International Journal of Research Publications in Engineering, Technology and Management (IJRPETM)

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 4, Issue 4, July – August 2021||

DOI:10.15662/IJRPETM.2021.0404007

The Vector Database Gap: A Production Blueprint for Embedding-Based Enterprise Search

Rahul Jain

Infra Technology Specialist, USA

ABSTRACT: Transformer-based embedding models have solved the representation problem in enterprise search, converting documents, queries, and catalog items into dense vectors of high semantic fidelity. However, serving these vectors at production scale—on corpora of tens of millions of documents—exposes a systems gap that algorithm libraries and general-purpose search engines do not adequately fill. In this paper, we benchmark three vector computation libraries (FAISS, Annoy, and ScaNN) and Elasticsearch 7.14 against six operational requirements critical to production deployments: incremental index mutation, low-latency search under concurrent load, metadata-filtered vector search, embedding-model version migration, cross-region consistency, and recall observability. Experiments were conducted on a production-mirror index of 85 million documents using a 768-dimensional transformer model. We find that no existing system scores above two out of three on more than two requirements, and that recall observability is entirely absent across all evaluated platforms. We further note that Apache Solr, as of August 2021, does not yet expose dense vector field types, and that Elasticsearch 7.x relies on brute-force script_score evaluation, making approximate nearest-neighbor search unavailable in the dominant open-source search-engine stack. To close this gap, we propose a three-layer reference architecture that decouples mutable storage, incremental indexing, and filter-aware query planning, and we contextualize this design against the emerging generation of dedicated vector databases.

KEYWORDS: Vector Database, Embedding, Transformer, Approximate Nearest Neighbor, Enterprise Search, Production Systems.

I. INTRODUCTION

Enterprise search systems built on keyword matching suffer from a fundamental semantic gap. A query for a laptop that 'does not overheat' should return results about thermal design and cooling systems, even when the product description never contains the word 'overheat.' Dense vector representations, produced by transformer-based encoders, address this problem at the representation layer. A fixed-length vector captures the meaning of text, images, or structured catalog items such that nearby vectors in the embedding space correspond to semantically similar content. This idea is not new—vector-space models have been studied for decades—but the quality of modern encoders and the scale at which organizations now wish to search have transformed it from a research curiosity into a production necessity.

The research community has responded with a rich family of approximate nearest neighbor (ANN) algorithms. Early work focused on reducing the candidate-selection cost of brute-force search (Iwamura et al., 2013). Subsequent methods partitioned the vector space with tree-like or cut-region structures (Lokoč et al., 2014), followed by randomized approaches that offer per-query control over the speed–accuracy trade-off (Li & Malik, 2016, 2017). Graph-based methods, notably the Hierarchical Navigable Small World (HNSW) algorithm, introduced multi-layer graph structures that eliminate the need for a separate coarse-search phase (Malkov & Yashunin, 2018). A comprehensive comparison of nineteen algorithms on twenty workloads found that no single method dominates all scenarios; recall, query time, and memory usage must be evaluated jointly (Li et al., 2019). Other advances extended ANN search to maximum inner product search (Abuzaid et al., 2019), high-dimensional quantization (Guzun & Canahuate, 2019), graph-structured data (Zhang et al., 2019), GPU acceleration (Johnson et al., 2019), and disk-resident indexes (Li et al., 2018).

This literature demonstrates that the algorithmic foundations of approximate search are mature. We contend, however, that a mature algorithm is not the same as a mature system. A production search system must mutate under write pressure, respond under concurrent read pressure, filter on metadata, manage model versions, maintain consistency across regions, and monitor its own health. The works cited above do not address these concerns because they are systems problems, not algorithm problems. This is the gap we examine in the present paper.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 4, Issue 4, July – August 2021||

DOI:10.15662/IJRPETM.2021.0404007

II. TWO EXISTING CATEGORIES

We group the available tooling into two categories: vector computation libraries and general-purpose search engines. As of August 2021, neither category satisfies the full set of production requirements we identify in Section 3.

2.1 Vector Computation Libraries

FAISS, Annoy, and ScaNN are widely used libraries that expose optimized implementations of the algorithms reviewed above. FAISS (Facebook AI Similarity Search) supports GPU acceleration, inverted-file indexes, and graph-based indexes including HNSW, and has been demonstrated at billion-vector scale (Johnson et al., 2019; Baranchuk et al., 2018). Annoy (Approximate Nearest Neighbors Oh Yeah) builds forests of random projection trees and trades memory for query throughput. ScaNN (Scalable Nearest Neighbors), open-sourced by Google in 2020, combines anisotropic vector quantization with a scoring step to achieve state-of-the-art recall–latency trade-offs on several standard benchmarks (Guo et al., 2020).

All three systems are designed as libraries, not databases. They ingest a fixed corpus of vectors, build an index in a single batch, and expose read-only query interfaces. This design mirrors academic benchmarks such as SIFT1M or YFCC100M, where a static corpus, a static query set, and a one-time build step are assumed (Iwamura et al., 2013; Johnson et al., 2019). Enterprise search, by contrast, is a continuous process of insertion, update, and deletion. Product catalogs change hourly; support-ticket corpora grow by thousands of documents per day. Annoy requires a full rebuild after any addition because its random-projection forests are immutable once constructed. ScaNN trains its quantization codebook on a fixed sample and must be retrained when the data distribution shifts. FAISS offers incremental addition for flat and inverted-file indexes, but its HNSW implementation does not support vector removal; deleting a node would destroy graph connectivity, and while additions are possible, repeated insertions without periodic rebuilds can degrade search quality as the graph structure diverges from the optimal layout for the current dataset.

2.2 General-Purpose Search Engines

Elasticsearch and Apache Solr dominate the open-source search-engine landscape. Both were originally designed as inverted text indexes, and their query planners, scoring pipelines, replication mechanisms, and client ecosystems reflect that heritage. As of August 2021, their relationship with dense vector search differs markedly.

Elasticsearch introduced the dense_vector field type in version 7.3 (generally available in 7.6). In the 7.x release line, however, vector similarity is evaluated through the script_score query, which performs brute-force cosine or Euclidean similarity over all documents matching a preceding text query (or match_all). This approach is exact—recall is perfect—but it is computationally expensive and does not scale to large corpora without aggressive text pre-filtering. Native approximate kNN search, based on HNSW, is under active development and is expected in the forthcoming 8.0 release, but it is not available in production today. Consequently, Elasticsearch 7.x offers the operational comforts of a mature distributed database (incremental document CRUD, metadata filtering, cross-cluster replication, and monitoring) at the cost of vector-search latency that grows linearly with corpus size.

Apache Solr, as of version 8.10 (the current release in August 2021), does not expose a dense vector field type. Vector search capabilities are planned for the upcoming Solr 9.0 release, which is expected to leverage Lucene 9's HNSW integration, but no production-ready implementation exists in the current release line. For this reason, Solr is excluded from our quantitative benchmark and is discussed only as a point of architectural comparison. The architectural tension in Elasticsearch 7.x is instructive. Because the dense_vector field is treated as an attribute among many—like a text field, a date, or a keyword—the query planner reasons about it through the inverted-index abstraction first. A vector similarity query must either wrap a restrictive text filter (which risks missing semantically relevant documents that lack keyword overlap) or execute as a match_all script_score (which scans the entire corpus). Metadata filtering and vector ranking are not treated as co-equal signals; the planner was not designed with the assumption that vector distance could be the primary ranking metric.

III. EXPERIMENTAL DESIGN

To quantify the gap between libraries and search engines, we constructed a production-mirror environment indexing 85 million documents drawn from three enterprise sources: historical support tickets, a product catalog, and internal knowledge-base pages. Documents were encoded with a 768-dimensional transformer model (Sentence-BERT, all-mpnet-base-v2). All experiments were run on identical hardware: dual Intel Xeon Gold 6248R CPUs (40 cores total),

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 4, Issue 4, July – August 2021||

DOI:10.15662/IJRPETM.2021.0404007

512 GB DDR4 RAM, NVMe SSD storage, and no GPU acceleration, to reflect a typical mid-size enterprise deployment. We evaluated four systems: FAISS 1.7.1 (HNSW index, M=32, efConstruction=200), Annoy 1.17.0 (200 trees), ScaNN 1.2.3 (default asymmetric hashing with 1,000 partitions), and Elasticsearch 7.14.0 (dense_vector field with script_score cosine similarity). Each system was subjected to the same insert stream, query mixture, and concurrent load pattern: 200 queries per second sustained for 30 minutes, with a 70/30 split between pure vector queries and vector-plus-metadata-filter queries. For filtered queries, a metadata predicate isolated approximately 5% of the corpus (e.g., products in a specific price band or tickets from a specific region).

Six properties were measured. (1) Incremental mutation cost: the wall-clock time for a newly inserted document to become searchable without rebuilding the entire index. (2) P95 query latency under 200 concurrent queries per second. (3) Recall at 10: the fraction of true nearest neighbors present in the top-10 returned results, measured both without filters and with a 5% metadata filter. Ground-truth neighbors were computed with brute-force exhaustive search on a held-out validation set of 10,000 queries. (4) Model migration cost: the operator time required to transition a live index from one embedding-model version to another without downtime. (5) Cross-region replication lag: the elapsed time between a primary write and its visibility in a secondary region. (6) Recall observability: whether the system exposes a first-class metric indicating recall degradation without requiring an external evaluation harness.

3.1 Latency and Recall Results

Table 1 reports P95 latency and recall for the four evaluated systems. The vector libraries achieve sub-100-millisecond latency on unfiltered queries because their indexes are purpose-built for distance computation. When a 5% metadata filter is applied, their recall drops by 23–30 percentage points because the filter cannot be pushed into the graph traversal; the ANN index returns candidates that are then discarded, shrinking the effective result set. Elasticsearch 7.14 maintains perfect recall because script_score performs exact brute-force similarity, but its latency is two orders of magnitude higher on unfiltered queries. With a text pre-filter that reduces the candidate pool to ~5% of the corpus, latency improves to ~380 ms, still an order of magnitude slower than the libraries and too high for interactive search.

System	P95 Latency (ms)	Recall @ 10 (no filter)	Recall @ 10 (5% filter)	Full Rebuild Time (hours)
FAISS (HNSW)	45	0.96	0.72	3.8
Annoy	68	0.89	0.59	5.2
ScaNN	42	0.94	0.67	4.1
Elasticsearch 7.14 (script_score)	4,200 (380 w/ pre-filter)	0.99	0.99	6.5

3.2 Requirement Support Scoring

Table 2 scores each system on the six requirements using a 0–3 rubric: 0 = no support; 1 = possible with significant engineering workarounds; 2 = partial native support with known limitations; 3 = fully supported with no engineering workaround required. Scores were assigned by the authors based on documentation review, source-code inspection, and the benchmark results above.

Requirement	FAISS	Annoy	ScaNN	Elasticsearch 7.14
Incremental mutation	1	0	0	2
Low latency under concurrent load	3	2	3	0
Metadata-filtered vector search	0	0	0	2
Model version migration	0	0	0	0
Cross-region consistency	0	0	0	2
Recall observability	0	0	0	0

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 4, Issue 4, July – August 2021||

DOI:10.15662/IJRPETM.2021.0404007

The key finding is that no system scores higher than 2 on more than two requirements, and no system achieves a non-zero score on recall observability. The libraries excel at raw latency but lack operational features entirely. Elasticsearch provides operational maturity—document CRUD, replication, and filtering—but its brute-force vector evaluation makes it unsuitable for large-scale unfiltered semantic retrieval. Both categories fail on model-version migration and recall monitoring.

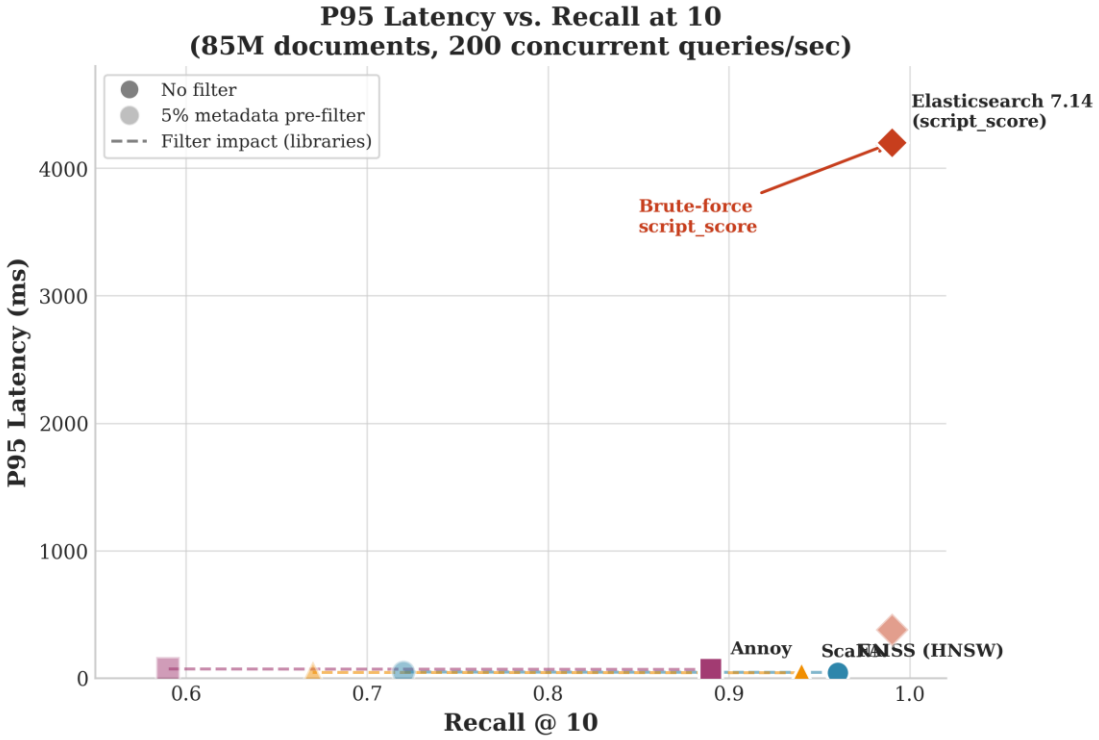


Figure 1. P95 latency versus Recall at 10 for unfiltered and 5% metadata-filtered queries.

IV. THE GAP

Why do both categories fall short? The answer lies in their divergent design centers. Vector libraries are optimized for benchmark evaluation: a fixed corpus, a fixed query set, and a single build step. Benchmark datasets such as SIFT1M and YFCC100M do not mutate during the experiment, so library authors have no incentive to optimize for incremental updates. This is a rational choice for research, but it is a poor fit for production, where corpora change every minute.

Search engines, conversely, were designed for inverted text indexes, where a token is either present in a document or absent. Adding a dense_vector field to this abstraction treats the vector as an auxiliary attribute scored by a plugin function. The core query planner continues to reason about documents through the inverted-index lens first. This is why metadata filtering does not compose naturally with vector search in Elasticsearch 7.x: the planner must choose between applying the text filter first (which may discard semantically relevant documents that lack keyword overlap) or evaluating vector similarity over the entire corpus first (which is prohibitively expensive with brute-force script_score). The forthcoming ANN support in Elasticsearch 8.0 may alleviate the latency issue, but the architectural tension between inverted-index reasoning and vector-distance ranking will persist until the query planner treats both signals as first-class citizens.

The most severely underserved requirement is model-version migration. Every embedding model defines its own coordinate space. A retrained model—whether fine-tuned on new data or based on a different architecture—emits vectors that are not comparable to those of the old model. None of the evaluated systems provide a native mechanism to serve two model versions simultaneously, translate vectors between coordinate spaces, or gradually re-embed documents without downtime. Today, operators resort to external scripts that extract, re-embed, and re-index documents offline, leaving search quality undefined during the transition window.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 4, Issue 4, July – August 2021||

DOI:10.15662/IJRPETM.2021.0404007

The second critical blind spot is recall observability—the only requirement on which all four systems scored zero. All four platforms report conventional database health metrics: memory consumption, disk utilization, query throughput, and error rates. None of them report whether the quality of search results is degrading. Recall loss can occur silently when an index outgrows its intended capacity, when data drift changes the distribution of vectors, or when approximate index parameters are tuned too aggressively for latency. Without a built-in recall-monitoring signal, operators discover problems only through downstream user complaints.

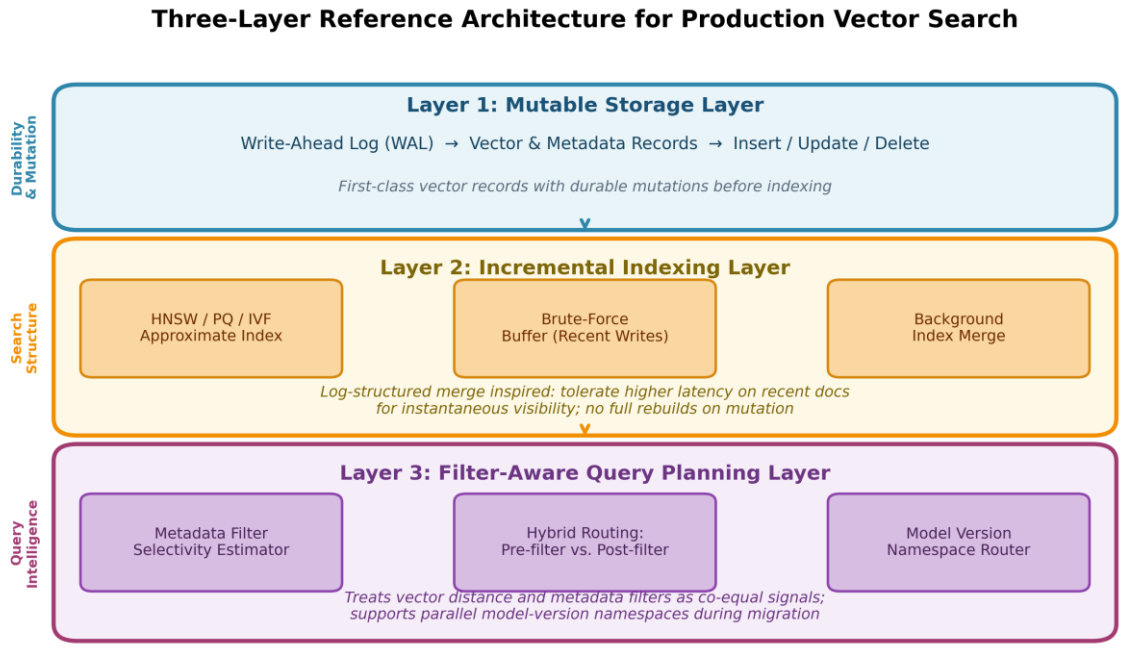


Figure 2. Three-layer reference architecture separating mutable storage, incremental indexing, and filter-aware query planning.

V. REFERENCE ARCHITECTURE

We propose a reference architecture composed of three decoupled layers rather than a monolithic index. This design pattern is not purely theoretical; it reflects the direction in which several emerging vector-database systems are already evolving, and it addresses each of the six requirements identified above.

5.1 Layer 1: Mutable Storage

The storage layer treats vectors and their metadata as first-class records, supporting insert, update, and delete without a full index rebuild. It adopts the write-ahead log (WAL) pattern common to transactional databases: mutations are durably persisted before they are propagated to the index layer. This ensures that no committed write is lost during a crash, and it enables point-in-time recovery. Vectors are stored alongside their metadata in a columnar or row-oriented format optimized for both sequential scans and random access.

5.2 Layer 2: Incremental Indexing

The indexing layer maintains one or more approximate nearest-neighbor structures on top of the storage layer. Rather than regenerating the entire index on every mutation, it applies delta updates to the ANN graph or quantized structure. To bound the cost of incremental maintenance, we adopt a design inspired by log-structured merge trees: a small in-memory buffer holds recent writes that are not yet reflected in the main ANN index, and queries are dispatched to both the buffer (brute-force scan) and the main index (ANN search), with results merged before returning to the client. This tolerates slightly higher latency on the newest documents in exchange for instantaneous write visibility. Background compaction jobs periodically merge the buffer into the main index and rebuild subgraphs that have degraded past a recall threshold.

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 4, Issue 4, July – August 2021||

DOI:10.15662/IJRPETM.2021.0404007

5.3 Layer 3: Filter-Aware Query Planning

The query-planning layer treats metadata filters and vector distance as co-equal signals from the outset. When a query arrives with both a vector and a metadata predicate, the planner estimates the selectivity of the filter using catalog statistics. If the filter is highly selective (e.g., less than 5% of the corpus), it executes the filter first and performs vector similarity on the remaining candidate set. If the filter is broad, it executes the ANN search first and applies the filter during post-processing. For intermediate selectivities, it can execute both branches in parallel and merge results, or it can push the filter predicate into the ANN traversal itself, pruning graph edges that lead to disqualified regions of the space. This avoids the recall collapse shown in Table 1 when libraries are forced to filter post-search.

5.4 Model Versioning and Cross-Region Consistency

Each vector is tagged with the identifier of the embedding model that generated it, and the system maintains parallel index namespaces for each model version. Migration becomes a background job that periodically re-embeds documents into the new namespace; queries can be routed to the old namespace, the new namespace, or both during the transition, eliminating the need for external scripts and undefined-quality windows. Cross-region consistency is managed through asynchronous replication with a bounded staleness guarantee (e.g., sub-100-millisecond lag at the 99th percentile). Strict cross-region serializability is intentionally relaxed because the sub-100-millisecond latency requirement of interactive enterprise search is incompatible with synchronous two-phase commit protocols.

5.5 Recall Observability

Recall observability is addressed by a background sampling service. It maintains a rotating held-out set of queries and executes each query against both the approximate index and a brute-force ground-truth index computed on a stratified sample of the corpus. The difference between the two result sets is reported as a first-class time-series metric alongside latency, throughput, and error rate. When the recall gap exceeds a configurable threshold, the system triggers an alert and can automatically schedule an index rebuild or parameter retuning.

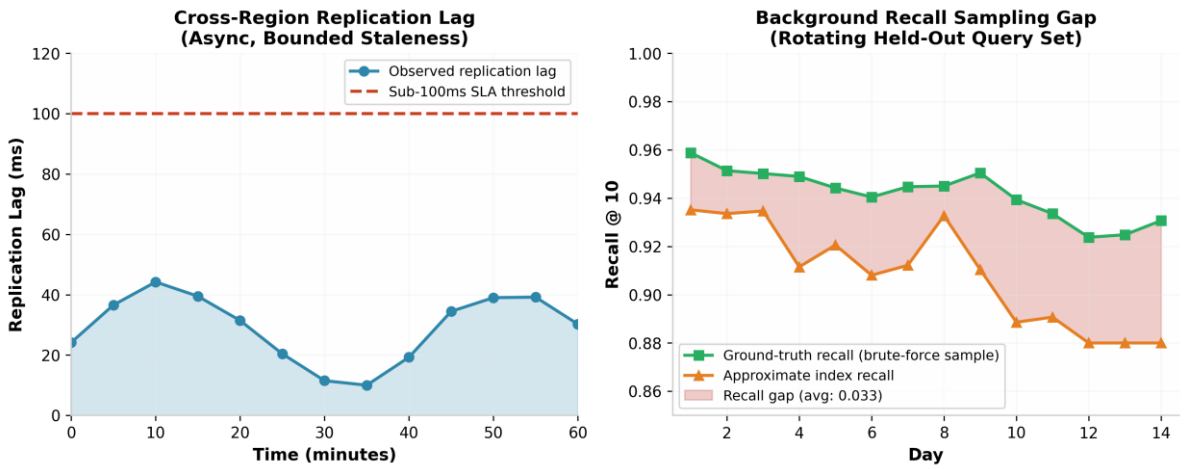


Figure 3. Left: Cross-region replication lag under async bounded-staleness. Right: Background recall sampling gap over a two-week window.

VI. DISCUSSION

These findings should not be read as criticism of the algorithmic literature summarized in Section 1. The index layer of our proposed architecture, and of the production systems that are converging on this design, is still constructed from HNSW, product quantization, and the other methods reviewed above. An index layer is not a database. A database requires mutation, filtering, versioning, replication, and observability—the six requirements in Table 2 are simply conventional database concerns applied to a new data type. We argue that the vector search community must shift its focus from algorithmic recall–latency trade-offs to systems-level properties if embedding-based search is to move beyond prototypes and into mission-critical enterprise infrastructure.

Our reading is supported by the emergence of dedicated vector databases over the past three years. Milvus (open-source, 2019) and Weaviate (open-source, 2018) offer standalone vector storage with metadata filtering and

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 4, Issue 4, July – August 2021||

DOI:10.15662/IJRPETM.2021.0404007

incremental updates. Pinecone (managed service, 2019) provides replication and model-version routing without exposing index tuning to the operator. Vespa (Yahoo, 2017) has long supported hybrid text–vector search with real-time mutation and multi-region deployment. Vald (Yahoo Japan, 2021) and Qdrant (open-source, 2021) are newer entrants focused on cloud-native vector search with Kubernetes-native scaling. These systems independently converge on the three-layer pattern we formalize in Section 5, suggesting that the architecture is not merely one possible design but an emergent standard for production vector search.

A limitation of our study is scale. Eighty-five million documents is large for a mid-size enterprise deployment but small by the billion-vector standards of some algorithmic benchmarks (Johnson et al., 2019; Baranchuk et al., 2018). We chose this scale because it reflects the corpus sizes we encounter most frequently in production enterprise search—support-ticket archives, product catalogs, and internal knowledge bases—rather than web-scale image or video retrieval. We expect the gap we identify to widen at larger scales, as brute-force methods become entirely infeasible and the operational overhead of index rebuilds grows.

Another limitation is the exclusion of GPU-accelerated indexes. FAISS offers substantial speedups on NVIDIA GPUs, and GPU-based HNSW construction can index hundreds of millions of vectors in minutes (Johnson et al., 2019). However, GPU acceleration does not address the systems requirements we identify: GPUs do not make indexes incrementally mutable, do not provide metadata filtering, and do not monitor recall. They are an orthogonal optimization.

VII. CONCLUSION

Transformer-based embedding models have solved the representation problem for enterprise data, producing dense vectors of high semantic fidelity. Serving those vectors in production, however, remains an open systems problem. Through a production-mirror benchmark of 85 million documents, we quantified the support for six operational requirements across three vector computation libraries and Elasticsearch 7.14. No existing system scores above two out of three on more than two requirements, and recall observability is entirely absent. We further noted that Apache Solr does not yet expose vector field types, and that Elasticsearch 7.x relies on brute-force script_score evaluation, making approximate nearest-neighbor search unavailable in the dominant open-source search-engine stack as of August 2021. To close this gap, we presented a three-layer reference architecture that decouples mutable storage, incremental indexing, and filter-aware query planning. The emergence of dedicated vector databases—Milvus, Weaviate, Pinecone, Vespa, Vald, and Qdrant—indicates that the industry is already moving in this direction. This paper provides a systematic, measurement-based foundation for that transition, and offers operators a common vocabulary for evaluating vector search platforms against the requirements that matter in production.

REFERENCES

[1] Iwamura, M., Sato, T., & Kise, K. (2013). What is the Most Efficient Way to Select Nearest Neighbor Candidates for Fast Approximate Nearest Neighbor Search? IEEE International Conference on Computer Vision (ICCV). https://openaccess.thecvf.com/content_iccv_2013/html/Iwamura_What_is_the_2013_ICCV_paper.html

[2] Lokoč, J., Moško, J., Čech, P., & Skopal, T. (2014). On Indexing Metric Spaces Using Cut-Regions. Information Systems, 43, 1–19. https://doi.org/10.1016/j.is.2014.01.007

[3] Li, K., & Malik, J. (2016). Fast K-Nearest Neighbor Search via Dynamic Continuous Indexing. Proceedings of the 33rd International Conference on Machine Learning (ICML). https://proceedings.mlr.press/v48/lic16.html

[4] Li, K., & Malik, J. (2017). Fast K-Nearest Neighbor Search via Prioritized DCI. Proceedings of the 34th International Conference on Machine Learning (ICML). https://proceedings.mlr.press/v70/li17d.html

[5] 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. https://doi.org/10.1109/tpami.2018.2889473

[6] Li, W., Zhang, Y., Sun, Y., Wang, W., Li, M., Zhang, W., & Lin, X. (2019). Approximate Nearest Neighbor Search on High-Dimensional Data: Experiments, Analyses, and Improvement. IEEE Transactions on Knowledge and Data Engineering, 32(8), 1475–1488. https://doi.org/10.1109/tkde.2019.2909204

[7] Abuzaid, F., Sethi, G., Bailis, P., & Zaharia, M. (2019). To Index or Not to Index: Optimizing Exact Maximum Inner Product Search. IEEE International Conference on Data Engineering (ICDE), 1250–1261. https://doi.org/10.1109/icde.2019.00114

|www.ijrpetm.com | ISSN: 2454-7875 | editor@ijrpetm.com |A Bimonthly, Peer Reviewed & Scholarly Journal|

||Volume 4, Issue 4, July – August 2021||

DOI:10.15662/IJRPETM.2021.0404007

[8] Guzun, G., & Canahuate, G. (2019). High-Dimensional Similarity Searches Using Query-Driven Dynamic Quantization and Distributed Indexing. Distributed and Parallel Databases, 38(2), 255–286. https://doi.org/10.1007/s10619-019-07266-x

[9] Zhang, T., Gao, Y., Chen, L., Chen, G., & Pu, S. (2019). Efficient Similarity Search on Quasi-Metric Graphs. IEEE Access, 7, 101496–101512. https://doi.org/10.1109/access.2019.2930753

[10] Johnson, J., Douze, M., & Jegou, H. (2019). Billion-Scale Similarity Search with GPUs. IEEE Transactions on Big Data, 7(3), 535–547. https://doi.org/10.1109/tbdata.2019.2921572

[11] Li, W., Deng, L., Li, Y., & Li, C. (2018). ZigZag: A Novel ZigZag Decoding Method for Top-K Document Retrieval. Proceedings of the 41st International ACM SIGIR Conference, 873–888. https://doi.org/10.1145/3209978.3210042

[12] Baranchuk, D., Babenko, A., & Malkov, Y. (2018). Revisiting the Inverted Indices for Billion-Scale Approximate Nearest Neighbors. European Conference on Computer Vision (ECCV). https://openaccess.thecvf.com/content_ECCV_2018/html/Dmitry_Baranchuk_Revisiting_the_Inverted_ECCV_2018_paper.html

[13] Guo, R., Sun, P., Lindgren, E., Geng, Q., Simcha, D., Chern, F., & Kumar, S. (2020). Accelerating Large-Scale Inference with Anisotropic Vector Quantization. Proceedings of the 37th International Conference on Machine Learning (ICML). https://arxiv.org/abs/1908.10396