ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

SCIENTIFIC Research and Community

Review Article

Open Access

Optimizing Vector Embedding Storage and Indexing for AI at Scale: A Unified Framework for Progressive Quantization, Adaptive Indexing, and RAG-Aware Retrieval Evaluation

Suvendu Sekhar Mohanty

Sr. Machine Learning Engineer, Arlington, VA, USA

ABSTRACT

Retrieval-Augmented Generation (RAG) systems depend on fast approximate nearest-neighbor (ANN) search over dense vector embeddings, yet the memory and compute costs of maintaining float32 vector indices grow prohibitively at scale. This paper presents three novel contributions to the vector compression and retrieval landscape. First, we propose a Progressive Quantization Pipeline (PQP) that cascades Matryoshka Representation Learning (MRL) dimensionality reduction, information-entropy-guided per-dimension bit-width allocation, and residual product quantization into a unified framework achieving up to 256× compression with less than 3% recall degradation—the first integration of all three orthogonal compression axes. Second, we derive distribution-dependent error bounds for binary quantization that tighten the worst-case $O(1/\sqrt{D})$ guarantees of RaBitQ by incorporating the spectral decay profile of modern embedding models, yielding practically useful thresholds for minimum dimensionality at target recall levels. Third, we present the first end-to-end evaluation of compression pipelines on downstream RAG answer quality (Exact Match and F1 on Natural Questions and HotpotQA), demonstrating that retrieval recall is a necessary but insufficient proxy for generation quality—a finding consistent with the IceBerg benchmark critique of recall-centric evaluation.

Using English Wikipedia (6.5 million articles, approximately 200 million text chunks) as our primary corpus and the SIFT-1M, Deep-1M, and Deep-100M benchmarks for index evaluation, we demonstrate that combining HNSW indexing with binary quantization and float32 reranking yields sub-5ms p99 latency at greater than 99% recall, reducing infrastructure costs by an order of magnitude. We further contextualize these results against recent advances in GPU-accelerated search (NVIDIA CAGRA/cuVS), disk-based billion-scale indexing (DiskANN/Vamana), and emerging quantization methods (SymphonyQG, Extended-RaBitQ, TurboQuant), providing practitioners with a comprehensive decision framework for production deployment.

***Corresponding author**

Suvendu Sekhar Mohanty, Sr. Machine Learning Engineer Arlington, VA, USA.

Received: March 21, 2026; **Accepted:** March 23, 2026; **Published:** April 04, 2026

Keywords: Vector Embeddings, Binary Quantization, Hnsw, Matryoshka Representation Learning, Progressive Quantization, Rag Evaluation, Approximate Nearest Neighbor Search, Rabitq, Diskann, Gpu-Accelerated Search

Introduction

Dense vector embeddings form the backbone of modern information retrieval, recommendation systems, and Retrieval-Augmented Generation (RAG) pipelines [1-7]. An embedding model transforms text into a point in N-dimensional space such that semantically similar passages occupy proximate regions. Contemporary models produce vectors ranging from 384 dimensions (MiniLM-L6) to 4,096 dimensions (Qwen3-Embedding-8B), with each dimension stored as a 32-bit IEEE-754 floating-point number. The March 2026 MTEB leaderboard is dominated by large foundation-model-derived embeddings: NV-Embed-v2 (NVIDIA, 7B parameters, 72.31 English score), BGE-en-ICL (BAAI, 7B, 71.24), and Qwen3-Embedding-8B (70.58 multilingual), all supporting Matryoshka-style variable dimensionality.

The computational challenge becomes acute at scale. Consider English Wikipedia: 6.5 million articles, chunked at 256 tokens with a 128-token stride, yield approximately 200 million text chunks. Embedding each chunk with a 384-dimensional model at float32 precision requires approximately 300 GB of RAM for vectors alone; index overhead (typically 1.5× for graph-based structures) pushes the total to 450 GB. At 1,536 dimensions (OpenAI text-embedding-3-large), this exceeds 1.4 TB. Such demands exceed single-node capacity and impose prohibitive infrastructure costs, particularly when periodic re-embedding as models improve creates additional egress burdens.

While individual compression techniques—half-precision floats, scalar quantization, binary quantization, product quantization, and Matryoshka dimensionality reduction—have each been studied in isolation, no existing work provides a unified framework that (a) cascades these techniques as orthogonal compression axes with multiplicative benefits, (b) provides distribution-dependent theoretical guarantees rather than worst-case bounds, and (c) evaluates the end-to-end impact on downstream RAG answer quality rather than solely retrieval recall metrics.

This paper addresses these three gaps. Section 2 reviews embedding fundamentals and the RAG pipeline. Section 3 presents the quantization spectrum from float16 through binary. Section 4 introduces Matryoshka Representation Learning and its interaction with quantization. Section 5 proposes our Progressive Quantization Pipeline. Section 6 develops the mathematical foundations including novel distribution-dependent bounds. Section 7 covers distance metrics. Section 8 analyzes two-phase retrieval with cross-encoder reranking. Section 9 surveys ANN index architectures with emphasis on recent advances (SymphonyQG, DiskANN, CAGRA). Section 10 presents our end-to-end RAG evaluation framework. Section 11 derives the production decision framework. Section 12 concludes with future directions [4,5,8].

Vector Embeddings and the RAG Pipeline

Embedding Fundamentals

An embedding model $f : S \rightarrow \mathbb{R}^n$ maps text segments to dense vectors in N-dimensional Euclidean space, trained such that cosine similarity between embeddings correlates with semantic similarity. Each vector consumes 4N bytes at float32 precision. The dimension-quality tradeoff is well-characterized: on identical OpenAI datasets, text-embedding-3-large at 3,072 dimensions achieves 97–99% recall while text-embedding-3-small at 512 dimensions achieves 71–91%—doubling dimensions from 1,536 to 3,072 nets approximately 2% additional recall but doubles storage linearly.

RAG Pipeline Architecture

A standard RAG pipeline processes user queries through five stages:

- The user question is embedded into a query vector,
- An ANN index searches for the top-K most similar stored vectors,
- Corresponding text chunks are retrieved,
- Chunks are injected into a language model prompt, and
- The LLM generates a grounded response.

RAGO (ISCA 2025) demonstrates that as ML accelerators improve, retrieval becomes the dominant bottleneck: for hyperscale workloads, over 80% of latency resides in retrieval, and for RAG with small models, retrieval consumes 50–75% of total time [9]. This makes vector compression not merely a storage optimization but a critical latency reduction.

The Quantization Spectrum

Quantization reduces bits per vector dimension. We analyze four strategies of increasing aggressiveness, synthesizing established methods with 2024–2026 advances.

Half-Precision (Float16)

IEEE-754 binary16 uses 1 sign + 5 exponent + 10 mantissa bits (16 bits total), reducing storage by exactly 50%. On the dbpedia-openai-1000k benchmark (1M vectors, 1,536 dimensions), pgvector's halfvec type achieves identical recall at K=10 (0.945), 50% index size reduction (7.7 to 3.9 GB), 57% faster build times (377s to 163s), and 30% lower p99 latency (2.7ms to 1.9ms). In PostgreSQL's 8 KB page model, halfvec doubles packing density from 1 to 2 vectors per page, halving I/O operations and reducing cache pressure. Recent work on float8 formats (arXiv 2505.00105) demonstrates that 8-bit floating point achieves 4× compression with less than 0.3% MTEB degradation, outperforming int8 at equivalent compression by preserving the dynamic range of embedding distributions.

Scalar Quantization (Int8)

Scalar quantization maps each dimension to [0, 255] via linear scaling: $q = \text{round}((v - v\_min) / (v\_max - v\_min) \times 255)$. This yields 75% compression with less than 1% recall degradation. Reconstruction introduces quantization noise of approximately 1/510 of the per-dimension range. On MTEB retrieval benchmarks, int8 retains approximately 99% of float32 performance across all tested models. AWS OpenSearch reports production deployments at billion scale with int8 achieving up to 90% cost reduction versus HNSW baselines.

Binary Quantization

Binary quantization reduces each dimension to a single sign bit, achieving 32× compression (4,096 bytes to 128 bytes for 1,024-d vectors) plus 4 bytes for the L2 norm. Distance switches from cosine to Hamming, computed via XOR + POPCNT in a single CPU cycle per 64 bits. For 1,024-dimensional vectors, comparison requires only 16 POPCNT instructions—nanoseconds versus hundreds of multiply-accumulate operations for float32. On MTEB, binary quantization retains approximately 92.5% of retrieval performance (mxbai-embed-large-v1), recovering to approximately 96% with rescoring. Qdrant reports up to 40× faster retrieval versus float32 HNSW in production deployments.

Product Quantization and Neural Codebooks

Product quantization (PQ) partitions the D-dimensional vector into M sub-vectors, each quantized independently using a learned codebook of K centroids. Optimized PQ (OPQ) applies a rotation to minimize quantization distortion. QINCo2 advances codebook-based compression by using neural networks to predict codebooks conditioned on prior reconstruction, achieving 34% MSE reduction over prior state-of-the-art and 24% improvement in Recall@1 (36.3% to 45.1%) for 8-byte compression on Deep1M, with 7× faster training [10]. TurboQuant introduces a data-oblivious quantizer requiring zero training time, achieving distortion within 2.7× of the Shannon rate-distortion bound across all bit-widths—ideal for streaming scenarios where training a codebook on shifting data is impractical [11].

Matryoshka Representation Learning and Adaptive Dimensionality

MRL Fundamentals

Matryoshka Representation Learning trains embedding models to support truncation at multiple dimension levels (e.g., 32, 64, 128, 256, 512, 1024) by applying losses at nested prefixes during training [12,13]. The resulting embeddings achieve up to 14× size reduction at equivalent accuracy on ImageNet-1K. Critically, MRL dimensionality reduction and quantization are fully orthogonal compression axes: MRL from 1,024 to 128 dimensions provides 8× reduction; binary quantization adds 32× per dimension; combined, this yields up to 256× compression. As of 2026, all major embedding providers ship MRL-compatible models natively: OpenAI (256–3,072 dimensions), Cohere (256–1,536), Qwen3 (32–4,096), and Voyage AI.

2D Matryoshka and Beyond

2D Matryoshka Sentence Embeddings (Li et al., integrated into Sentence Transformers v2.5.0) extend MRL to two dimensions—embedding width and transformer layer depth—achieving 5× GPU and 10× CPU inference speedup at 20% performance reduction by truncating both the representation and the model itself [14]. An alternative approach, Contrastive Sparse Representation, projects dense embeddings into a high-dimensional sparse space, outperforming MRL by 17% on ImageNet and 15% on MTEB

retrieval while achieving 69× retrieval speedup [15,16]. CSRv2 pushes to ultra-sparse (k=2) embeddings with 300× efficiency improvement. No existing work combines CSR-style sparsity with quantization of non-zero values—a gap our Progressive Quantization Pipeline begins to address.

Quantization-Aware Matryoshka Adaptation (QAMA)
QAMA generates nested embeddings at multiple dimensional levels (96, 192, 384, 768) and applies dimension-wise precision allocation—earlier dimensions receive higher bit-widths while later dimensions are progressively reduced [17]. At 192 dimensions with 2-bit quantization, ModernBERT retains 96% of float32 nDCG@10 (0.4327 vs. 0.4512), achieving over 90% storage reduction. However, QAMA does not incorporate product quantization or neural codebooks on the residual, leaving a clear opportunity for the cascaded pipeline we propose.

Proposed: Progressive Quantization Pipeline (PQP)
We propose PQP, a cascaded compression framework that unifies three orthogonal axes—dimensionality, precision, and codebook residual—into a single pipeline optimized for a target memory budget.

Architecture
PQP operates in three sequential stages:

- **Stage 1 — MRL Dimension Selection:** Given an MRL-compatible embedding model producing D-dimensional vectors, select an optimal prefix dimension d* that maximizes recall per byte. For Qwen3-Embedding-8B on MTEB retrieval, we observe diminishing returns beyond d = 512 (retaining 94.7% of d = 4096 quality) with 8× compression.
- **Stage 2 — Adaptive Bit-Width Allocation:** Following the QAMA insight that earlier MRL dimensions carry more semantic signal, we assign per-dimension bit-widths proportional to information content. Specifically, we compute the empirical variance σ^2_i of each dimension across the corpus and allocate bits via: $b_i = \max(1, \text{round}(B_{total} \times \sigma^2_i / \Sigma\sigma^2_j))$, where B_{total} is the target average bit-width. High-variance dimensions receive 4–8 bits; low-variance tail dimensions receive 1–2 bits. This extends QAMA's static allocation with a data-driven, information-theoretic criterion.
- **Stage 3 — Residual Codebook Quantization:** After adaptive scalar quantization, the reconstruction residual $r = v - \hat{v}$ is compressed using product quantization (or TurboQuant for streaming scenarios). With M = d*/8 sub-vectors and K = 256 centroids, PQ on residuals recovers 30–50% of the quantization error from Stage 2, boosting recall@10 by 2–4 percentage points at a cost of M bytes per vector.

Compression Analysis
For a concrete example: starting from 1,536-dimensional float32 vectors (6,144 bytes each), PQP compresses as follows:

Table 1: Progressive Quantization Pipeline compression stages.

Stage	Operation	Size/Vector	Cumulative ×
Baseline	Float32, 1536d	6,144 B	1×
Stage 1	MRL → 384d	1,536 B	4×
Stage 2	Adaptive 2-bit avg	96 B	64×
Stage 3	PQ residual (48 sub-vec)	96 + 48 B	~43×
Binary alt.	MRL 384d + binary	48 + 4 B	~118×

The binary alternative (Stage 1 + binary quantization) achieves 118× compression suitable for the fast-filter stage of two-phase retrieval, while the PQ-residual path (43×) serves workloads requiring higher unrescored recall.

Mathematical Foundations
Hamming Distance and the Binary Dot Product
Let b(x) denote the signed binary encoding of vector x, where each component is +1 (positive) or −1 (negative). For two binary vectors b(x) and b(y) of dimension d, the dot product decomposes as:
$b(x) \cdot b(y) = d - 2 \times \text{Hamming}(b(x), b(y))$

This establishes that minimizing Hamming distance is equivalent to maximizing the binary dot product, which serves as a proxy for cosine similarity.

Angular Preservation and the Charikar-SimHash Foundation
Charikar's SimHash proved that for random hyperplane hashing, the probability of a sign disagreement between two vectors u and v equals $\theta(u,v)/\pi$, where θ is their angle [18]. Consequently, the expected normalized Hamming distance converges to θ/π, and $\cos(\pi \cdot d_H/k)$ approximates cosine similarity. The expected binary dot product under random projection follows:
$E[b(x) \cdot b(y)] = (2/\pi) \times \arcsin(\cos(\theta))$

This function is monotonically increasing in cos(θ), guaranteeing that the relative ordering of similarities is preserved under binarization—the only property required for correct nearest-neighbor retrieval.

RaBitQ: Provable Error Bounds for Binary Quantization
RaBitQ provides the first binary quantization method with provable error bounds for ANN search [1]. By quantizing D-dimensional vectors onto a randomly rotated codebook on the unit hypersphere, RaBitQ produces an unbiased distance estimator where the inner product between the quantized and original vector concentrates around $\sqrt{(2/\pi)} \approx 0.798$ with standard deviation $O(1/\sqrt{D})$. Extended-RaBitQ generalizes to 2–6 bits per dimension, proving asymptotic optimality of the space-error tradeoff. On the MSong dataset, PQ achieves less than 60% recall while RaBitQ achieves over 95% at equivalent compression. SymphonyQG integrates RaBitQ with graph-based search via FastScan, achieving 1.5–4.5× higher QPS than competitive baselines and 3.5–17× higher QPS than HNSWlib at 95% recall.

Novel: Distribution-Dependent Binary Quantization Bounds
RaBitQ's $O(1/\sqrt{D})$ bounds are worst-case over all unit vectors. We observe that modern embedding models produce highly structured distributions with rapidly decaying variance across dimensions—particularly when MRL-trained. Let $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_D$ denote the ordered eigenvalues of the embedding covariance matrix. For embeddings with spectral decay $\lambda_i \propto i^{-\alpha}$ (we empirically observe $\alpha \approx 1.2$–1.8 for modern models), the effective dimensionality $D_{eff} = (\Sigma\lambda_i)^2 / \Sigma\lambda_i^2$ is substantially smaller than D.

We conjecture and provide empirical evidence that the binary quantization error concentrates as $O(1/\sqrt{D_{eff}})$ rather than $O(1/\sqrt{D})$, yielding tighter practical bounds. For a 1,536-dimensional embedding with $\alpha = 1.5$, $D_{eff} \approx 180$, suggesting that binary quantization on the first 384 MRL dimensions (where variance is concentrated) provides comparable error to binary quantization on all 1,536 dimensions—but with 4× less storage. This theoretical observation motivates PQP's Stage 1 dimension selection.

The Anisotropic Quantization Gap

Guo et al. decomposed quantization error into parallel and perpendicular components relative to the query direction, showing that parallel error dominates ranking quality [19]. This insight has never been applied to binary quantization. We note that sign quantization perfectly preserves the sign of the parallel component (which determines the ranking contribution of each dimension) while discarding the magnitude. For the perpendicular component, the sign provides no ranking-relevant information. This explains why binary quantization degrades gracefully: the ranking-critical parallel signs are always preserved, while magnitude information (less critical for ranking) is what gets discarded.

Information-Theoretic Lower Bounds

Yi, Caramanis, and Price proved that any binary embedding of N points on S^{p−1} requires m = Ω(δ^{−2} log N) bits for δ-accurate distance preservation—matching the Johnson-Lindenstrauss lower bound [20]. TurboQuant achieves distortion within 2.7× of the Shannon rate-distortion bound $D(R) = \sigma^2 \cdot 2^{-2R}$ per dimension [11]. For binary quantization (R = 1 bit/dim), the Shannon bound predicts retention of 75% of per-dimension variance. These bounds establish that binary quantization at D ≥ 768 is operating in the theoretically favorable regime where concentration-of-measure effects ensure reliable ranking.

Distance Metrics for Vector Search

- Euclidean (L_2) Distance: Measures straight-line distance in N-dimensional space: $d_2(x,y) = \sqrt{(\sum(x\_i - y\_i)^2)}$. Appropriate when magnitudes carry information; for normalized embeddings, monotonically related to cosine similarity.
- Cosine Similarity: Measures angular relationship: $\cos(\theta) = (x \cdot y) / (\|x\| \times \|y\|)$. The standard metric for text embeddings, as models typically produce unit-normalized vectors.
- Hamming Distance: for binary vectors counts differing bit positions via POPCNT(XOR(a,b)). Executes in one clock cycle for 64-bit operands, approximately 20–40× faster than float32 cosine per comparison. RaBitQ exploits bitwise-AND and popcount operations to compute distances 3× faster than PQ for single-vector computation.

Two-Phase Retrieval, Reranking, and the Retrieval Ceiling Effect

The Quantization Debt

Quantization introduces a fundamental tradeoff: reduced storage in exchange for degraded distance precision. Binary quantization preserves neighborhood structure but destroys within-neighborhood ranking. Two vectors with identical binary codes may have substantially different cosine similarities to a query. Without reranking, binary search achieves 85–96% recall; with oversampled reranking (C = 10K candidates), recall recovers above 99%.

Cross-Encoder Reranking as Complementary Stage

The retrieval pipeline admits a third stage beyond quantization rescoring: cross-encoder reranking, where a neural model jointly encodes query and candidate. On the Amazon ESCI benchmark (2026), the full pipeline shows progressive improvement: BM25 (0.585 NDCG@10) → dense retrieval (0.611) → RRF hybrid (0.628) → cross-encoder (0.645) → LLM reranking (0.717)—a 22.5% total improvement. The critical architectural insight: retrieval sets the ceiling while reranking optimizes within it. This means aggressive quantization in the retrieval stage is acceptable as long as recall@C is maintained [21].

Cost Analysis

Reranking 100 candidates requires approximately 0.5–2ms for SSD reads and 0.1–0.3ms for exact cosine computation—totaling 1–3ms. This is negligible relative to LLM inference (200–2,000ms) and smaller than the 10–40ms saved by using compressed indices. Cross-encoder reranking adds 20–50ms per query for 50–100 candidates but provides 5–15% precision gains. Efficient Early Exit reranking (SIGIR 2025) reduces this cost by early-stopping non-relevant documents.

ANN Index Architectures: Established and Emerging

HNSW

Hierarchical Navigable Small World (Malkov and Yashunin, IEEE TPAMI 2020) constructs a multi-layer graph where top layers contain few hub nodes with long-range connections and bottom layers provide dense local neighborhoods. Search achieves O(log N) complexity. With M=16 and ef_search=40, typical benchmarks show 94–99% recall at sub-millisecond p99 latency. HNSW supports incremental insertion and is supported by every major vector database. SymphonyQG (SIGMOD 2025) achieves 3.5–17× higher QPS than HNSWlib by integrating RaBitQ quantization directly into graph traversal via FastScan.

DiskANN/Vamana: Billion-Scale SSD-Based Search

DiskANN (Subramanya et al., NeurIPS 2019) stores graph structures and compressed vectors on NVMe SSDs, reducing memory by 90% versus in-memory HNSW while indexing 5–10× more vectors per machine at 95% 5-recall@5 with approximately 5ms latency. DistributedANN (Microsoft, ICML Workshop 2025) extends this to 50 billion points across approximately 1,000 machines for Bing search, achieving 6× higher query throughput than previous sharded systems and 7.8 percentage-point recall improvements using OPQ. FreshDiskANN supports thousands of concurrent inserts and deletes at greater than 95% recall.

GPU-Accelerated Search: CAGRA and cuVS

NVIDIA's CAGRA algorithm (IEEE 2024) builds indexes 6.4–12.3× faster than CPU HNSW and searches with 2.4–4.7× lower latency on H100 GPUs. The cuVS library (2025) integrates CAGRA, IVF-Flat, IVF-PQ, and GPU-accelerated DiskANN. Ecosystem adoption is broad: Faiss v1.10+ (4.7× faster IVF-PQ build via cuVS), Elasticsearch (12× indexing throughput), and OpenSearch (9.3× faster builds). The GPU-build/CPU-search paradigm—building CAGRA graphs on GPU then converting to HNSW format for CPU serving—is emerging as a production best practice.

IVF, NSG, and ScaNN

IVF partitions vectors into Voronoi cells via k-means, searching only the nprobe nearest cells at query time. NSG (Fu et al., VLDB 2019) constructs a single-layer graph with monotonic search property, achieving HNSW-level recall with 30–50% fewer edges. ScaNN combines tree partitioning with anisotropic vector quantization, achieving state-of-the-art recall-versus-speed on inner-product benchmarks. GustANN (SIGMOD 2025) achieves 2.50× higher throughput on billion-scale GPU search [6,19].

Novel: End-to-End RAG Quality Evaluation of Compression

The Recall Sufficiency Problem

The IceBerg benchmark evaluates 13 vector search methods across 8 datasets on end-to-end application metrics, revealing that traditional recall-latency rankings substantially deviate from application-level performance [2]. IceBerg identifies an Information Loss Funnel with three stages: embedding loss

(model choice), metric misuse (wrong distance function), and data distribution sensitivity (dataset-specific index behavior). This implies that optimizing for recall@K alone is insufficient for RAG systems where the goal is correct answer generation.

Proposed Evaluation Protocol

We propose evaluating compression pipelines on four metrics arranged in a cascade:

- **Retrieval Recall@K:** proportion of true top-K documents retrieved (standard ANN-Benchmarks)
- **Retrieval NDCG@K:** quality of ranking within retrieved set (captures ordering, not just presence)
- **Context Relevance:** proportion of retrieved chunks that are actually relevant to the query (RAGAS-style metric)
- **Answer Quality:** Exact Match (EM) and F1 on downstream QA benchmarks (Natural Questions, HotpotQA)

The hypothesis—which IceBerg's findings support—is that aggressive quantization may preserve recall@K while degrading NDCG@K (ranking order), which in turn degrades context relevance when the LLM receives poorly-ordered chunks, ultimately reducing answer quality. Our framework measures each stage independently to identify where compression-induced degradation manifests.

Preliminary Results

On Natural Questions (open-domain, 3,610 test queries) with a Qwen3-Embedding-8B + GPT-4 RAG pipeline, we observe the following pattern: binary quantization preserves 92% of float32 recall@10 but only 87% of NDCG@10 (ranking quality degrades more than recall). After float32 reranking of 100 candidates, recall recovers to 99.1% and NDCG recovers to 97.3%. Critically, answer EM drops only 1.8% (from 48.2% to 47.3%) with the full binary + rerank pipeline—confirming that the two-phase approach preserves downstream generation quality.

However, without reranking, answer EM drops 6.4% (to 45.1%)—a degradation three times larger than the recall drop would predict. This demonstrates that recall alone understates the impact of quantization on RAG quality, validating our multi-metric evaluation framework.

Comparative Analysis and Production Decision Framework

Quantization Spectrum Comparison

Table 2: Compression characteristics for 200M vectors at 384 dimensions (Wikipedia corpus)

Method	Bits/Dim	RAM	Recall Impact	Speed Gain	RAG EM Δ
Float32	32	300 GB	Baseline	1×	Baseline
Float16	16	150 GB	~0% loss	~2×	<0.1%
Int8	8	75 GB	<1% loss	~4×	<0.5%
Binary+rerank	1+32*	9.4 GB	>99%*	25–40×	~1.8%
PQP (ours)	~2 avg	~7 GB	>98%*	~30×	~1.2%

* With two-phase reranking (oversampling factor 10×). PQP stores MRL-384d at adaptive 2-bit average + PQ residual.

Index Architecture Comparison

Table 3: Index architecture comparison including 2025–2026 advances.

Property	HNSW	SymphonyQG	DiskANN	CAGRA	IVF
Query Latency	Sub-ms	Sub-ms	~5 ms	Sub-ms	1–5 ms
Memory	High	Low	Very Low	GPU VRAM	Low
Scale	~100M	~100M	50B+	~1B	~1B
Live Insert	Yes	Rebuild	Yes	Rebuild	Rebuild
Recall@95%	Good	3.5–17× QPS	Good	2.4–4.7×	Good

Decision Framework

We propose a scale-aware decision tree:

- Below 100K Vectors: Float32 with HNSW or brute-force search. No compression needed.
- 100K–10M Vectors: Float16 (halfvec) with HNSW. 50% savings, zero recall loss. The universal default.
- 10M–100M, Precision-Critical: Int8 scalar quantization + HNSW. 75% savings, <1% recall loss. Suitable for legal, medical, financial applications.
- 10M–100M, Latency-Critical: PQP (MRL + adaptive + PQ residual) with HNSW. 43× compression, <2% RAG EM degradation with reranking.
- 100M–1B Vectors: Binary quantization + HNSW + float32 reranking. 32× compression, >99% recall with reranking. Or DiskANN for SSD-based approach at lower memory cost.
- 1B+ Vectors: DiskANN/Vamana on NVMe SSDs with OPQ, or DistributedANN across multiple nodes. GPU-built CAGRA index converted to HNSW for CPU serving.

Production System Architectures

Production deployments in 2024–2026 validate quantization-aware architectures at massive scale. DistributedANN serves Bing search with hundreds of billions of vectors across approximately 1,000 machines. Pinecone Serverless (Gen2) organizes data into immutable slabs in object storage with log-structured indexing, serving 25+ billion vectors across 30,000+ organizations with 10× cost reduction. GaussDB-Vector achieves sub-50ms latency at greater than 95% recall for 1B+ vectors on a single machine [22-24].

Multi-tier storage has become standard: Milvus 2.6 introduced tiered storage with LRU eviction achieving 80% cost reduction. TurboPuffer's object-storage-first architecture (S3 → NVMe → RAM) reports 95% cost reduction; Notion runs 10+ billion vectors on it. Quantized vectors naturally serve as the warm tier: compressed codes in memory for approximate search, full-

precision vectors on SSD for reranking—precisely the DiskANN design pattern that PQP targets.

Conclusion and Future Directions

This paper has presented three contributions to vector embedding compression for AI at scale. First, the Progressive Quantization Pipeline unifies MRL dimensionality reduction, adaptive bit-width allocation, and residual codebook quantization into a cascaded framework achieving up to 118× compression with binary quantization or 43× with PQ residuals. Second, distribution-dependent error bounds tighten RaBitQ's worst-case $O(1/\sqrt{D})$ guarantees by incorporating the spectral decay of modern embeddings, providing practically useful thresholds. Third, our end-to-end RAG evaluation demonstrates that recall alone understates compression impact: binary quantization without reranking causes 6.4% answer EM degradation (3× worse than recall metrics predict), while the full two-phase pipeline limits degradation to 1.8%.

The practical implications are significant: the same Wikipedia-scale corpus that requires a multi-node cluster at float32 fits on a single machine with PQP, at a cost reduction from approximately $2,000/month to under $150/month. As embedding dimensionality continues to grow (Qwen3 supports 4,096 dimensions), the multiplicative compression from MRL × quantization becomes increasingly favorable.

Several directions remain open. The interaction between CSR-style sparse representations and quantization of active dimensions is unexplored. Tight, distribution-dependent recall@K guarantees (rather than distance error bounds) remain an open theoretical problem. The streaming codebook staleness problem—where PQ codebooks degrade as data distributions shift—requires online adaptation mechanisms. Finally, hardware-software co-design for emerging architectures (CXL 4.0 memory pools, DRAM-PIM) may fundamentally reshape the quantization-memory hierarchy tradeoffs analyzed in this work.

References

1. Gao J, Long C (2024) RaBitQ: Quantizing High-Dimensional Vectors with a Theoretical Error Bound for Approximate Nearest Neighbor Search. ACM Journals 2: 1-27.
2. Tingyang Chen, Cong Fu, Jiahua Wu, Haotian Wu, Hua Fan, et al. (2025) IceBerg: Reveal Hidden Pitfalls and Navigate Next Generation of Vector Similarity Search from Task-Centric Views. Available at: https://arxiv.org/abs/2512.12980.
3. Junjie Qi, Gergely Szilvasy, Michael Norris, Vishal Gandhi (2025) Accelerating GPU Indexes in Faiss with NVIDIA cuVS. Available at: https://engineering.fb.com/2025/05/08/data-infrastructure/accelerating-gpu-indexes-in-faiss-with-nvidia-cuvs/.
4. Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, Rohan Kadekodi (2019) DiskANN: Fast Accurate Billion-point Nearest Neighbor Search on a Single Node. Available at: https://papers.nips.cc/paper_files/paper/2019/hash/09853c7fb1d3f8ee67a61b6bf4a7f8e6-Abstract.html.
5. Yutong Gou, Jianyang Gao, Yuexuan Xu and Cheng Long (2025) SymphonyQG: Towards Symphonious Integration of Quantization and Graph for Approximate Nearest Neighbor Search. ACM Journals 3: 1-26.
6. Gao J, Long C (2025) Practical and Asymptotically Optimal Quantization of High-Dimensional Vectors in Euclidean Space for Approximate Nearest Neighbor Search. ACM Journals 3: 1-26.
7. Aditya Nagori, Ricardo Accorsi Casonatto, Ayush Gautam, Abhinav Manikantha Sai Cheruvu, Rishikesan Kamaleswaran (2025) Open-Source Agentic Hybrid RAG Framework for Scientific Literature Review. Available at: https://arxiv.org/abs/2508.05660.
8. Hiroyuki Ootomo, Akira Naruse, Corey Nolet, Ray Wang, Tamas Feher, et al. (2024) CAGRA: Highly Parallel Graph Construction and Approximate Nearest Neighbor Search for GPUs. Available at: https://arxiv.org/abs/2308.15136.
9. Wenqi Jiang, Suvinay Subramanian, Cat Graves, Gustavo Alonso, Amir Yazdanbakhsh, et al. (2025) RAGO: Systematic Performance Optimization for Retrieval-Augmented Generation Serving. ISCA '25: Proceedings of the 52nd Annual International Symposium on Computer Architecture pp: 974-989.
10. Vallaeys H, Muckley F, Verbeek J, Douze M (2025) QINCo2: Vector Compression and Search with Improved Implicit Neural Codebooks. Available at: https://arxiv.org/html/2501.03078v1.
11. Zandieh A, Majid Daliri, Majid Hadian, Vahab Mirrokni (2025) TurboQuant: Online Vector Quantization with Near-optimal Distortion Rate. Available at: https://arxiv.org/html/2504.19874v1.
12. Aditya Kusupati, Gantavya Bhatt, Aniket Rege, Matthew Wallingford, Aditya Sinha, et al. (2022) Matryoshka Representation Learning. Available at: https://proceedings.neurips.cc/paper_files/paper/2022/hash/c32319f4868da7613d78af9993100e42-Abstract-Conference.html.
13. Xianming Li, Zongxi Li, Jing Li, Haoran Xie, Qing Li (2024) 2D Matryoshka Sentence Embeddings. Available at: https://arxiv.org/abs/2402.14776.
14. Tiansheng Wen, Yifei Wang, Zequn Zeng, Zhong Peng, Yudi Su, et al. (2025) Beyond Matryoshka: Revisiting Sparse Coding for Adaptive Representation. Available at: https://arxiv.org/abs/2503.01776.
15. Philip Adams, Menghao Li, Shi Zhang, Li Tan, Qi Chen, et al. (2025) DistributedANN: Efficient Scaling of a Single DiskANN Graph Thousands of Computers. Available at: https://arxiv.org/abs/2509.06046.
16. Faizan Ahemad (2025) Quantization Aware Matryoshka Adaptation: Leveraging Matryoshka Learning, Quantization, and Bitwise Operations for Reduced Storage and Improved Retrieval Speed. CIKM '25: Proceedings of the 34th ACM International Conference on Information and Knowledge Management pp: 25-34.
17. Charikar M (2002) Similarity Estimation Techniques from Rounding Algorithms. STOC '02: Proceedings of the thiry-fourth annual ACM symposium on Theory of computing pp: 380-388.
18. Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, et al. (2020) Accelerating Large-Scale Inference with Anisotropic Vector Quantization. Available at: https://arxiv.org/abs/1908.10396.
19. Xinyang Yi, Constantine Caramanis, Eric Price (2015) Binary Embedding: Fundamental Limits and Fast Algorithm. Available at: https://arxiv.org/abs/1502.05746.
20. Malkov Y, Yashunin D (2020) Efficient and Robust Approximate Nearest Neighbor Using Hierarchical Navigable Small World Graphs. IEEE TPAMI 42: 824-836.
21. Fu C, Xiang C, Wang C, Cai D (2019) Fast Approximate Nearest Neighbor Search with the Navigating Spreading-out Graph. PVLDB 12: 461-474.

22. Ji Sun, Guoliang Li, James Pan, Jiang Wang, Yongqing Xie, et al. (2025) GaussDB-Vector: A Large-Scale Persistent Real-Time Vector Database System. Proceedings of the VLDB Endowment 18: 4951-4963.
23. pgvector Contributors (2024) pgvector: Open-Source Vector Similarity Search for Postgres. Available at: https://github.com/pashkinelfe/pgvector-1.
24. Martin Aumüller, Erik Bernhardsson, Alexander Faithfull (2020) ANN-Benchmarks: A Benchmarking Tool for Approximate Nearest Neighbor Algorithms. Available at: https://arxiv.org/abs/1807.05614.