the original author and source are credited.