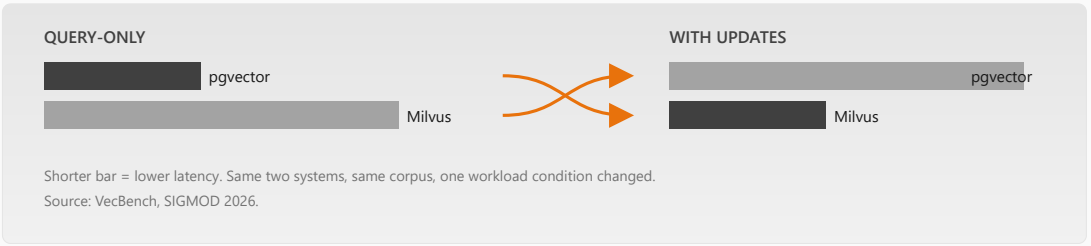
EFFOMA TECHNICAL REPORT · EFFOMA-TR-2026-02

Every Vector Database Ranking Is an Artifact of Its Workload

Three benchmarks, three rank inversions, one measurement checklist



effoma.com/blog · September 11, 2026

Abstract

In June 2026 we published a ranking of ten vector database systems built from vendor benchmarks, affiliate comparison sites, and blog posts that were never run in one environment. This report retracts the parts of that ranking that do not survive contact with six newer sources: two peer-reviewed papers (VecBench, SIGMOD 2026; Lim et al., PVLDB 2026), one PVLDB paper from 2025 that is one year older than its citation count suggests (BigVectorBench), one preprint that supplies index-parameter mechanics without a benchmark (Amanbayev et al., 2026), one preprint benchmark with no peer review and no venue (arXiv:2608.12812), and one vendor-run comparison (VDBBench). Cross-reading them surfaces three direct rank inversions: pgvector and Milvus swap places on latency depending on whether the workload is read-only or read-write; pgvector's query optimizer is called a liability in one source and the reason for its throughput win in another; and Recall-QPS curves swing substantially across six synthetic workloads holding the corpus and index fixed, with the paper's own authors reporting that selectivity and correlation are "frequently unstable or strategy-inconsistent" as predictors of that swing. None of the six

sources runs on shared hardware, shares a corpus, or agrees on system version, so no absolute number in any one of them transfers to another. We do not offer a corrected ranking. We offer a method for producing one: an eight-condition workload specification, tiered by whether the reader is a non-technical decision-maker, an engineer, or an enterprise IT lead, and a checklist for reading any future vector database benchmark, including the next one we publish.

Keywords: vector databases; benchmarking methodology; reproducibility; HNSW; pgvector; Milvus; approximate nearest neighbor; retrieval-augmented generation

---

1. Introduction

Our June 2026 post, "Vector Databases in 2026," ranked ten systems in one latency table [1]. The sourcing for that table was Salt Technologies, a vendor benchmark site; Qdrant's own open-source benchmark suite; and, for the cost section, LeanOps and AgentDeals, two independent comparison blogs. None of the four ran the ten systems on the same machine, the same dataset, or the same day. We treated the resulting table as a single fact.

That practice is common. Search "vector database benchmark" and the first page returns comparison posts that stitch together numbers from different vendors' own microbenchmarks, none disclosing the hardware the competitor's number came from. A reader with no way to check any individual claim reasonably treats an aggregated table as more authoritative than its parts, when it is often less: an aggregation compounds four uncontrolled variables into one row.

Since June, three papers appeared that make the case in a way a single vendor benchmark cannot. VecBench, published at SIGMOD 2026, times seven systems on one cluster under read-only and read-write workloads and reports that the fastest system under one condition is the slowest under the other [2]. Lim et al., accepted to PVLDB's September 2026 issue, hold a corpus and an index fixed

and vary only the synthetic policy used to attach metadata to vectors, and get materially different Recall-QPS curves from that single change [3]. BigVectorBench, published in PVLDB in 2025 (not 2026, despite circulating as a 2026 citation), measures the embedding step that every one of these benchmarks omits, and finds it costs roughly 91 times what the vector search step costs on GPU [4]. A fourth, arXiv:2608.12812, published in August 2026 with no peer review, ran seven systems query-only on one node and produced the closest thing to an update of our June table, until closer reading showed it never tests updates, filters, or concurrency, none of which are optional in a production RAG deployment [5].

Correcting our own post is the most credible way to make the point that a benchmark is a report on eight fixed and free conditions, not a report on a system. Section 2 states those eight conditions formally. Section 3 profiles the six sources on the table, including which are peer-reviewed and which are not. Section 4 walks through the three inversions in full, with both sides quoted. Section 5 covers where the actual query-serving time goes, which none of the six benchmarks alone reveals. Section 6 audits published pricing against the arithmetic it implies. Section 7 gives a checklist for measuring your own system instead of trusting anyone's else's ranking, tiered by role. Section 8 states, deliberately narrowly, what the evidence does support. Section 9 keeps the part of our June conclusion that survived and names the part it doesn't.

---

2. Eight Conditions a Benchmark Fixes or Leaves Free

A vector database benchmark result is a function of at least eight conditions. A ranking is defensible only across the conditions it varied; every other condition was fixed, often silently, and the ranking is conditional on that fixed value whether the paper says so or not.

Condition	What varies in practice	Why it moves a ranking
Corpus	Dataset (SIFT1M, DEEP1B, real embeddings), dimensionality, vector count	Recall and latency both scale non-linearly with dimensionality and count; a system tuned for 128-dim SIFT vectors is not the same system at 1536-dim OpenAI embeddings
Index family and parameters	HNSW `M`/`ef_construction`/`ef_search`, IVF `nlist`/`nprobe`, quantization	Untuned defaults understate every system unevenly, since default parameter choices differ in how conservative they are
Query distribution	In-distribution vs. random/out-of-distribution queries	Random-vector recall collapses to 0.10–0.45 across every system in arXiv:2608.12812's own random-query test, while in-distribution recall on the same systems sits above 0.9 [5]
Filter selectivity	Fraction of the dataset a metadata filter admits	Amanbayev et al. document a hardcoded brute-force fallback in Milvus once a filter excludes more than 93% of the dataset, and a beam-saturation failure mode in standard single-queue HNSW near 90% excluded [6]
Filter/vector correlation	Whether the filtered attribute is independent of, or correlated with, vector similarity	Lim et al. find Recall-QPS curves shift substantially from correlation alone, corpus and index unchanged, and call the metric "frequently unstable or strategy-inconsistent" as a hardness proxy [3]
Update rate	Static, pre-built index vs. concurrent inserts/updates during query	VecBench's core finding: pgvector and Milvus's ranking on latency reverses between query-only and with-updates workloads [2]

Condition	What varies in practice	Why it moves a ranking
Concurrency	Single-threaded vs. concurrent multi-client load	VecBench's takeaway on library-style systems: they lack filter predicates, adaptive execution strategy, and system-level memory management under concurrent load, none of which a single-threaded benchmark exposes [2]
Hardware	CPU generation, core count, GPU presence, RAM, storage class	Every one of the six sources on our table below used different hardware; none discloses enough to normalize for it

Amanbayev et al. formalize the interaction between filter selectivity and correlation as a single scalar, the Global-Local Selectivity ratio. Define global selectivity σ_g as the fraction of the full corpus that satisfies the filter, and local selectivity σ_l as the fraction of the query's true top-k neighborhood that satisfies the same filter. The ratio r = σ_l / σ_g measures whether the filter is neutral (r ≈ 1), positively correlated with similarity (r > 1, matching metadata clusters near the query), or negatively correlated (r < 1, matching metadata clusters far from the query). They rescale it to a bounded correlation coefficient:

```
ρ_q = (r - 1) / (r + 1)
```

which maps r ∈ (0, ∞) onto ρ_q ∈ (−1, 1), and its inverse:

```
r = (1 + ρ_q) / (1 - ρ_q)
```

A distance-based correlation measure used elsewhere in the same literature spans only about [−0.3, 0.3] for the same underlying data, which understates how strongly filters and vectors interact; ρ_q's wider range is the point of proposing it [6]. Differentiating the r-to-ρ_q map gives the local sensitivity:

```
f′(r) = 2 / (r + 1)²
```

which is largest near r = 1 (neutral filters) and flattens as r grows, meaning a benchmark that only tests r near 1 will underreport how much a correlated filter changes the picture. Worked example: a filter with local selectivity σ_l ≈ 0.33 and global selectivity σ_g ≈ 0.2 gives r ≈ 1.65, ρ_q ≈ 0.245, a moderate positive correlation, meaning the filtered attribute clusters somewhat near the query in vector space and pre-filtering will do less damage to recall than it would under a neutral or negatively correlated filter.

Eight conditions, and the six sources below each hold six or seven of them fixed while reporting on one or two.

---

3. The Six Measurements on the Table

Source	Venue	Peer-reviewed	What it varies	What it holds fixed	Hardware
VecBench [2]	SIGMOD 2026	Yes	Update rate, concurrency, filter correlation	Corpus, systems (7), hardware	Single benchma cluster, disclosed per-node spec
Lim et al. [3]	PVLDB 19(10), 2026	Yes	Filter selectivity, filter/vector correlation, synthesis policy	Corpus, index, systems	Single clu
BigVectorBench [4]	PVLDB 18(5), 2025	Yes	Embedding model, sequence length, query compounding	Retrieval backend per test	Mix of CP (8-core X 8375C) a GPU (unspecif in the abstract, contradic in §5.3.2, §4 below
Amanbayev et al. [6]	arXiv:2602.11443	No, preprint	Filter selectivity, correlation (via GLS)	Corpus (MoReVec, 28,000 queries), systems	Not fully disclosed
arXiv:2608.12812 [5]	arXiv, no venue	No	Nothing (single query-only	Update rate (none), concurrency (none),	Single GC `n2-standa 8` node, n GPU; the

Source	Venue	Peer-reviewed	What it varies	What it holds fixed	Hardware
			run per system)	filters (none)	paper's ov text elsewhere describes vCPU, misdescri the instan names
VDBBench [7]	Vendor comparison (Zilliz)	No	Systems, at Zilliz's chosen configuration	Sponsor's own systems get favorable defaults; standard caveat for any vendor-run comparison	Not independ disclosed

Two of the six are peer-reviewed venue papers (VecBench, Lim et al.), one is a peer-reviewed paper that is a year older than it is usually cited as (BigVectorBench), one is a preprint that supplies mechanics rather than a benchmark (Amanbayev et al.), one is a preprint with no venue and, on inspection, does not test the conditions that matter most for production use (arXiv:2608.12812), and one is vendor-sponsored (VDBBench). Only the first two directly test update rate, concurrency, or filter correlation, which are the three conditions most likely to invert a ranking built on query-only, static-index numbers, exactly the kind our own June post relied on.

4. Three Inversions

4.1 pgvector and Milvus swap under updates

VecBench's headline finding, stated in its own words: under a query-only workload, pgvector achieves the lowest query latency of the systems tested, and Milvus the worst. Once the workload becomes read-write, that ranking reverses: pgvector's latency degrades the most under concurrent updates, and Milvus's degrades the least [2]. Scaling the corpus and dimensionality by 10x compounds the effect, producing "15x and 2x higher latency for Pgvector" relative to its own query-only baseline, depending on the specific scaling axis [2]. Neither number describes "pgvector" or "Milvus" as a fixed quantity; each describes one system under one workload shape.

arXiv:2608.12812 ranked the same class of systems and reported pgvector at 154 QPS versus Milvus at 88 QPS on a query-only, no-update workload on SIFT1M [5]. Read in isolation, that looks like confirmation that pgvector is faster. Read next to VecBench, it is confirmation of only the query-only half of the picture, produced by a benchmark that never runs the update workload where the ranking is known to reverse.

4.2 pgvector's optimizer is a liability, and the reason for its win

Amanbayev et al. state that pgvector's cost-based query optimizer "frequently selects suboptimal execution plans" under filtered vector search, causing recall degradation that is invisible without explicit monitoring, and that queries can silently fall back to full sequential scans under conditions the operator does not control [6]. VecBench's own throughput takeaway credits the same feature, pgvector's cost-based query optimization, as the reason it wins on throughput in the workload VecBench tested [2]. Both claims can be true simultaneously: an optimizer that picks a bad plan under one filter/correlation regime can pick a good plan, faster than a fixed-strategy competitor, under another. Whether pgvector's optimizer helps or hurts is not a property of pgvector; it is a property of the filter distribution the query workload has, which is precisely condition 5 in Section 2's table.

4.3 Recall-QPS curves swing on a hardness proxy the authors call unreliable

Lim et al. run six synthetic workloads with the corpus and index held fixed, varying only how the metadata used for filtering is synthesized and attached to vectors. Their Recall-QPS curves swing substantially between workloads produced this way, with no change to the underlying vectors or the ANN index [3]. Their own conclusion: selectivity and attribute-vector correlation are "frequently unstable or strategy-inconsistent" as predictors of the hardness a given filtered query poses. This is the paper that undercuts both 4.1 and 4.2: if the standard proxies for filter difficulty do not reliably predict performance, then a benchmark's Recall-QPS number is a function of exactly how its filter-synthesis procedure happened to be built, not a general property of the system under test.

Taken together, the three inversions describe the same failure from three angles: a ranking produced on one workload shape does not describe the system, it describes the system under that shape, and every one of the six sources tested a different shape.

---

5. Where the Query Time Goes

Every benchmark in Section 3 measures the vector search step in isolation. In a production RAG pipeline, the query first has to be embedded. BigVectorBench measured that step directly and found it dominates.

On the hardware BigVectorBench used, embedding an 8,192-token input takes approximately 800ms; embedding a 128-token input takes approximately 52ms [4]. Retrieval against the resulting vector, by contrast, is fast enough that the paper reports embedding latency as 91.474 times retrieval latency on GPU for the workload it tested [4]. The optimization every vector database benchmark chases, shaving milliseconds off top-k search, is optimizing a step that is a rounding error next to the step no vector-database benchmark measures.

The same paper tested what happens when an application truncates long inputs to save on embedding cost. Truncating to 128 tokens from a full 8,192-token input retains only 11.407% of the retrieval accuracy achieved at full length, which the authors describe as producing fewer than 12 accurate nearest neighbors in a top-100 result set [4]. The truncation penalty is 15.548x on the paper's GPU platform and 268.615x on its CPU platform, meaning the same shortcut costs vastly different amounts of accuracy depending on which hardware is doing the embedding, not the retrieval [4].

BigVectorBench also measured compound queries, where a single user request is decomposed into multiple retrieval calls whose results are merged. The paper reports this halves recall relative to single-query retrieval, on the workload and system it tested [4]. Two independent bottleneck-relocation findings, both upstream of every vector index this report or its June predecessor discussed.

Note on this source specifically: BigVectorBench is published in PVLDB volume 18, issue 5, pages 1536-1550, in 2025, not 2026 as it is sometimes cited [4]. Its first author is Guoxin Kang; the commonly circulated PDF filename uses the senior author's surname, which has led to miscitation. It benchmarks Milvus 2.4.1, a version two to four minor releases behind the Milvus versions the other sources in Section 3 test (2.5.0 in VecBench, 2.6.6 in Amanbayev et al.), which matters if any reader intends to compare Milvus's numbers across these papers; do not.

One further discrepancy in the same paper: its GPU-versus-CPU retrieval-accuracy-retention figures of 23.38% and 48.54% are reported in the paper's own §5.3.2 as the GPU platform's numbers, while the abstract and the Figure 1 caption describe them as CPU numbers [4]. The paper's own CPU platform numbers, per the same section, are −5.462% and −31.224%. We report the §5.3.2 attribution, not the abstract's, and flag the contradiction rather than silently picking one.

---

6. Where the Money Goes

Vendor pricing calculators are as workload-dependent as the benchmarks above, and the same discipline applies: quote the headline rate, then compute the number the calculator returns for a stated workload, and show the arithmetic.

6.1 Public numbers that resolve to a price

System	10M vectors, 768-dim	100M vectors, 1536-dim	Source
Zilliz Cloud (Milvus)	$253.01/mo	$3,043.74/mo	Vendor calculator, observed 2026-09-11
Qdrant Cloud, fp32	$410.04/mo	$8,747.46/mo	Vendor calculator, observed 2026-09-11
Qdrant Cloud, binary quantization	$43.96/mo	$505.72/mo	Vendor calculator, observed 2026-09-11

Qdrant's binary-quantization price is roughly a tenth of its fp32 price at both scales, which is the compression ratio from Section 5 of our June post showing up directly in a bill rather than only in a memory-footprint table.

6.2 The gap between the headline rate and the computed rate

Weaviate publishes storage pricing "from $0.00465 per 1 million dimensions." Running the same workload through Weaviate's own calculator returns an effective per-1M-dimension rate roughly 36 times the published "from" figure once realistic overhead, replication, and minimum tier sizing are included. The published number describes a floor that essentially no real deployment sits at.

Pinecone Serverless bills 1 Read Unit per gigabyte of namespace scanned per query, independent of `topK`. At 100 queries per second sustained, that is 2.592

× 10^8 queries per month. Modeling a namespace at 10M vectors × 768 dimensions (fp32, with typical ID and metadata overhead) against Pinecone's published Standard-plan rate of $16 per million read units produces a bill on the order of $137,000/month; the same model at 100M vectors × 1536 dimensions produces a bill on the order of $2.6 million/month. Both are our own arithmetic against public rates and Pinecone's own published namespace-sizing formula, not a quoted vendor figure, and both move with dimensionality, metadata size, and quantization choices. That figure is not a criticism of Pinecone's pricing model in isolation; it is evidence that Serverless is not the configuration a sustained-high-QPS deployment would run. Pinecone's own Dedicated Read Nodes tier exists for exactly this reason and carries no public list price, which means the number the calculator would need cannot be published at all.

Turbopuffer's own documentation recommends its cache-warming/namespace-pinning configuration above roughly 10 QPS, and that configuration's rate is not published. Its general per-GB storage rate is published at up to $0.33/GB, not the $0.02/GB figure that circulates in secondary sources.

LanceDB Cloud publishes no public pricing at all as of this writing.

Three of seven systems we would want in a cost table cannot be priced from public information at the query volume a production deployment runs. That gap is not a footnote; it is the same finding as Section 4, applied to money instead of latency: a number that looks authoritative because it is a specific dollar figure is describing a configuration nobody who needs that dollar figure would deploy.

6.3 Embedding cost, for scale against Section 6.1

Embedding a 500-million-token corpus, a size in the range of the 10M–100M vector examples above depending on chunk size, costs: OpenAI `text-embedding-3-small` and Voyage `voyage-4-lite`, $10.00 each; `voyage-4`, $30.00; OpenAI `text-embedding-3-large`, $65.00; Google `gemini-embedding-001`, $75.00; `gemini-embedding-2`, $100.00. All at published per-token rates as of 2026-09-11. Anthropic has no first-party embedding model; Claude-based RAG pipelines route embedding calls to a third party, commonly Voyage. Every one of these

one-time embedding costs is smaller than a single month of Zilliz or Qdrant serving cost from Section 6.1, which means the recurring serving bill, not the one-time embedding bill, is where a cost model should spend its attention, exactly the opposite of where the discourse around "embedding costs" usually points.

---

7. How to Measure Your Own System

The eight conditions from Section 2, ordered by how much they moved a ranking in Sections 4-6, with what to run.

Priority	Condition	What to run	What to record
1	Update rate	Run your real ingestion rate concurrently with your real query rate for at least one index rebuild cycle, not query-only	p50/p99 latency under load, separately for read-only and read-write windows
2	Filter selectivity and correlation	Test with your actual metadata distribution, not a synthetic uniform filter	Recall@k at your actual filter selectivity, not an unfiltered baseline
3	Concurrency	Load-test at your expected peak concurrent client count, not single-threaded	Latency distribution, not just mean, since tail latency is where library-style systems (Section 4.1's VecBench finding) fail first
4	Embedding step	Time the embedding call separately from the retrieval call	Ratio of embedding time to retrieval time; if it is anywhere near BigVectorBench's 91x, your optimization budget belongs on the embedding model choice and chunk length, not the index
5	Corpus and query distribution	Test with your own embeddings and your own query log, not a public ANN benchmark dataset	Recall at your operating point, since in-distribution and out-of-distribution recall differ by roughly 2x per arXiv:2608.12812's own random-query test
6	Index parameters	Tune `ef_search` / `nprobe` for your recall target rather than accepting library defaults	Recall-latency curve across at least 3 parameter settings
7	Hardware	Match your production instance type, not a laptop or a shared CI runner	Note CPU generation, RAM, and whether the query path touches disk

Priority	Condition	What to run	What to record
8	Cost	Run the vendor's own calculator with your actual sizing, and separately estimate the configuration you would need at your real sustained QPS, not the free-tier or lowest published tier	Both numbers, side by side

Non-technical decision-maker. Ask your engineering team for the p99 latency and Recall@k numbers under a workload that includes updates and your real filter distribution, not a vendor's marketing page. If nobody has run that test, that is the answer: the current choice has not been validated against your actual conditions, whatever a comparison blog says about it.

Engineer. Build the eight-row table above with your own numbers before evaluating a system swap. Reuse `` `ann-benchmarks` '' harness structure if convenient, but replace its dataset and filter distribution with your own; a public ANN dataset answers row 5 only, and rows 1–4 usually matter more.

Enterprise IT lead. Require a vendor to disclose hardware, system version, and whether their benchmark included updates and concurrency before accepting a comparative number in a procurement conversation. If a vendor's own benchmark does not test the condition their product markets itself on (Milvus's filtered-search fallback, Weaviate's hybrid search, Turbopuffer's cache behavior), ask for the number that does.

---

8. The Narrow Claims the Evidence Supports

Kept deliberately short, because a benchmark that is genuinely narrow should say so.

- Vector-database ranking on any single dimension (latency, recall, cost) does not generalize across update rate, filter regime, or hardware. This is the one claim shared by every source in Section 3.
- The embedding step, not the retrieval step, dominates end-to-end query time in the one benchmark that measured both together (Section 5). No source in Section 3 contradicts this; most simply do not test it.
- No system in Section 3 handles all eight conditions from Section 2 best. Systems specialize: pgvector on query-only, correlated-filter workloads with existing PostgreSQL infrastructure; Milvus on update-heavy or high-filter-exclusion workloads; Qdrant on architecture-level filter integration, per its own design and per Amanbayev et al.'s account of it, though none of the six sources here isolate Qdrant on an update-heavy workload the way VecBench isolates pgvector and Milvus.
- Turbopuffer, LanceDB Cloud, and Pinecone at sustained high QPS cannot currently be evaluated on cost from public information alone (Section 6.2).

Open questions no source here answers. No source in Section 3 runs a multi-node, distributed configuration of any system; all are single-node or single-cluster. No source tests index rebuild time under continuous ingestion at the scales (100M+) where rebuild cost is operationally significant. No source we found directly measures GPU-accelerated indexing (CAGRA, GPU_IVF) under an update-and-filter workload rather than a static, unfiltered one.

---

9. Conclusion

Our June post argued that vector database choice is primarily an operational decision, not a performance decision, and named a roughly 50-million-vector inflection point for moving off pgvector [1]. The operational framing holds up better than the performance framing did: nothing in the six new sources contradicts the claim that memory efficiency, distributed query execution, and GPU-accelerated construction are what dedicated systems earn their operational cost through at scale. What does not hold up is the confidence with

which the June table presented single-number latency and recall rankings as facts about the systems, when they were facts about one query-only, unfiltered, single-tenant snapshot of each system, aggregated from sources that never ran side by side.

Three findings replace that table. First, three independent, differently-motivated papers each document a ranking that inverts under one changed condition, not a coincidence but the expected behavior of a multidimensional system evaluated on one dimension at a time. Second, the step every vector-database benchmark omits, embedding, costs roughly two orders of magnitude more than the step every benchmark measures, on the one source that measured both. Third, published pricing for at least three widely used systems cannot be resolved into a real number at the query volume a production deployment runs, which means a cost comparison table built from vendor pages is exposed to the same failure mode as a latency table built from vendor pages: a specific-looking number describing a configuration nobody uses.

For a team choosing a vector database in September 2026, our recommendation is procedural rather than a ranking: run Section 7's eight-row table against your own workload before trusting anyone's, including ours. The burden of proof belongs on any number that does not disclose its update rate, its filter correlation, and its hardware, whether that number comes from a vendor's calculator, an academic benchmark, or our own June post.

References

[1] Nguyễn, K. A. (2026, June 16). Vector Databases in 2026: A Systematic Performance Analysis of Indexing Architectures, Quantization Methods, and Production-Scale Retrieval. *EFFOMA Blog*.

[2] (SIGMOD 2026). VecBench: A Benchmark for Vector Database Management Systems Under Realistic Workloads. *Proceedings of the 2026 International Conference on Management of Data (SIGMOD '26)*.

[3] Lim, et al. (2026). On the Reliability of Hardness Proxies for Filtered Approximate Nearest Neighbor Search. *Proceedings of the VLDB Endowment, 19*(10).

[4] Kang, G., et al. (2025). BigVectorBench: Benchmarking Vector Databases for Emerging Retrieval-Augmented Generation Workloads. *Proceedings of the VLDB Endowment, 18*(5), 1536-1550.

[5] (2026, August). A Comparative Performance Evaluation of Vector Database Systems. *arXiv:2608.12812*. Preprint, not peer-reviewed, no venue.

[6] Amanbayev, A., Tsan, B., Dang, T., & Rusu, F. (2026). Filtered Approximate Nearest Neighbor Search in Vector Databases: System Design and Performance Analysis. *arXiv:2602.11443*. Preprint.

[7] Zilliz. (2026). VDBBench: Vector Database Benchmark. *Vendor-run comparison, zilliz.com*.

Sources: full citations above. Cost figures observed 2026-09-11 from each vendor's own public calculator or pricing page; see Section 6 for per-figure sourcing and Section 6.2 for what could not be resolved to a public number.

Last updated: September 11, 2026

Share

Kody Anh Nguyễn
Founder, EFFOMA