End-to-End Benchmarking Methodology for Vector Search Systems under Multi-Modal, Multi-Query Workloads

Hamza Qureshi[1] **and Imran Siddiq**[2]

[1]Department of Computer Science, University of Gujrat, Jalalpur Jattan Road, Gujrat 50700, Pakistan
[2]Department of Information Technology, The University of Haripur, Hattar Road, Haripur 22620, Pakistan

ABSTRACT

Vector search has become a common substrate for retrieval in applications that combine text, images, audio, and structured metadata. In practice, systems are evaluated under heterogeneous workloads that include nearest-neighbor lookups, filtered retrieval, hybrid sparse-dense ranking, and re-ranking stages that depend on model inference. Benchmarking these systems end-to-end is difficult because measured outcomes couple embedding quality, index construction, storage layout, distributed execution, caching, and concurrency control, while user-facing objectives trade latency, throughput, recall, and operational cost. This paper proposes an end-to-end benchmarking methodology for vector search systems under multi-modal, multi-query workloads. The methodology defines workload families that mix modalities and query operators, prescribes dataset construction that preserves cross-modal alignment and temporal drift, and introduces a harness that measures tail latency, recall at fixed budgets, stability under load, and cost-normalized performance across deployment topologies. It provides guidance for controlling confounders such as warmup effects, background compaction, and replica imbalance, and for reporting uncertainty using statistically grounded confidence intervals and sensitivity analyses. The methodology emphasizes reproducibility by specifying parameter disclosures, seeds, hardware and kernel settings, and failure-mode reporting. The result is a structured approach that makes benchmark outcomes interpretable as system-level trade-offs rather than isolated micro-metrics, and that supports comparative analysis across approximate nearest-neighbor algorithms, storage engines, and multi-stage retrieval pipelines without relying on workload simplifications that omit common production behaviors.

KEYWORDS

1. Introduction

Vector search systems implement similarity retrieval over learned representations, typically embedding functions that map objects into a metric space where geometric proximity approximates semantic relatedness [1]. While early evaluations often focused on approximate nearest-neighbor accuracy versus query time for a single modality and a single query primitive, contemporary deployments rarely match that simplification. Production traffic mixes modalities, includes metadata filters and access-control constraints, uses multiple embedding models that evolve over time, and frequently executes multi-stage retrieval pipelines such as candidate generation followed by re-ranking and post-filtering [2]. These realities create an evaluation problem in which the unit of measurement is not merely an index or an algorithm, but the entire system as deployed, including ingestion, indexing, query planning, caching, replication, background maintenance, and model inference. Under these conditions, isolated ANN benchmarks can misrepresent behavior because the

Article info:
Hamza Qureshi and Imran Siddiq. *End-to-End Benchmarking Methodology for Vector Search Systems under Multi-Modal, Multi-Query Workloads*. Nextqueries.

dominant costs shift from distance computations to memory bandwidth, network transfer, serialization, filter selectivity, and contention on shared resources.

An end-to-end methodology should separate what is intrinsic to the retrieval model from what is induced by the systems architecture, and it should allow benchmarking across systems with different internal designs while preserving comparability [3]. The central challenge is that vector search is not one workload; it is a distribution over query types, embedding distributions, and operational constraints. A method that only reports average latency at a fixed recall can miss the instability induced by load spikes, the degradation caused by index aging and incremental updates, or the changes in embedding geometry across modalities [4]. In addition, multi-modal workloads introduce alignment and calibration issues: image and text embeddings may live in a joint space but exhibit different norm distributions, anisotropy, or neighborhood densities, affecting both approximate search behavior and downstream ranking metrics.

This paper describes an end-to-end benchmarking methodology that targets multi-modal, multi-query workloads. The approach begins by formalizing workload composition and system interfaces, then specifies dataset and embedding construction, then defines a harness that orchestrates ingestion, indexing, query generation, load shaping, and measurement with strict control of confounders [5]. The methodology uses explicit budget constraints for latency, throughput, and cost, and treats accuracy as a family of measures that includes recall, ranking quality, calibration, and stability over time. Mathematical modeling is used to relate embedding geometry to index behavior, to quantify approximation error and its propagation through pipelines, and to define multi-objective optimization views that clarify trade-offs [**?**]. The intended outcome is a benchmark protocol that produces interpretable and reproducible results, enabling comparative evaluation across software stacks and hardware configurations without collapsing the workload into an unrepresentative toy problem.

2. Workload and System Model

A benchmark requires a workload model that is rich enough to cover production patterns while remaining parameterizable and repeatable. Let a corpus consist of N items, each item i containing one or more modalities drawn from a set $\mathcal{M}$ such as text, image, audio, and structured fields. For each modality $m \in \mathcal{M}$, an embedding function f_m maps raw modality content to a vector $x_i^{(m)} \in \mathbb{R}^{d_m}$. In joint-embedding systems, a shared space dimension d is used with $x_i^{(m)} \in \mathbb{R}^d$, while in heterogeneous systems a late-fusion stage may combine multiple spaces. Queries arrive as a stream with time index t; each query is characterized by a modality $m(q)$, an embedding vector y_q, optional structured predicates π_q over metadata, and an operator family ω_q such as k-nearest-neighbors, range search, maximal marginal relevance, or hybrid retrieval with a sparse component [6]. A multi-query workload is then a mixture distribution over operators and modalities, represented as $P(\omega, m, \pi, k, B)$, where B denotes budgets such as maximum latency, maximum CPU time, or maximum cost per query.

A system under test can be modeled as a pipeline with ingestion and query paths. The ingestion path includes embedding computation (possibly offline), vector normalization, optional compression, index build or incremental insert, and background maintenance such as graph refinement, segment merges, or quantizer updates [7]. The query path includes embedding computation for the query, query planning that selects an operator and index, candidate retrieval from one or more shards, optional pre-filtering or post-filtering, and a scoring stage that produces ranked results. The end-to-end latency decomposes into components that can be attributed but not assumed independent [8]. Let T_q denote the observed latency for query q, and let it decompose as

$$T_q = T_{\text{embed}}(q) + T_{\text{plan}}(q) + T_{\text{retrieve}}(q) + T_{\text{filter}}(q) + T_{\text{rerank}}(q) + T_{\text{ne} \tag{1}$$

where the terms represent embedding inference, planning, retrieval within the index, filtering costs, re-ranking, network transfer, and overhead such as serialization and scheduling. The benchmark should report not only T_q but also the distributional properties such as $p95$ and $p99$ under specified concurrency and background activity conditions, because tail behavior often dominates user experience [9].

Multi-modal workloads also require a representation of alignment and cross-modal similarity. If a joint space is used, similarity may be computed as cosine similarity $s(x,y) = \frac{x^\top y}{\|x\|\|y\|}$, dot product, or negative Euclidean distance. Geometry differs across modalities even within a shared model due to differing tokenization or feature extraction pipelines; anisotropy and norm variation can cause approximate search heuristics to behave differently for image versus text queries [10]. A benchmark should therefore include modality-conditional performance reporting and should shape query sets to cover low-entropy and high-entropy regions of the embedding space. One practical model for density is the local intrinsic dimension estimated from neighbor distance ratios, which can be used to stratify queries into easy and hard subsets without relying on external labels.

The workload model must incorporate filters and access constraints because these often dominate query planning [11]. If predicates π_q restrict the candidate set to a subset of items, the effective search problem becomes nearest-neighbor search over a conditional subset, which can be approximated by pre-filtering within the index, by maintaining per-filter indexes, or by retrieving candidates globally and post-filtering. Selectivity interacts with ANN hyperparameters, and there exists a regime where post-filtering drastically reduces recall because the candidate pool fails to contain enough valid items. A benchmark should specify filter selectivity distributions and correlate them with modalities, since real workloads often show such correlations, for example image queries being more common in browsing contexts with category filters [12].

3. Representations, Embedding Geometry, and Indexing Choices

End-to-end benchmarking must treat representation learning and index design as coupled, because embedding geometry

Table 1 Benchmark datasets used for multi-modal, multi-query evaluation.

Dataset	Modality	# Objects	Vector Dimensionality
MM-1M-ImgTxt	Image + Text	1,000,000	512 (img), 768 (txt)
MM-100K-Video	Video + Text	100,000 clips	1024 (vid), 768 (txt)
AudioCaps-500K	Audio + Text	500,000	256 (aud), 768 (txt)
WikiQA-1M	Text	1,000,000	768 (txt)
Ecom-MM-10M	Product + Text	10,000,000	512 (img), 768 (txt)

Table 2 Query workload taxonomy used in the end-to-end benchmark.

Workload	Description	Query Mix	Target Selectivity
Q1: Single-Vector	Single kNN over one modality	100% kNN	10^{-3}–10^{-2}
Q2: Cross-Modal	Join of two modalities (e.g., img→txt)	70% kNN, 30% filter	10^{-4}–10^{-3}
Q3: Hybrid	Vector + scalar predicates	60% kNN, 40% filter	10^{-5}–10^{-3}
Q4: Multi-Query Batch	Batched queries with shared modality	100% batched kNN	10^{-3}
Q5: Multi-Tenant	Mixed workloads across tenants	Heterogeneous	Tenant-specific

influences recall-latency trade-offs for ANN algorithms. Let the corpus embeddings form a matrix $X \in \mathbb{R}^{N\times d}$ in a shared space. Many embedding spaces exhibit anisotropy, where vectors concentrate in a narrow cone, leading to inflated dot products and reduced discriminative power [13]. A common remedy is centering and whitening, modeled by transforming vectors as $\tilde{x} = W(x-\mu)$ where μ is the empirical mean and W is derived from the covariance $\Sigma = \frac{1}{N}\sum_i (x_i-\mu)(x_i-\mu)^\top$. If $\Sigma = U\Lambda U^\top$ is an eigen-decomposition, a whitening transform can be $W = \Lambda^{-1/2}U^\top$ with truncation to control noise, which links directly to PCA and low-rank approximations. In benchmark design, applying such transforms changes the neighborhood structure and can improve or degrade ANN performance depending on the index. Therefore, a methodology should specify whether representation post-processing is allowed and, if so, treat it as a declared system component rather than an unreported pre-benchmark tweak.

Compression is often central to vector search scalability [14]. Product quantization and related schemes replace vectors with compact codes, changing the distance computation into a table-lookup approximation. Let x be partitioned into M sub-vectors of dimension d/M, each quantized to a codeword from a learned codebook [15]. The approximate squared distance becomes a sum of precomputed lookup values. Approximation error can be modeled as $d(x,y) = \hat{d}(x,y) + \varepsilon(x,y)$, where ε depends on quantization distortion. Under multi-modal workloads, distortion may be modality-dependent because embedding distributions differ; a single quantizer trained on pooled data can bias error toward the dominant modality. A benchmark should therefore consider both pooled and per-modality quantization training regimes and report distortion statistics stratified by modality [16].

Graph-based indexes such as hierarchical navigable small-world graphs rely on greedy search, whose performance depends on navigability properties related to the underlying metric and the graphs degree distribution. The expected search cost is not purely $O(\log N)$ or $O(N^\alpha)$ in practice; it depends on local expansion and on how well the graph approximates a proximity graph [17]. In multi-query workloads with filters, graph search can degrade because the greedy path may traverse filtered-out nodes, increasing exploration. Some systems handle this by storing filter bitmaps per node or by maintaining multiple graphs for common filters, both of which affect memory locality and update cost. Since the benchmark is end-to-end, it should include update and maintenance phases and measure how recall and latency drift as incremental inserts accumulate [18]. This is particularly relevant when embeddings change due to model refreshes; an index built with older embeddings may exhibit degraded neighborhood consistency relative to new query embeddings.

Hybrid retrieval integrates sparse lexical signals and dense semantic signals. One neutral model is a weighted similarity score [19]

$$S(q,i;\alpha) = \alpha\, s_{\text{dense}}(y_q, x_i) + (1-\alpha)\, s_{\text{sparse}}(q,i), \tag{2}$$

where $\alpha \in [0,1]$ balances modalities of evidence. End-to-end benchmarks should not treat α as fixed; instead, α can be tuned under constraints to maximize a task metric subject to latency budgets, or learned as a function of query features [20]. If a system performs adaptive weighting, the benchmark must include a reproducible policy for adaptation and report the sensitivity of outcomes to this policy. From an optimization standpoint,

Table 3 Vector search systems and index types evaluated.

System	Primary Index Type	Approximation	Notes
VS-A	HNSW	Graph-based ANN	In-memory, CPU-only
VS-B	IVF-PQ	Quantization-based ANN	GPU-accelerated
VS-C	DiskANN	Disk-aware graph	SSD-optimized
VS-D	Flat (BLAS)	Exact	Baseline, no ANN
VS-E	Product-Graph	Hybrid IVF+HNSW	Multi-modal aware

Table 4 Key index configuration parameters per vector search system.

System	Parameter	Default Value	Sweep Range
VS-A (HNSW)	`M`	32	8, 16, 32, 64
VS-A (HNSW)	`efSearch`	128	32–512
VS-B (IVF-PQ)	# Lists	4096	512–8192
VS-B (IVF-PQ)	PQ Subvectors	16	8, 16, 32
VS-C (DiskANN)	Beam Width	32	8–64
VS-E (Hybrid)	Fusion Weight λ	0.5	0.2–0.8

selecting α under multiple objectives can be expressed as

$$\min_{\alpha \in [0,1]} \mathbb{E}[T_q(\alpha)] + \lambda_1 \mathbb{E}[\text{Cost}_q(\alpha)] - \lambda_2 \mathbb{E}[\text{Quality}_q(\alpha)], \tag{3}$$

where λ_1, λ_2 define a weighted-sum trade-off; a benchmark can additionally report the Pareto frontier by sweeping α and other hyperparameters rather than presenting a single operating point [21].

When backprop-friendly derivatives matter, for instance in learned retrieval systems that incorporate differentiable approximations to top-k selection, the benchmark should specify whether training-time components are included. A practical end-to-end benchmark usually fixes models and evaluates serving-time behavior, but some systems fine-tune calibrations or quantizers online [22]. If differentiable components exist, an objective may incorporate smooth approximations to ranking losses, and constraints such as memory or energy budgets can be enforced via Lagrangians. If θ denotes tunable system parameters and $g_j(\theta) \leq 0$ denotes constraints like average latency minus a budget, a constrained formulation is

$$\min_{\theta} \mathcal{L}_{\text{quality}}(\theta) + \sum_j \lambda_j g_j(\theta) \quad \text{with} \quad \lambda_j \geq 0, \tag{4}$$

which supports principled exploration of trade-offs between quality and operational constraints within the benchmarks parameter sweep [23].

4. Benchmark Harness, Load Shaping, and Measurement Protocol

The benchmark harness is responsible for executing ingestion and query workloads, collecting measurements, and enforcing controlled conditions. For end-to-end comparability, the harness should define standardized phases: data preparation, ingestion and indexing, warmup, steady-state measurement, and stress or degradation scenarios such as bursty traffic and node failures. Each phase should be timed and logged, and background tasks such as compaction, graph maintenance, or cache eviction should be permitted if they are part of normal operation, but their occurrence must be observable to interpret tail effects [24].

A harness that supports multi-query mixtures should generate queries from a reproducible seed and should label each query with its type, modality, filter selectivity, and difficulty stratum. Difficulty can be estimated without labels by using oracle neighbors computed with an exact method on a subset or by using a high-recall configuration as a proxy oracle [25]. In practice, an oracle for N large can be computed by exact search on a sample, by brute force on GPU for a limited set, or by using a very conservative ANN setting, and then treating this as the reference for recall computation. The harness should avoid conflating system time with client-side time and should report both observed client latency and server-internal timings when available. Clock synchronization across nodes and clients is required; monotonic clocks and per-request identifiers should be used to correlate spans [26].

Load shaping must represent realistic concurrency. A common approach is to use an open-loop arrival process, where requests are generated at a target rate independent of response times, as opposed to closed-loop where clients wait for responses [27]. Open-loop better models external demand and reveals overload behavior. If arrivals follow a Poisson process with rate λ, then inter-arrival times are exponential, but real traffic often exhibits burstiness. A benchmark can incorporate burst models by modulating $\lambda(t)$ or using self-exciting processes, while remaining reproducible by seeding the generator [28]. Under multi-modal workloads, modality composition may shift with time; the harness should include scenarios where the mix-

Table 5 Hardware and deployment configuration for benchmark experiments.

Component	Specification	Value	Notes
CPU	Cores / Threads	32 / 64	x86_64, 2.8 GHz base
GPU	Model / Memory	1x A100 / 40 GB	Used by VS-B
Memory	RAM Capacity	256 GB	NUMA disabled
Storage	SSD	2 TB NVMe	Used by VS-C
Network	Link Speed	25 Gbps	Client–server mode
OS	Kernel Version	Linux 5.x	CFS scheduler

Table 6 Evaluation metrics reported in the end-to-end methodology.

Metric	Definition	Higher Is Better	Reported As
Recall@10	Fraction of top-10 ground truth retrieved	Yes	% (0–100)
NDCG@10	Discounted gain at rank 10	Yes	Unitless (0–1)
P95 Latency	95th percentile end-to-end latency	No	ms/query
P99 Latency	99th percentile end-to-end latency	No	ms/query
QPS	Sustained throughput under SLA	Yes	Queries/s
Cost/QPS	Cloud cost per unit throughput	No	USD per QPS

ture distribution changes, for example a switch from text-heavy to image-heavy traffic, to expose modality-specific bottlenecks in embedding inference and caching.

Measuring tail latency requires careful handling of outliers and timeouts [29]. The harness should set an explicit deadline per query and record whether results were returned before the deadline, partially returned, or failed. Systems that return incremental results or approximate partial answers should be benchmarked with a consistent policy, such as evaluating the best-effort results within the deadline and tracking quality degradation. Because vector search systems often incorporate caching, the harness should define cache state explicitly, including cold-cache runs, warm-cache steady state, and cache-thrashing scenarios [30]. Warmup should be long enough to stabilize JIT compilation, memory allocation patterns, and background index tuning, but the benchmark must also report the warmup cost since production often experiences cold starts after deployments or scaling events.

The harness should isolate measurement from environment noise as much as practical by pinning processes, fixing CPU frequency scaling policies, controlling NUMA placement, and documenting kernel and runtime settings. If GPUs are used for embedding inference or brute-force search, GPU clocks and power limits should be fixed and reported [31]. For distributed systems, network topology and bandwidth should be specified, and cross-zone latency should be included when relevant. When comparing systems, the harness should report resource utilization metrics such as CPU cycles, cache miss rates, memory bandwidth, disk IO, and network bytes, because two systems can have similar latency but different cost profiles [32]. A cost-normalized metric can be expressed as queries per second per dollar or per watt, but the benchmark should also include raw utilization to allow readers to recompute cost models.

Query planning in systems that support multiple indexes or multi-stage execution can be modeled as a decision problem. Let a plan p select an index configuration and a candidate size C, with expected latency $\mathbb{E}[T \mid p, q]$ and expected quality $\mathbb{E}[Q \mid p, q]$. Selecting p under a deadline D can be formulated as [33]

$$p^\star(q) = \arg\max_{p \in \mathcal{P}} \ \mathbb{E}[Q \mid p, q] \quad \text{s.t.} \quad \mathbb{E}[T \mid p, q] \le D, \tag{5}$$

which resembles a constrained optimization that may be solved via heuristics or learned policies. If the system uses dynamic programming for plan selection across stages, the benchmark should capture this overhead and its sensitivity to the number of candidate plans [34]. Since exact optimization can be NP-hard in general multi-stage selection problems when combining discrete choices and resource constraints, practical systems employ heuristics such as greedy selection, bandit tuning, or rule-based thresholds; the benchmark should treat the planner as part of the system and evaluate its stability under distribution shift in query mixtures.

Table 7 Per-modality performance on MM-1M-ImgTxt under workload Q2.

Modality	System	Recall@10 (%)	P95 Latency (ms)
Image → Image	VS-A	96.3	18.4
Image → Text	VS-E	94.8	22.1
Text → Image	VS-E	93.5	24.7
Text → Text	VS-B	97.1	15.9
Joint (Img+Txt)	VS-E	95.6	26.3

Table 8 Performance of multi-query execution patterns on Ecom-MM-10M.

Pattern	Description	P50 / P99 (ms)	Throughput (QPS)
MQ-1: Independent	Per-query kNN, no sharing	21 / 75	5,100
MQ-2: Batched	Fixed-size batches of 32	18 / 62	7,400
MQ-3: Fused Modalities	Joint img+txt queries	25 / 88	4,600
MQ-4: Cascaded	Coarse-to-fine reranking	27 / 93	4,900
MQ-5: Tenant-Grouped	Tenant-aware batching	20 / 70	6,200

5. Metrics, Statistical Rigor, and Approximation Error Analysis

End-to-end benchmarking requires metrics that correspond to user-facing outcomes and that allow meaningful cross-system comparison. For candidate-generation stages, recall at k relative to a reference set is common, but multi-modal, multi-query pipelines require more nuanced metrics because quality depends on filters, hybrid scoring, and re-ranking [35]. One approach is to report a vector of metrics: recall at k for filtered and unfiltered queries, ranking correlation with a reference ranking if available, calibration metrics for score distributions, and task-specific measures such as click-through proxies when labels exist. When labels are sparse, self-consistency measures can be used, such as agreement between independent retrieval methods or stability under small perturbations of embeddings.

Approximation error in ANN affects downstream ranking [36]. Suppose the true top-k set is $G_k(q)$ and the system returns $R_k(q)$. Recall is $\text{Recall@}k(q) = \frac{|G_k(q) \cap R_k(q)|}{k}$. In multi-stage retrieval, candidate generation returns C candidates and a re-ranker selects final results [37]. The quality impact depends on whether the true top items are included in the candidate set. If a downstream re-ranker would select item $i^\star$ given full access but $i^\star \notin R_C(q)$, the final quality loss can be significant even if recall at k is high for some k unrelated to C. Therefore, the benchmark should align recall measurement to the candidate sizes used in the pipeline and should measure end-to-end task metrics when possible [38].

To model approximation error, let similarity scores be approximated as $\hat{s}(q,i) = s(q,i) + \epsilon_{q,i}$ with random error $\epsilon_{q,i}$. If $\epsilon_{q,i}$ is mean-zero with variance depending on quantization and search truncation, then the probability of rank inversion between two items i and j depends on the score gap $\Delta_{ij} = s(q,i) - s(q,j)$. Under a Gaussian approximation, $\epsilon_{q,i} - \epsilon_{q,j}$ has variance $\sigma^2_{q,i} + \sigma^2_{q,j}$, yielding an inversion probability approximately

$$\mathbb{P}\big(\hat{s}(q,i) < \hat{s}(q,j)\big) \approx \Phi\left(\frac{-\Delta_{ij}}{\sqrt{\sigma^2_{q,i} + \sigma^2_{q,j}}}\right), \tag{6}$$

where Φ is the standard normal CDF. This connects embedding geometry to robustness: in dense neighborhoods with small gaps, even small approximation variance can reorder results [39]. A benchmark can estimate empirical gap distributions per modality and use them to interpret recall degradation and to motivate reporting of difficulty-stratified metrics.

Statistical rigor requires repeated runs and uncertainty reporting. End-to-end systems exhibit variance due to scheduling, cache state, network jitter, and background maintenance [40]. The benchmark should use multiple independent trials with different random seeds for query ordering and, when feasible, different placements of shards across nodes. Confidence intervals for mean latency are insufficient; tail quantiles require specialized estimators [41]. A practical approach is to report empirical quantiles with bootstrap confidence intervals, taking care to respect dependence induced by bursty traffic. If requests are correlated in time, block bootstrap methods can be used, where contiguous blocks of measurements are resampled. For throughput under overload, the benchmark should report achieved throughput versus offered load and identify the knee where latency diverges, rather than only reporting a single operating point [**?**].

When comparing systems, hypothesis testing should be complemented by effect size reporting. For example, if system A has $p99$ latency lower than system B by δ milliseconds under the same recall constraint, the benchmark should report δ and its uncertainty, along with the associated resource usage. Because benchmarks often compare tuned configurations, multiple-comparison issues arise [42]. A conservative methodology reports full parameter sweeps and emphasizes Pareto

Table 9 Ablation study of system VS-E on MM-1M-ImgTxt under workload Q3.

Variant	Change	Recall@10 (%)	QPS / Rel. Cost
Baseline	Full pipeline	95.6	4,000 / 1.0×
No Fusion	Disable cross-modal fusion	91.2	4,300 / 0.9×
No Caching	Disable result cache	95.4	2,600 / 1.1×
Light Rerank	Top-50 instead of top-200	94.1	4,800 / 0.8×
Heavy Rerank	Cross-encoder on top-500	96.8	2,100 / 1.4×

dominance rather than declaring a single winner. Let each configuration yield an objective vector $(T_{p99}, \text{Cost}, 1 - \text{Recall@}k)$. A configuration is Pareto-dominated if another configuration is no worse on all objectives and strictly better on at least one [43]. Reporting the non-dominated set provides a more robust comparison under varying priorities.

Bayesian-ish modeling can be used to pool information across query strata while preserving uncertainty. For example, modality-conditional recall can be modeled with a beta-binomial structure, where for modality m the number of hits h_m out of n_m opportunities has likelihood $\text{Binomial}(n_m, \rho_m)$ and prior $\rho_m \sim \text{Beta}(a, b)$. Posterior intervals for ρ_m then provide calibrated uncertainty that avoids overconfidence when n_m is small [44]. Similar hierarchical models can be used for latency quantiles by modeling log-latency with heavy-tailed distributions and sharing parameters across strata. The benchmark need not require complex inference, but it should encourage reporting that reflects uncertainty rather than single-point estimates [45].

6. Distributed Execution Mechanics, Storage Internals, and Performance Engineering

Vector search systems are often distributed by sharding the corpus across nodes and aggregating partial results. The distributed execution plan affects both latency and quality because approximate search within shards followed by global merge can miss true neighbors that reside in different shards if shard-level candidate pools are too small. Suppose the corpus is partitioned into S shards and each shard returns k_s candidates, merged to produce final top-k [46]. If k_s is too small, the probability that a true global top-k item is included depends on its shard rank. Benchmarking should therefore measure end-to-end recall in distributed mode, not only single-node recall, and should report how recall changes with shard count at fixed budgets.

Network and serialization overhead can dominate at scale [47]. Result merging requires transferring candidate identifiers and scores, and possibly vectors for re-ranking. If each candidate requires b bytes and total transferred candidates per query is $C_{\text{tot}} = \sum_s k_s$, then network bytes per query is approximately bC_{tot} plus protocol overhead. Under high QPS, this can saturate links and introduce queueing delays that inflate tail latency [48]. A benchmark should include measurements of network utilization and should test sensitivity to result payload size, for instance by toggling whether the server returns only IDs or also returns additional metadata. Because payload size interacts with compression and encoding, storage representation choices are part of end-to-end performance.

Storage internals influence recall and latency via memory locality [49]. Graph indexes and inverted lists have different access patterns; some favor sequential memory scans while others involve pointer chasing. Cache miss rates and TLB behavior become critical [50]. When vectors are stored in compressed form, decompression overhead can shift the bottleneck from memory bandwidth to compute. A benchmark should therefore capture hardware counter measurements when possible and should standardize on how such counters are collected to avoid instrumentation bias. If a system uses memory-mapped files, page cache behavior depends on OS settings and recent access patterns; benchmarks should document whether the page cache is warmed or cleared and should report file-system and block-device settings [51].

Update handling is another decisive factor. Many systems support incremental inserts and deletes, which can fragment indexes and degrade performance [52]. The benchmark should include an update workload interleaved with queries, with a specified update-to-query ratio and distribution over item popularity. Deletion semantics matter: tombstones, lazy deletion, and compaction influence both query-time filtering and background IO. Under multi-modal workloads, updates may be modality-dependent, for example image content being updated less frequently than text descriptions [53]. The benchmark should incorporate such heterogeneity because it affects background maintenance schedules and thus tail latency.

Concurrency control and scheduling policies can cause nonlinearities. If embedding inference and ANN search share CPU cores, contention can increase variance [54]. Some systems isolate inference on GPUs and search on CPUs, but then PCIe transfers and GPU queueing add new tail effects. The benchmark should define whether inference is included, and if included, should specify model batch sizes and batching policies [55]. Batching can improve throughput but harm tail latency for interactive workloads, especially under low load. Therefore, benchmarking should include both throughput-oriented and latency-oriented scenarios, with explicit arrival processes and deadlines.

From a complexity perspective, the benchmark should connect observed scaling to theoretical expectations while acknowledging constant factors [56]. For inverted-file with product quantization, a common complexity model for a query is $O(n_{\text{probe}} \cdot L \cdot M)$ where n_{probe} is the number of visited lists, L is the average list length scanned, and M is the number of sub-

quantizers. For graph search, complexity is often approximated as $O(E)$ where E is the number of visited edges, which depends on search parameters and graph degree. However, end-to-end latency is influenced by vector fetch costs and filter evaluations, which add terms proportional to candidate counts and predicate complexity [57]. The benchmark should record parameter settings such as n_{probe}, graph efSearch, and candidate pool sizes, and should report how latency scales with these parameters, enabling readers to distinguish algorithmic scaling from implementation bottlenecks.

When planning across indexes and tiers, systems sometimes use heuristics akin to approximation algorithms for constrained optimization. Selecting an index per query with per-query budgets resembles a knapsack-style problem when multiple resources are constrained, which is NP-hard in general when decisions are discrete and multi-dimensional. Practical planners use greedy rules, learned regressors, or bandit strategies [58]. A benchmark should include stress tests where query mix shifts, exposing whether planners overfit to a narrow distribution. It should also report planner overhead and misprediction rates, such as the fraction of queries that violate deadlines.

7. Reproducibility, Reporting Standards, and Evaluation Artifacts

A methodology is only as useful as its reproducibility [59]. End-to-end benchmarking should provide enough information for an independent party to rerun the evaluation and obtain comparable results within uncertainty bounds. This includes dataset versions, embedding model versions, preprocessing steps, index parameters, and full hardware and software environment descriptions [60]. Random seeds should be recorded for dataset splits, query generation, and any stochastic index training such as k-means for IVF or codebook learning for quantization. If the system includes approximate components with nondeterminism due to parallelism, the benchmark should measure run-to-run variability and report it rather than assuming determinism.

Reporting should include both absolute numbers and normalized comparisons [61]. Absolute metrics such as $p50$, $p95$, and $p99$ latency, achieved throughput, and recall at candidate size should be accompanied by resource usage such as CPU utilization, memory footprint, and storage size on disk. For distributed systems, the number of nodes, replication factor, shard sizes, and network characteristics should be reported [62]. Where cost is modeled, the benchmark should state the cost assumptions explicitly, such as on-demand pricing, depreciation schedules, or power costs, and should avoid presenting a single cost metric as universally applicable. Because users prioritize different objectives, reporting should emphasize trade-off curves and Pareto sets.

The benchmark should include failure-mode reporting [63]. Systems may exhibit query timeouts, partial results, or degraded recall under overload. These outcomes are part of real-world performance and should be recorded [64]. Similarly, background maintenance may cause periodic latency spikes; the benchmark should capture time-series traces rather than only aggregate summaries. If compactions, merges, or graph rebuilds occur, the benchmark should annotate the timeline with these events. This level of reporting prevents misinterpretation where a system appears stable due to averaging, while actually producing unacceptable spikes [65].

Evaluation artifacts should include machine-readable logs, configuration files, and scripts that reproduce the load generation and measurement. Where proprietary systems limit disclosure, the benchmark can still require a minimal set of disclosed parameters that affect performance, such as index hyperparameters and cache sizes. To ensure comparability, the methodology should specify a baseline harness implementation that can be adapted to different systems via an interface layer, minimizing the risk that one system is benchmarked with an optimized client while another uses a naive client [66]. The harness should validate correctness by checking that returned identifiers are valid, that filters are respected, and that scoring semantics match the declared similarity metric. For multi-modal workloads, correctness also includes verifying that modality-specific embedding functions are applied consistently and that cross-modal queries map into the intended space [67].

Finally, the benchmark should account for temporal drift. Embedding models may be updated, corpora evolve, and query distributions shift. An end-to-end methodology can include a time-sliced evaluation where the index is built on a snapshot, then updated with incremental data, while queries are drawn from corresponding time windows [68]. Measuring performance across time slices reveals whether the system degrades under incremental updates and whether calibration changes across model versions. While a full longitudinal study may be beyond a single benchmark run, the methodology should at least include an aging scenario with controlled inserts and deletes, and should report the resulting changes in latency, recall, and resource usage [69].

8. Conclusion

This paper presented an end-to-end benchmarking methodology for vector search systems evaluated under multi-modal, multi-query workloads. The methodology frames the benchmark as a system-level evaluation problem where embedding geometry, index structure, filtering, hybrid scoring, distributed execution, and background maintenance jointly determine user-facing outcomes. It specified a workload model that mixes modalities and query operators with explicit budget constraints, emphasized dataset construction that preserves cross-modal alignment and supports difficulty stratification, and described a harness that executes ingestion and query phases under controlled load shaping and cache-state scenarios [70]. It argued for reporting distributions and uncertainty rather than single-point metrics, and for interpreting results through approximation error models and multi-objective trade-offs that can be summarized by Pareto frontiers. It highlighted how distributed aggregation, storage internals, and update behavior influence end-to-end recall and tail latency, and it provided reproducibility and reporting standards aimed at making benchmark outcomes comparable across systems and deployments. The overall approach treats benchmarking as a disciplined measurement exercise that aims

to reduce confounding, expose stability and degradation modes, and support decision-making under competing operational objectives without assuming a single canonical workload or a single universally optimal operating point [71].

Acknowledgments

We would like to thank the reviewers of this research for their helpful comments and suggestions.

References

[1] A. B. Chernyshev, V. Antonov, and Z. Mayransaev, "Unified sectoral criterion for the stability of distributed systems," in *Proceedings of X All-Russian Scientific Conference "System Synthesis and Applied Synergetics"*, pp. 351–355, Southern Federal University, 10 2021.

[2] S. Soori, B. Can, M. Gurbuzbalaban, and M. M. Dehnavi, "Async: Asynchronous machine learning on distributed systems.," 7 2019.

[3] A. S. M. Noor, N. F. M. Zian, and F. N. M. S. Bahri, "Survey on replication techniques for distributed system," *International Journal of Electrical and Computer Engineering (IJECE)*, vol. 9, pp. 1298–1303, 4 2019.

[4] R. Malik, S. Kim, X. Jin, C. Ramachandran, J. Han, I. Gupta, and K. Nahrstedt, "Mlr-index: An index structure for fast and scalable similarity search in high dimensions," in *International Conference on Scientific and Statistical Database Management*, pp. 167–184, Springer, 2009.

[5] P. Jeswani, "Transparency and security issues in distributed system," 4 2018.

[6] D. Le-Phuoc and M. Hauswirth, *Semantic Stream Processing*, pp. 1505–1513. Springer International Publishing, 2 2019.

[7] O. V. Talaver and T. A. Vakaliuk, "Telemetry to solve dynamic analysis of a distributed system," *Journal of Edge Computing*, vol. 3, pp. 87–109, 5 2024.

[8] C. B. Jones, T. Haines, and R. Darbali-Zamora, "Hosting capacity considerations for the combination of wind and solar on distribution electric power systems subject to different levels of coincident operations," *Journal of Renewable and Sustainable Energy*, vol. 17, 11 2025.

[9] I. B. Fink, I. Kunze, P. Hein, J. Pennekamp, B. Standaert, K. Wehrle, and J. Rüth, "Advancing network monitoring with packet-level records and selective flow aggregation," in *NOMS 2025-2025 IEEE Network Operations and Management Symposium*, pp. 1–6, IEEE, 5 2025.

[10] Z. Lin, L. Cao, F. Ahmed, H. Lu, and P. Sharma, "When caching systems meet emerging storage devices: A case study," in *Proceedings of the 15th ACM Workshop on Hot Topics in Storage and File Systems*, pp. 37–43, ACM, 7 2023.

[11] T. Srinivasan, R. Chandrasekar, V. Vijaykumar, V. Mahadevan, A. Meyyappan, and A. Manikandan, "Localized tree change multicast protocol for mobile ad hoc networks," in *2006 International Conference on Wireless and Mobile Communications (ICWMC'06)*, pp. 44–44, IEEE, 2006.

[12] J. Pennekamp, R. Matzutt, S. S. Kanhere, J. Hiller, and K. Wehrle, "The road to accountable and dependable manufacturing," *Automation*, vol. 2, pp. 202–219, 9 2021.

[13] D. Olliaro, "Models for throughput maximisation in distributed systems," *ACM SIGMETRICS Performance Evaluation Review*, vol. 52, pp. 19–22, 1 2025.

[14] I.-A. Secara, *Challenges and Considerations in Developing and Architecting Large-scale Distributed Systems*, pp. 1–15. B P International (a part of SCIENCEDOMAIN International), 4 2023.

[15] C. Holland, Q. Nguyen, and J. Twitchell, "The economics of school bus fleet electrification with battery storage," in *2024 IEEE Electrical Energy Storage Application and Technologies Conference (EESAT)*, pp. 1–6, IEEE, 1 2024.

[16] K. Habibi, "Web service replica selection analysis using a multiagent-based simulator," 5 2021.

[17] L. Grote, I. Kunze, C. Sander, and K. Wehrle, "Instant messaging meets video conferencing: Studying the performance of im video calls," in *2023 7th Network Traffic Measurement and Analysis Conference (TMA)*, pp. 1–10, IEEE, 6 2023.

[18] A. Kolar-Poun, M. Jani, M. Rot, and G. Kosec, *Oscillatory Behaviour ofătheăRBF-FD Approximation Accuracy Under Increasing Stencil Size*, pp. 515–522. Germany: Springer Nature Switzerland, 6 2023.

[19] R. Chandrasekar, R. Suresh, and S. Ponnambalam, "Evaluating an obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 International Conference on Advanced Computing and Communications*, pp. 628–629, IEEE, 2006.

[20] H. E. Fazazi, M. Elgarej, M. Qbadou, and K. Mansouri, "Design of an adaptive e-learning system based on multiagent approach and reinforcement learning," *Engineering, Technology & Applied Science Research*, vol. 11, pp. 6637–6644, 2 2021.

[21] K. Grining, "Privacy-preserving protocols in unreliable distributed systems," 10 2020.

[22] F. FernándezBravoăPeñuela, J. ArjonaăAroca, F. MuñozEscoí, Y. YatsykăGavrylyak, I. IllánăGarcía, and J. BernabéuAubán, "Delta: A modular, transparent, and efficient synchronization of dlts and databases," *International Journal of Network Management*, vol. 34, 8 2024.

[23] M. Han, R. Chen, W. Shen, H. Zhang, J. Yang, and H. Chen, "Real-time, work-conserving gpu scheduling for concurrent dnn inference," *ACM Transactions on Computer Systems*, vol. 44, pp. 1–42, 11 2025.

[24] Z. Kolesnyk, O. Mezhenskyi, O. Davykoza, and H. Kuchuk, "Fog computing technology in distributed systems," *, '. *, vol. 1, pp. 94–97, 2 2024.

[25] E. B. Gulcan, J. Neto, and B. K. Ozkan, "Generalized concurrency testing tool for distributed systems," in *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pp. 1861–1865, ACM, 9 2024.

[26] M. Bisen, "Integration technology of distributed system based on multi-objective hotopy algorithm," *Distributed Processing System*, vol. 3, 7 2022.

[27] M. F. Rehme, S. Zimmer, and D. Pflüger, *Hierarchical Extended B-splines for Approximations on Sparse Grids*, pp. 187–203. Germany: Springer International Publishing, 10 2021.

[28] N. Benmoussa, A. Elyamami, K. Mansouri, M. Qbadou, and E. Illoussamen, "A multi-criteria decision making approach for enhancing university accreditation process," *Zenodo (CERN European Organization for Nuclear Research)*, 2 2019.

[29] *IM - Tofino + P4: A Strong Compound for AQM on High-Speed Networks?*, 5 2021.

[30] "Proceedings of the 1st international symposium on parallel computing and distributed systems," in *2024 International Symposium on Parallel Computing and Distributed Systems (PCDS)*, pp. 1–1, IEEE, 9 2024.

[31] S. Shankar, "The canonical controller for distributed systems," *Multidimensional Systems and Signal Processing*, vol. 32, pp. 303–311, 8 2020.

[32] K. Marszaek, A. Domaski, R. Cupek, J. C.-W. Lin, and A. Kwiecie, "A queueing system to jobs scheduling problem in agvs," in *2024 32nd Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, pp. 199–206, IEEE, 3 2024.

[33] C. Hanane, A. Battou, and O. Baz, "Performance security in distributed system: Comparative study," *International Journal of Computer Applications*, vol. 179, pp. 29–33, 1 2018.

[34] C. Marcelino, S. Gollhofer-Berger, T. Pusztai, and S. Nastic, "Cosmos: A cost model for serverless workflows in the 3d compute continuum," in *2025 IEEE International Conference on Smart Computing (SMARTCOMP)*, pp. 106–113, IEEE, 6 2025.

[35] D. G. Balreira, T. da Silva Araújo, and F. Petrillo, "Visualizing kubernetes distributed systems: An exploratory study," in *2023 IEEE Working Conference on Software Visualization (VISSOFT)*, pp. 12–22, IEEE, 10 2023.

[36] R. Chandrasekar and T. Srinivasan, "An improved probabilistic ant based clustering for distributed databases," in *Proceedings of the 20th International Joint Conference on Artificial Intelligence, IJCAI*, pp. 2701–2706, 2007.

[37] G. Sun, T. Alpcan, B. I. P. Rubinstein, and S. Camtepe, "A communication security game on switched systems for autonomous vehicle platoons," in *2021 60th IEEE Conference on Decision and Control (CDC)*, pp. 2690–2695, IEEE, 12 2021.

[38] M. Broy, *Designing Concurrent Distributed Systems by Interface Contracts*, pp. 234–250. Germany: Springer Nature Switzerland, 10 2025.

[39] M. Hauswirth, D. L. Phuoc, and J. X. Parreira, *Semantic Streams*, pp. 3419–3425. Springer New York, 12 2018.

[40] C. R. F. Campusano, "Distributed eventual leader election in the crash-recovery and general omission failure models.," 1 2020.

[41] B. Bollig and P. Gastin, "Non-sequential theory of distributed systems.," 4 2019.

[42] A. Labutkina, A. Selezneva, T. John, and D. Hausheer, "Multiobjective path optimization for deadline-aware multipath over scion," in *2024 IEEE International Conference on Machine Learning for Communication and Networking (ICMLCN)*, pp. 56–62, IEEE, 5 2024.

[43] X. Wang, B. Shi, and Y. Fang, "Distributed systems for emerging computing: Platform and application," *Future Internet*, vol. 15, pp. 151–151, 4 2023.

[44] R. Achar, P. Dawn, and C. V. Lopes, "Onward! - gotcha: an interactive debugger for got-based distributed systems," in *Proceedings of the 2019 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*, pp. 94–110, ACM, 10 2019.

[45] M. Senn and C. Cachin, "Asymmetric failure assumptions for reliable distributed systems," in *Proceedings of the 12th Workshop on Principles and Practice of Consistency for Distributed Data*, pp. 8–14, ACM, 3 2025.

[46] L. Czaja, *Failures and Damages in Distributed Systems*, pp. 157–185. Springer International Publishing, 1 2018.

[47] N. Evangelou-Oost, C. Bannister, and I. J. Hayes, "Contextuality in distributed systems," 1 2022.

[48] *FSTTCS - Dynamic Network Congestion Games.*, 10 2020.

[49] B. V. S. Pinto, D. R. Melo, C. A. Zeferino, E. A. Bezerra, and F. Viel, "Implementation of double sha-256 in hls for fpga using real bitcoin blocks," in *2025 17th Seminar on Power Electronics and Control (SEPOC)*, pp. 1–7, IEEE, 11 2025.

[50] E. Becks, M. Josten, V. Matkovic, and T. Weis, "Artifact: Revising poor man's eye tracker for crowd-sourced studies," in *2023 IEEE International Conference on Pervasive Computing and Communications Workshops and other Affiliated Events (PerCom Workshops)*, pp. 7–8, IEEE, 3 2023.

[51] V. Vijaykumar, R. Chandrasekar, and T. Srinivasan, "An obstacle avoidance strategy to ant colony optimization algorithm for classification in event logs," in *2006 IEEE Conference on Cybernetics and Intelligent Systems*, pp. 1–6, IEEE, 2006.

[52] J. Breuer, B. emusová, J. Fischer, J. Roztoil, and V. Vigner, *Synchronization of Distributed Systems using GPS*, pp. 95–120. River Publishers, 10 2024.

[53] J. Zhang, D. Du, L. Zhang, and Y. Xia, "Sprig: Low-latency startup for knative serverless platforms with asyncfork," in *2025 IEEE International Conference on Joint Cloud Computing (JCC)*, pp. 61–68, IEEE, 7 2025.

[54] A. Poshtkohi and M. B. Ghaznavi-Ghoushchi, *Advanced Operating System Concepts in Distributed Systems Design*, pp. 23–43. Auerbach Publications, 2 2023.

[55] W. B. Daszczuk, *DepCoS-RELCOMEX - Fairness in Temporal Verification of Distributed Systems*, pp. 135–150. Springer International Publishing, 5 2018.

[56] "Reliable networked and distributed systems," 4 2024.

[57] J. Raffety, B. Stone, J. Svacina, C. Woodahl, T. Cerny, and P. Tisnovsky, *Multi-source Log Clustering in Distributed Systems*, pp. 31–41. Germany: Springer Singapore, 4 2021.

[58] K. Orynbekova, S. Kadyrov, A. Bogdanchikov, and S. Oktamov, "A novel recommender system for adapting single machine problems to distributed systems within mapreduce," *Bulletin of Electrical Engineering and Informatics*, vol. 14, pp. 687–695, 2 2025.

[59] F. Lu, X. Wei, Z. Huang, R. Chen, M. Wu, and H. Chen, "Serialization/deserialization-free state transfer in serverless workflows," in *Proceedings of the Nineteenth European Conference on Computer Systems*, pp. 132–147, ACM, 4 2024.

[60] I. S. Ochôa, L. A. Silva, G. de Mello, B. D. Silva, J. F. D. Paz, G. V. González, N. M. Garcia, and V. R. Q. Leithardt, "Prichain: A partially decentralized implementation of ubipri middleware using blockchain.," *Sensors (Basel, Switzerland)*, vol. 19, pp. 4483–, 10 2019.

[61] S. Akbari and M. Hauswirth, "Universal workers: A vision for eliminating cold starts in serverless computing," in *2025 IEEE 18th International Conference on Cloud Computing (CLOUD)*, pp. 442–444, IEEE, 7 2025.

[62] U. A. Kuehn, "A decentralized iot payment service for digital currencies," in *2025 IEEE International Conference on Omni-layer Intelligent Systems (COINS)*, pp. 1–4, IEEE, 8 2025.

[63] Y. Braidiz, D. Efimov, A. Polyakov, and W. Perruquetti, "On robustness of finite-time stability of homogeneous affine nonlinear systems and cascade interconnections," *International Journal of Control*, pp. 1–11, 9 2020.

[64] D. E. Bouman, *Distributed Systems*, pp. 1204–1205. Springer International Publishing, 9 2018.

[65] P. Waibel, C. Hochreiner, S. Schulte, A. Koschmider, and J. Mendling, "Viepep-c: A container-based elastic process platform," *IEEE Transactions on Cloud Computing*, vol. 9, pp. 1657–1674, 10 2021.

[66] N. Nischal, "Automated evaluation for distributed system assignments," 1 2023.

[67] A. B. Library, *Leader Election in Distributed Systems*. 12 2019.

[68] R. Malik, C. Ramachandran, I. Gupta, and K. Nahrstedt, "Samera: a scalable and memory-efficient feature extraction algorithm for short 3d video segments.," in *IMMERSCOM*, p. 18, 2009.

[69] C. Elbaz, L. Rilling, and C. Morin, "Mesurer et prévenir l'évolution de la menace dans un cloud d'infrastructure," 6 2018.

[70] M. Zaccarini, F. Poltronieri, D. Borsatti, W. Cerroni, L. Foschini, G. Y. Grabarnik, D. Scotece, L. Shwartz, C. Stefanelli, and M. Tortonesi, "Chaos engineering based kubernetes pod rescheduling through deep sets and reinforcement learning," in *NOMS 2025-2025 IEEE Network Operations and Management Symposium*, pp. 1–7, IEEE, 5 2025.

[71] W. Yin, B. B. Zhu, Y. Xie, P. Zhou, and D. Feng, "Backdoor attacks on bimodal salient object detection with rgb-thermal data," in *Proceedings of the 32nd ACM International Conference on Multimedia*, pp. 4110–4119, ACM, 10 2024.