

RSLM: Training-Free Vector Quantization for Approximate Nearest Neighbor Search

Rastislav Lenhardt
Google
Zurich, Switzerland
lenhardt@google.com

Teodora Dobos
Technical University of Munich
Munich, Germany
Google
Zurich, Switzerland
teodora.dobos@tum.de

Thomas Vecchiato
University of Copenhagen
Copenhagen, Denmark
Google
Zurich, Switzerland
thomas.vecchiato@di.ku.dk

Jiří Iša
Google
Zurich, Switzerland
jisa@google.com

Igor Ginzburg
Google
San Jose, CA, USA
igorginzburg@google.com

Abstract

By introducing Rslm (Rotated Scaled Lloyd-Max), a family of training-free vector quantization codecs compressing embeddings to 1–4 bits per dimension, we reduce memory cost and memory bandwidth of a typical large-scale Approximate Nearest Neighbor (ANN) search system, while reducing its complexity and keeping or improving recall across multiple benchmark datasets. State-of-the-art systems filter candidates using coarse partitions, approximately score them to narrow the set, and then rescore the best with higher precision representations (often ≥ 8 bits per dimension). Our relativized codecs can bring this down to 2–4 bits per dimension.

We use the properties of the ANN system to encode residual vectors instead of full vectors, both for the approximate scoring phase and the rescoring phase. Since Maximum Inner Product Search (MIPS) is very sensitive to vector norms, we correct the L_2 norms of quantized vectors. Our major innovation is that we correct the L_2 norm of the final reconstructed vector rather than just the residual. Our rescaling replaces more complicated schemes, such as Anisotropic loss. The residualization scheme gives us a more favorable quality vs size trade-off than generic quantization methods.

Our high-performance implementation leverages a block-wise cascaded Fast Walsh-Hadamard Transform (FWHT) with linear-like complexity, AVX SIMD-optimized codebooks, and a steganographic encoding of scaling factors for perfect cache-line alignment.

1 Introduction

High-dimensional vector embeddings serve as the foundational representation for capturing unstructured semantic information across various modern domain pipelines, including search [28, 33, 34], recommendation [31, 37], and Retrieval-Augmented Generation (RAG) [20, 38]. Approximate Nearest Neighbor (ANN) search systems are often used to locate relevant vectors within billion-scale databases with high throughput and low latency [9]. In modern high-throughput database systems, search index design is fundamentally dictated by tight hardware constraints: DRAM-to-CPU memory bandwidth limits and physical DRAM capacity [25].

High-scale database architectures rely on clustering-based [27] and tree-based partitioning indexes [7, 16]. These paradigms divide

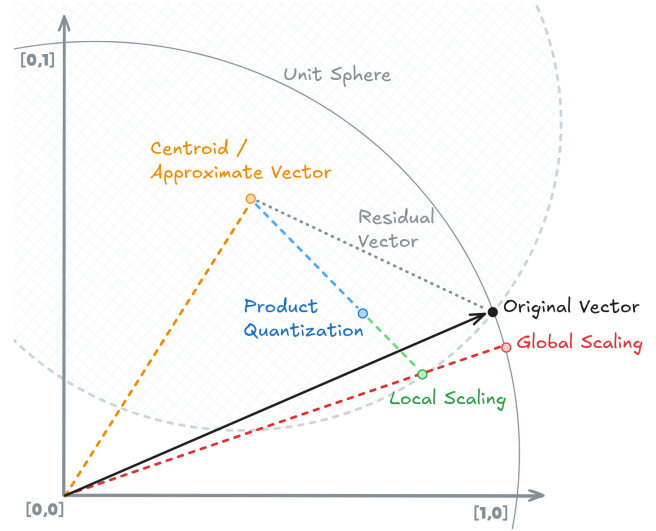


Figure 1: Geometric interpretation of residual quantization with both local and global scaling fixing the norm.

the vector space into localized Voronoi cells or spatial nodes, restricting online distance evaluations to a pruned subset of candidate vectors. For example, inverted file index (IVF) systems first identify spatially clustered candidate partitions and subsequently perform exhaustive Maximum Inner Product Search (MIPS) candidate re-ranking of candidates within the selected partitions [6, 43].

Within this second stage, vector quantization serves as an important technique to compress embeddings into compact representations [27]. This compression delivers a multi-fold system advantage: it reduces the overall DRAM memory footprint — enabling billion-scale in-memory indexes — while simultaneously cutting end-to-end query latency and boosting queries per second throughput by reducing DRAM-to-CPU transfer.

Traditional data-dependent quantization techniques, such as product quantization [27] (see Section 2), rely on offline training to construct dataset-specific codebooks. While data-oblivious quantizers eliminate this offline complexity and overhead, they suffer

from recall degradation in aggressive low-bitwidth regimes below 3 bits per dimension [18]. To address this, a recently emerging class of rotation-based data-oblivious quantizers, such as EDEN [42], TurboQuant [52] and BlockQuant [3], leverages randomized orthogonal transforms to homogenize coordinate variance prior to quantization. By mapping embedding dimensions to known analytical distributions and using a fixed codebook, these frameworks enable accurate, training-free quantization.

Our Motivation and Positioning. We improve on the existing generic data-oblivious quantizers by respecting ANN-specific constraints and leveraging ANN-specific opportunities. First, dense orthogonal projections (such as the Haar-distributed random orthogonal matrices employed in TurboQuant [52] and RaBitQ [19]) scale quadratically with vector dimensionality D , incurring large matrix storage overhead and query transformation latency. Second, while Fast Walsh-Hadamard Transforms (FWHT) mitigate this computational cost by scaling in $O(D \log D)$, global full-vector FWHTs are strictly restricted to power-of-two dimensions. When deployed on arbitrary dimensionalities (e.g., $D = 2049$), global FWHT requires zero-padding up to the next power of two, significantly increasing both computational work and memory bandwidth requirements. Most importantly though, existing general-purpose data-oblivious codecs are traditionally designed as standalone, full-vector encoders optimized for coordinate-wise reconstruction rather than an integrated part of a vector search retrieval system.

1.1 Contributions

To address these limitations, we introduce **Rslm (Rotated Scaled Lloyd-Max)**, a training-free vector quantization family of 1- to 4-bit codecs designed for high-performance clustering-based IVF and tree-based ANN rescoring. Rslm is optimized to operate across both the full vector space and the residual space of IVF/tree partitions and/or approximate vectors. By quantizing residual vectors, Rslm delivers near-lossless recall at extreme compression rates without any dataset-specific training. Our main contributions follow.

Linear rotation complexity. Instead of dense quadratic projections, or $O(D \log D)$ FWHT approach by EDEN, Rslm implements a cascaded 2-pass Block-FWHT. By restricting FWHT to fixed-size blocks of size B , the formal encoding complexity is $O(D \log B)$. Since the block size B is a small, hardware-optimized constant (we use $B \leq 128$), $\log B$ scales as a constant factor, yielding a linear complexity of $O(D)$.

Final-vector L_2 norm correction: MIPS ranking degrades if the norm is not corrected [15]. Previous approaches fixed the norm for the residual. Rslm takes this one step further, and explicitly corrects the L_2 norm of the final reconstructed vector rather than just the residual, matching or exceeding the recall of complex anisotropic loss formulations without requiring offline codebook training.

Zero-byte steganographic metadata: In the 4-bit Rslm4Lite variant, we embed the norm scale factor in the quantized vector. This reduces per-vector metadata overhead to exactly zero and enables perfect cacheline alignment for SIMD register lookups.

ANN simplification with training-free codecs: We evaluate Rslm codecs on full vector quantization and relative quantization in

respect to approximate vectors across five datasets, and conduct an end-to-end comparison against Faiss [17] and ScaNN [23] on *glove* and *openai* datasets, where we employ them to encode approximate vectors. We demonstrate that both in approximate scoring and rescoring phase our training-free codecs match or exceed the quality of the trained codecs.

Memory storage and bandwidth savings: When applied in rescoring phase, our relativized codecs achieve for all five evaluated datasets no recall@20@30 losses with 4-bit quantization (compared to default 8 bits used by ScaNN), and our 2-bit variants achieve 99%+ recall@20@40.

1.2 Limitations

Inner Product. We focus on ANN search to retrieve items with the **maximum inner product**. Because most modern embedding models output unit-normalized vectors, rankings generated by inner product, cosine similarity, and L_2 distance are mathematically identical, and computing dot-products is faster in practice. In our evaluation, four out of five datasets are unit-normalized, while the remaining *bigann* dataset exhibits minimal norm variance.

IVF Index. We focus on tree-based IVF indexes because they are highly optimized for environments with strict physical memory scaling limits and high-throughput requirements. As noted on *ann-benchmarks.com* [5], graph-based indexes like HNSW often achieve lower latency for a single query at a given recall target. However, IVF architectures maintain a crucial advantage in memory footprint and batched processing throughput. For our end-to-end comparison in Subsection 5.5, we evaluate ScaNN—one of the fastest throughput-optimized algorithms—and Faiss, one of the most widely deployed indexes in industry.

Our codecs can still be very useful also for graph-based HNSW architectures. Non-relative codecs could be used to compress node-vectors to reduce the memory required during graph traversal. And our relative codecs could be applied in a multistage approach similar to Milvus’s HNSW-PRQ [44], where the full precision vectors could be replaced by quantizing residuals against the coarse routing vectors, enabling highly accurate, training-free rescoring.

2 Background and Related Work

Scaling ANN search to billions of vectors is constrained by DRAM costs and memory bandwidth [25]. Similarity search engines typically navigate this tradeoff via multi-stage filtering: coarse spatial indices first prune the search space to candidate partitions, followed by quantized in-memory scanning to efficiently rescore vectors.

2.1 Space Partitioning and Candidate Selection

Modern similarity search engines decouple candidate pruning from final ranking using space-partitioning index structures [17]. While graph-based architectures like DiskANN [25] achieve exceptional recall-latency trade-offs, their reliance on in-memory graph adjacency lists imposes large physical RAM requirements. Similarly, Locality-Sensitive Hashing (LSH) frameworks like Fargo [54] are frequently bypassed in production-grade systems due to non-optimal empirical precision and mathematical disalignment with exact MIPS distance properties. To address these limitations, high-throughput engines adopt a two-stage retrieval pipeline generally built on

IVF structures [27] or tree-based space partitions [7, 16]. For instance, hybrid systems like SPANN [11] store coarse centroids in memory and keep postings on disk, whereas IVF-RaBitQ [39] pairs IVF with randomized hypercube projections. In the approximate scoring stage, the system first identifies localized Voronoi cells and uses low-bit vector quantization codecs (e.g., 1-bit or 2-bit) to rapidly compute estimated distances and prune candidates under strict memory bandwidth constraints. In the subsequent rescoring stage, the system fetches full-precision vectors or higher-fidelity quantized representations for a shortlist of top-k candidates to re-evaluate exact MIPS distances and finalize the top ranking.

2.2 Quantization

Vector quantization (VQ) maps high-dimensional vectors $x \in \mathbb{R}^D$ to a discrete set of codes from a codebook $C = \{c_1, \dots, c_K\}$ to reduce their memory footprint. The standard objective is to find a quantizer $q(x)$ that minimizes the reconstruction error, typically formulated as the mean squared error (MSE) or squared L_2 distortion:

$$\mathcal{L}_{\text{MSE}}(x) = \|x - q(x)\|^2 \quad (1)$$

Quantization frameworks can be broadly categorized as either *data-dependent* or *data-oblivious*, depending on how their codebooks and parameters are constructed.

In *data-dependent settings*, codebook learning is performed offline over a training dataset $\mathcal{X}$ by minimizing the empirical reconstruction error using iterative clustering algorithms such as k-means. Once learned, online encoding deterministically maps a vector x to its nearest codebook centroid $q(x) = \arg \min_{c \in C} \|x - c\|^2$. Because the codebook size of unconstrained VQ scales exponentially with the vector dimension D for a fixed quantization error, three main categories of structured codebook relaxations are widely adopted:

- *Scalar Quantization (SQ)*: This approach quantizes each coordinate independently using global or coordinate bounds learned from $\mathcal{X}$ (e.g., the Faiss codecs in Section 5.5.1). For instance, SQ8 performs scalar quantization to 2^8 levels, using a static per-dimension scale factor shared across the dataset.
- *Product Quantization (PQ)* [27]: This scheme decomposes the D -dimensional vector space into M orthogonal subspaces of dimension D/M , quantizing each sub-vector independently using a learned sub-codebook $C^{(m)}$. Optimized Product Quantization [21] improves PQ by using a learned orthogonal rotation to balance subspace variances, while Pyramid-PQ [45] adaptively merges adjacent subspaces to accelerate online distance computation.
- *Multi-Codebook and Residual Quantization (MCQ/RQ)*: To achieve higher reconstruction fidelity, multi-codebook methods such as Residual Quantization [12], Product Residual Quantization, and implicit neural codecs like Qinc2 [41] sequentially encode quantization residuals across cascading stages. However, these approaches require computationally intensive offline training phases and impose substantial decoding overhead during online query execution.

In contrast to data-dependent techniques, *data-oblivious methods* eliminate offline training by computing transformation parameters analytically or deriving them per vector at runtime. For instance,

Scaled Scalar Quantization (SSQ) [1] applies per-vector scaling. By storing a scale factor alongside the packed low-bit coordinates,¹ SSQ eliminates the need for offline training and guarantees robustness against out-of-distribution data shifts. We also note that the scale can be hidden inside the quantized vector. For instance, in Section 5 we compare the Rslm codecs against SSQ4Lite, which is a 4-bit scalar quantization codec that hides the 32-bit scale factor within the least significant bits of the quantized vector.²

Norm-Aware and MIPS-Oriented Quantization. While standard vector quantization minimizes MSE and implicitly weights reconstruction errors uniformly regardless of direction, MIPS depends on vector lengths. Dai et al. [15] introduced Norm-Explicit Quantization (NEQ), showing that quantization error in the vector norm has a greater negative impact on inner product ranking than error in vector direction. NEQ explicitly decouples and quantizes the scalar norm separately from the normalized direction vector. Furthermore, ScANN [23] penalizes parallel quantization errors via an anisotropic loss function. These methods demonstrate the necessity of norm preservation for MIPS, but rely on offline codebook training.

Orthogonal Preconditioning and Rotation-Based Codecs. While data-oblivious scalar quantizers perform well at high bit levels (e.g., 8-bit), coordinate variance skew and outlier dimensions cause recall degradation in aggressive low-bit regimes. To overcome this without resorting to offline k-means training, a distinct class of data-oblivious methods leverages randomized orthogonal preconditioning to homogenize coordinate variance prior to quantization. Within this class, EDEN [42] applies a randomized FWHT over a 1D Gaussian codebook, supporting flexible, non-integer number of bits per coordinate via coordinate sparsification. TurboQuant [52] constructs a 1D Lloyd-Max codebook calibrated to the exact Beta spherical coordinate marginal under a dense QR rotation, combined with a 1-bit QJL (Quantized Johnson-Lindenstrauss) [53] residual estimator. RaBitQ [19] and E-RaBitQ [18] project unit-normalized vectors onto rotated hypercube vertices for 1-bit and spherical multi-bit grids, though indexing complexity scales steeply with target bit-rate. BlockQuant [3] groups adjacent dimensions to perform continuous spherical k-means, relying on dense Haar rotations.

LLM-Centric Quantization. Beyond similarity search, vector quantization is used in post-training quantization and activation compression in Large Language Models (LLMs). SmoothQuant [49] introduces systematic 8-bit scaling, while QuIP [10] demonstrates that randomized orthogonal preconditioning suppresses channel outliers—a principle generalized by QuaRot [4] through computational invariance. Advanced low-bit frameworks include CRVQ [50], which applies codebook relaxation to critical weight channels, and LiftQuant [24], which enables quasi-continuous bit-widths. For memory-efficient attention serving, KIVI [32] enables 2-bit Key-Value (KV) cache compression, PolarQuant [48] applies recursive polar coordinate transformations, and HARP [51] leverages learnable butterfly orthogonal transforms. While LLM quantization techniques share preconditioning foundations with search codecs, they

¹In practice, this per-vector scale factor is derived from coordinate min/max bounds [1], vector norms [42, 52], or scalar multipliers optimized for MSE [19, 42].

²Specifically, for vectors with at least 64 dimensions, the 32-bit scale factor is embedded directly into the least significant bits of the quantized coordinates.

optimize for matrix multiplication throughput and perplexity rather than fast MIPS candidate rescoring.

Positioning of Rslm. As shown in Table 1, our proposed Rslm family occupies a unique operating point. We systematically categorize state-of-the-art training-free, rotation-based ANNS quantizers based on three key dimensions: their geometric preconditioning operator, the computational complexity of the projection step, and their per-vector metadata footprint. While classical oblivious quantizers operate exclusively on the full vector space, Rslm is designed to quantize both full vectors and relative residuals relative to coarse tree/IVF partitions. By operating on the residual space, Rslm compresses highly concentrated error vectors, drastically reducing quantization distortion.

Furthermore, while existing preconditioning methods require either quadratic $O(D^2)$ dense rotations or super-linear $O(D \log D)$ projections (as in EDEN [42]), Rslm implements a block-wise cascaded Block-FWHT that enforces a strict linear complexity of $O(D)$. Finally, while competing schemes rely on explicit per-vector scaling metadata to preserve distance estimation accuracy under inner product search, Rslm scales the reconstructed vector length to preserve the original L_2 norm, and through its Rslm4Lite variant, it hides this metadata steganographically, achieving zero storage overhead and enabling highly optimized prefetching.

Table 1: Taxonomy of state-of-the-art training-free, rotation-based ANNS quantizers, compared with Rslm and Rslm4Lite.

Method	Rotation	Rotation Complexity	Overhead-Free
EDEN [42]	Randomized Hadamard	$O(D \log D)^\dagger$	✗
RaBitQ [19]	Dense Orthogonal Rotation	$O(D^2)$	✗
E-RaBitQ [18]	Dense Orthogonal Rotation	$O(D^2)$	✗
TurboQuant [52]	Dense Orthogonal Rotation	$O(D^2)$	✗
BlockQuant [3]	Dense Orthogonal Rotation	$O(D^2)$	✗
Rslm (Ours)	Randomized Block-Hadamard	$O(D)^\dagger$	✗
Rslm4Lite (Ours)	Randomized Block-Hadamard	$O(D)^\dagger$	✓

[†] Hadamard-based transforms are computed via FWHT.

3 Rslm codec family

In this section, we describe the Rslm codec family, which provides vector quantization across 4, 3, 2, and 1-bit rates. Each codec in the Rslm family implements three main steps. First, it applies efficient pre-quantization transformations such that all coordinates have uniform variance. Second, it maps the rotated coordinates onto pre-computed optimal Lloyd-Max codebooks calibrated for the standard normal distribution. Third, it appends a 2-byte per-vector scale to restore vector length and prevent under-scoring in MIPS.

3.1 Pre-Quantization Transformations

Direct quantization of raw embedding vectors can lead to suboptimal codebook capacity utilization due to uneven coordinate variance and outliers. To mitigate this, we follow existing quantization

techniques [30, 42, 52] and apply orthogonal transformations to the original embeddings. These transformations distribute energy uniformly across all coordinates, shaping the transformed data toward a standard normal distribution while guaranteeing that inner product distances are preserved.

We implement a fast transformation as a cascaded 2-pass rotation combining three main components: the Fast Walsh-Hadamard Transform (FWHT), static sign-flipping, and block-wise coordinate permutations. These components are detailed below, followed by the complete execution pipeline in Section 3.1.3.

3.1.1 Fast Walsh Hadamard Transform. While applying a full random orthogonal transformation (e.g., via a Haar-distributed random matrix) achieves the desired properties of data uniformization, it is computationally prohibitive. For an embedding vector $x \in \mathbb{R}^D$, general matrix multiplication requires $O(D^2)$ operations. To overcome this computational bottleneck, we employ the Fast Walsh-Hadamard Transform. The unnormalized Walsh-Hadamard matrix $\hat{H}_n \in \{-1, +1\}^{n \times n}$ of dimension $n = 2^k$ is recursively defined as:

$$\hat{H}_{2^k} = \begin{bmatrix} \hat{H}_{2^{k-1}} & \hat{H}_{2^{k-1}} \\ \hat{H}_{2^{k-1}} & -\hat{H}_{2^{k-1}} \end{bmatrix}, \quad (2)$$

$$\hat{H}_1 = [1]$$

The orthonormal transform is given by $H_n = \frac{1}{\sqrt{n}} \hat{H}_n$. The FWHT algorithm exploits this recursive structure to apply the transformation in-place in $O(n \log n)$ time without requiring auxiliary memory. Because the entries of $\hat{H}_n$ are restricted to ± 1 , the transformation reduces to a sequence of additions and subtractions, with a single scaling factor of $1/\sqrt{n}$ applied at the end. This avoids expensive multiplications.

To optimize for hardware efficiency and memory bandwidth, we partition the vectors into blocks of constant size $B = 128$ and apply the FWHT to each block independently. This block-wise application bounds the overall computational complexity to $O(D \log B)$ —which is linear in the embedding dimension D . Furthermore, to maximize throughput, we implement the block-wise FWHT using SIMD vector instructions (e.g., AVX2). We optimize memory bandwidth by performing the initial stages of the transform entirely in-register before resolving the remaining stages across registers, maximizing instruction-level parallelism.

To handle arbitrary vector dimensions, we adjust the block-wise application in two ways. First, if the dimension is strictly smaller than 128, we dynamically shrink the block size to the largest power of 2 that fits. Second, if the dimension is not a multiple of the active block size, we use an overlapping approach: we process the vector in standard blocks, and handle the leftover elements by running a final block of the same size that overlaps with the preceding one. Because these block operations are non-commutative, they must be applied in reverse order during vector decompression to achieve exact invertibility.

3.1.2 Sign Flips and Dimension Permutation. The deterministic nature of the Hadamard transform means that it may not sufficiently uniformize certain input distributions (e.g., vectors aligned with the transform axes). To guarantee robust uniformization, we introduce pseudo-randomness using fixed, pre-computed sign-flip and permutation tables. Using static tables ensures that the transformations

remain strictly orthogonal while requiring zero metadata overhead. Specifically, the pipeline utilizes:

- **Sign Flips:** A static table of 256 pseudo-random signs (± 1) S_1 is tiled across the vector coordinates before the transform.
- **Dimension Permutation:** A static 128-element permutation P shuffles coordinates within each block. For blocks smaller than 128, a dynamic bit-reversal permutation is used instead.

For vectors exceeding 128 dimensions ($D > 128$), coordinates are interleaved across blocks using a strided transpose to break local block boundaries. For an arbitrary dimension D with active block size $B = 128$, for $M = \lfloor D/B \rfloor$ complete blocks, the prefix of length $M \cdot B$ is permuted and transposed by mapping $x'[k \cdot M + b] = x[b \cdot B + P(k)]$ for $0 \leq b < M$ and $0 \leq k < B$, while any leftover tail coordinates ($i \geq M \cdot B$) are left unchanged ($x'[i] = x[i]$). These untransposed tail coordinates are subsequently rotated and mixed into the global embedding space by the overlapping trailing FWHT block covering indices $[D - B, D]$ in the second pass. This ensures strict orthogonality, exact invertibility, and comprehensive dimension uniformization for arbitrary dimensions without requiring zero-padding or truncation.

3.1.3 Cascaded Rotation Pipeline. With the individual components defined in Sections 3.1.1 and 3.1.2, the pre-quantization transformation is executed. For the 4-bit codecs (Rslm4 and Rslm4Lite), we apply a single-pass optimization if the vector dimension is small ($D \leq 256$). In this case, we only apply the first-pass sign flips and the overlapping FWHT, bypassing the permutation, transpose, and second pass entirely to maximize query throughput. For larger dimensions ($D > 256$, or always for Rslm1/2/3), the complete transformation is executed in a two-pass cascade. For an input vector $x \in \mathbb{R}^D$, the operations are performed in the following order:

- (1) **First-Pass Sign Flips:** Multiply coordinates by the static sign-flip sequence S_1 .
- (2) **First-Pass FWHT:** Apply the block-wise FWHT.
- (3) **Dimension Permutation and Interleaving:** Permute coordinates within each block, and interleave them across blocks using the strided transpose operation.
- (4) **Second-Pass Sign Flips:** Apply the second static sign-flip sequence S_2 (which is S_1 offset by 127 elements).
- (5) **Second-Pass FWHT:** Apply the final block-wise FWHT.

3.2 Quantization Using Lloyd-Max Codebook

Following pre-quantization transformations, coordinate values approximate a zero-mean Gaussian distribution $N(0, \sigma^2)$ [42]. To map these coordinates onto our fixed Lloyd-Max codebooks – which are calibrated for the standard normal distribution $N(0, 1)$ – we scale the transformed coordinates prior to centroid assignment.

While the theoretical average coordinate variance is given by $\sigma^2 = \frac{1}{D} \|x\|^2$, individual rotated vectors might exhibit heavy-tailed coordinate peaks. Scaling strictly by sample variance can cause extreme coordinate outliers to saturate at the codebook boundaries, degrading quantization accuracy. To maintain robust dynamic range control and prevent outlier clipping, we apply Extreme Value Theory (EVT) [14] to compute a vector-adaptive initial scale factor

s_{initial} . Under the assumption that rotated coordinates behave as i.i.d. Gaussians, the expected maximum absolute coordinate value of a D -dimensional standard Gaussian vector is a known constant $E_{\text{max}}(D)$. We set s_{initial} dynamically based on the observed maximum coordinate magnitude of the vector $|z_i|$:

$$s_{\text{initial}} = \frac{\max_i |z_i|}{E_{\text{max}}(D)} \quad (3)$$

Because $1/E_{\text{max}}(D)$ is pre-computed, determining s_{initial} requires only a single pass to locate $\max_i |z_i|$ followed by a scalar multiplication. Normalizing coordinates by $1/s_{\text{initial}}$ dynamically anchors the codebook range to the vector’s actual peak coordinate, mitigating tail-clipping errors.

After normalizing the rotated vector by $1/s_{\text{initial}}$, we map its coordinates to the centroids. Depending on the target bit rate b , we adapt the codebook dimensionality and centroid count K . Specifically, for $b = 1$ we use a 4D codebook with $K = 16$, for $b = 2$ a 2D codebook with $K = 16$, and for $b = 3$ and $b = 4$ we utilize 1D codebooks with $K = 8$ and $K = 16$ centroids, respectively. When the embedding dimensionality D is not an exact multiple of the sub-vector size (4 for $b = 1$ or 2 for $b = 2$), the vector is zero-padded to the nearest multiple prior to centroid assignment; during query scoring and norm correction, only the valid D coordinates are accumulated by zeroing out trailing query terms, preserving exact inner-product evaluation without requiring separate 1D fallback codebooks.

As detailed in Section 5.4, restricting codebooks to at most 16 centroids is critical for hardware SIMD acceleration. To obtain the fixed Lloyd Max codebooks, we performed k-means clustering on a large set of synthetic samples drawn from a standard normal distribution of the corresponding dimensionality using K clusters. The resulting cluster centroids define the optimal quantization levels that minimize the mean squared error under the Gaussian assumption. Importantly, the codebooks that we use are data-independent.

3.3 Norm Correction

The Rslm codecs append a 2-byte per-vector scale factor to the quantized vector (except for Rslm4Lite, where the scale factor is embedded into the first dimensions, see Section 3.4) in order to restore vector length and improve MIPS ranking [15, 23]. Concretely, for an embedding vector x and its first-stage quantization $\hat{x}$, the reconstructed vector representation is $\tilde{x} = s \cdot \hat{x}$. The scale factor $s = \|x\|_2 / \|\hat{x}\|_2$ guarantees that the L_2 norm of the reconstructed vector matches that of the original vector.

This exact norm restoration contrasts with standard scaled scalar quantization, which inflates the expected norm, and product quantization, which deflates it (and while doing so they introduce variance in the norm across encoded vectors):

- SSQ models rounding error as independent uniform noise bounded by the quantization step size Δ . This uniform error over $[-\frac{\Delta}{2}, \frac{\Delta}{2}]$ has a variance of $\sigma^2 = \frac{\Delta^2}{12}$, causing the expected squared magnitude of the quantized vector to inflate according to $\mathbb{E}[\|\hat{x}\|_2^2] \approx \|x\|_2^2 + D \frac{\Delta^2}{12}$ [47].
- PQ centroids represent the conditional means of their corresponding Voronoi cells. Under this centroid property of optimal vector quantization, the expected quantized norm is deflated: $\mathbb{E}[\|q(x)\|_2^2] = \mathbb{E}[\|x\|_2^2] - \text{MSE}$ [22, 27].

This theoretical discrepancy is empirically pronounced in high-dimensional spaces. For instance, on the 1536-dimensional openai dataset, where original vectors are unit-normalized, the mean reconstructed norm of SSQ4Lite inflates to 1.82 ($\sigma = 0.03$), whereas 4-bit per dimension PQ deflates to 0.72 ($\sigma = 0.01$). Rslm eliminates this distortion entirely by enforcing exact norm alignment.

3.3.1 Scale Encoding. For the scale encoding, Float32 (S1E8M23) offers good range (useful for un-normalized embeddings) and good recall, but 4 bytes is expensive, especially for use cases with lower dimensionality and bits per dimension. The standard 16-bit floats both have drawbacks:

- BFloat16 (S1E8M7) offers range similar to Float32, but loses recall.
- Float16 (S1E5M10) achieves similar recall to Float32, but limits the range.

Since scales are non-negative, the sign bit is redundant. A parameter sweep showed UE7M9 offering a good middle ground. We implement UE7M9 without subnormals by converting to Float32 with a few simple ALU instructions.

3.4 Codecs Description

All Rslm variants share a unified compression pipeline: (1) apply a 2-pass cascaded orthogonal transformation as described in Section 3.1.3, (2) estimate the initial scale factor and quantize the normalized vector (Section 3.2), (3) compute the refined scale factor to preserve the L_2 norm (Section 3.3), and (4) pack the codes and scale factor.

Table 2 summarizes the technical specifications of each variant.

Table 2: Rslm Codec Specifications.

Codec	Bits/Dim	Type	K	Scale
Rslm1	1	4D	16	Suffix (2B)
Rslm2	2	2D	16	Suffix (2B)
Rslm3	3	1D	8	Suffix (2B)
Rslm4	4	1D	16	Suffix (2B)
Rslm4Lite	4	1D	16/8	Embedded

Rslm4Lite is designed for embedding dimensions $D \geq 64$ to eliminate the 2-byte storage overhead of the scale factor. For the first 16 dimensions, it uses $K = 8$ (3-bit) centroids. This frees up the most significant bit in each of the first 16 encoded dimensions. A 16-bit float (UE7M9) scale factor is then steganographically embedded into these 16 unused bits, yielding a packed size of exactly $\lceil D/2 \rceil$ bytes. In practice this means that, e.g., for 256-dimensional embeddings, memory-bandwidth bound systems need only two cachelines instead of three to prefetch the vector. This design was inspired by our optimization run using AlphaEvolve [35].

3.5 Relative Quantization

The Rslm codecs can be used either for full vector quantization as described in the previous sections, or for relative quantization (see Figure 1). In the latter setting, an Rslm codec is used to quantize the residual vector $r = x - a$, where x is the original vector and a is

the vector approximation from the initial quantization stage.³ The Rslm family supports two scaling options for relative quantization:

3.5.1 Local Scaling. In the default non-global mode (denoted as RelApprox in Table 6), the residual r is quantized on its own to $\tilde{r}$, preserving its own norm via a local scale factor (see Section 3.3).

To compute the dot-product for ranking, we can reuse the already computed dot-product of the query q with the approximate vector a because

$$\langle q, (a + \tilde{r}) \rangle = \langle q, a \rangle + \langle q, \tilde{r} \rangle. \quad (4)$$

This mode stores one 2-byte (UE7M9) float scale per vector to reconstruct the residual, except for Rslm4Lite, which has zero storage overhead.

3.5.2 Global Scaling. To correct for accumulated quantization error across all stages, the global mode (denoted as RelApproxGlobal in Tables 6 and 7 or Rslm_Global in Section 5.5) scales the combined approximation to restore the original vector length.

We compute a global scale factor $s = \|x\|_2 / \|a + \tilde{r}\|_2$ during encoding and append it as a 2-byte float to the packed data. Thus, two 2-byte floats are stored per vector (see Section 3.3). Since the global scale is close to 1.0, we do not need as much range as the local scale, so we use UE4M12, which devotes more bits to the mantissa, improving recall. We can achieve zero metadata overhead by applying again the trick from Rslm4Lite also to RelApproxGlobal-Rslm4Lite variant and hiding both scalars in the first 32 dimensions.

During scoring, the final dot product is reconstructed by scaling the combined scores, i.e. equal to $s(\langle q, a \rangle + \langle q, \tilde{r} \rangle)$.

4 Evaluation

4.1 Quality Metrics Used

Traditional vector quantization literature evaluates performance via MSE [23]. However, in database retrieval, user satisfaction and downstream ranking models depend on ranking accuracy rather than coordinate fidelity. Throughout this paper, we optimize and evaluate against ranking-centric metrics like Recall@X@Y (the fraction of top X ground truth exhaustive search results retrieved at top Y positions).

We evaluate four metrics: **Recall@20@30** (our default), **Recall@20@40** (more lenient, useful for downstream post-processing), **Recall@20@20** (traditional strict recall), and **1%-Tolerant Recall@20** [29].

Based on findings in the recent industrial benchmarks [29], we use Recall@20@30 as our default. As Table 3 illustrates, Recall@20@30 provides a necessary buffer to absorb benign mathematical noise—such as equidistant noise shuffling, tiny quantization artifacts, and borderline order swaps—that strict metrics like traditional Recall@20@20 falsely penalize. Instead, it strictly captures genuine quality failures, such as missing high-relevance matches. While 1%-Tolerant Recall is conceptually aligned with these goals, Recall@20@30 is significantly easier to measure reliably in distributed database engines, requiring only candidate document IDs to be returned rather than exact uncompressed float scores.

³We discuss how the approximate vector a can be computed in Section 5.2.

Table 3: Quality Attribution Across Recall Metrics.

Retrieval Scenario	Valid Loss?	R@20 @20	R@20 @30	Tolerant R@20
Missing top GT match	Yes	Loss	Loss	Loss
Swapping adjacent items	No	Loss	Ignored	Ignored
Tiny score artifacts	No	Loss	Ignored	Ignored
Dynamic index shifts	No	Loss	Ignored	Ignored
Equidistant noise shuffling	No	Loss	Mitigated	Ignored

Table 4: Overview of Evaluated Datasets

Dataset	Dim.	Size	Description & Characteristics
glove [36]	100	1 M	Word embeddings
bigann [26]	128	100 M	SIFT image descriptors; strictly non-negative coordinates with strong inter-dimension correlation; not unit normalized
openai [13]	1536	2 M	OpenAI text embeddings
wiki_full	3072	2 M	Gemini embeddings of Wikipedia
wiki_pca	384	2 M	PCA + random rotation applied to wiki_full

4.2 Datasets and Brute-Force Evaluation

We evaluate codecs across five standardized datasets covering diverse dimensionalities, dataset sizes, and coordinate distributions, see Table 4. To decouple quantization loss from index partitioning errors, our quality evaluations focus on brute-force approach. We use 500 queries per dataset and for each query, we isolate the ground-truth top-500 nearest neighbors and evaluate ranking metrics directly on this candidate pool.

As we are interested in codecs with limited quantization error, it is improbable for a candidate outside the top-500 ground truth to reach the top 20 or 30 with this limited error. We verified this also in practice for openai, bigann and glove datasets by comparing with quantizing all documents for given queries (see our reproduction Colab notebook for glove). The only statistically significant impact was recall difference on non-relativized Rslm1 for all and Rslm2 only for bigann, i.e., when the recall was already low. Using the best-scoring subset for the evaluation allowed us to both iterate quickly and be closer to the real world settings, where modern ANN systems usually score only a small fraction of all the results.

4.3 Evaluated codecs

We compare our Rslm codecs with SQ8 (default rescoring codec used in ScaNN), SSQ4Lite (simple 4-bit scalar quantization variant which doesn’t need a codebook), and 4-bit Turboquant representing as a reference point EDEN, TurboQuant and BlockQuant (several existing algorithms on which ideas we build). We specifically picked TurboQuant as we had access to optimized C++ version of its code.

5 Results

5.1 Quality Results for Full Vector Quantization

We present the key observations based on Table 5, which shows Recall@20@30 on full vector (non-relative) quantization with 95% confidence intervals.

- (1) **Impact of Rotation:** The unrotated codecs Slm4⁴ and SSQ4Lite suffer severe recall drops on wiki_full (79.1% and 89.8%), bigann (60.9% and 60.2%), and openai (57.5% and 60.6%) due to non-zero coordinate means and variance skew. In contrast, Rslm4 and TurboQuant achieve >99% recall on wiki_full and openai and >92% recall on bigann by homogenizing variance prior to quantization. wiki_pca had full random rotation applied as the last step of PCA, so Slm4 and SSQ4Lite perform there well as expected.
- (2) **Dimensionality Scaling:** Higher embedding dimensionality naturally protects against quantization noise. On wiki_full (D=3072), even 3-bit and 2-bit codecs achieve 99% and 94% recall, respectively. These results are supported by the theoretical properties of the behavior of random projections and quantization noise [8, 46]. The true inner product signal power grows quadratically, while the variance of uncorrelated, zero-mean quantization noise grows linearly. Consequently, the Signal-to-Noise Ratio of the estimated inner product scales as $D \cdot 2^{2b}$ [23]. Therefore, on average, quadrupling the dimensionality allows a reduction of one bit per dimension without degrading the statistical reliability of the inner product.
- (3) **Lite vs. Standard 4-bit:** Embedding the scalar avoids per-vector storage overhead for cacheline alignment (see Subsection 5.4). Although this trade-off is more visible on lower-dimensional datasets where the 16 hidden dimensions represent a larger fraction of the vector, the impact remains negligible in high dimensions, making Rslm4Lite optimal for high-dimensional workloads.

5.2 Quality Results for Residual (Relative) Vector Quantization

For relative quantization (see Figure 1), we precompute approximate vectors using a multi-stage residual quantization pipeline. First, we partition the datasets such that there are approximately 100 vectors per partition⁵, and assign each vector to its nearest coarse partition center. The resulting first-stage residual is then quantized using ScaNN’s Asymmetric Hashing method (PQ) [23] with 1 bit per dimension (see Section 5.5 for ScaNN codecs description and note that we could also apply training-free codec for approximate scoring as we do in Section 5.5).

We reconstruct the final approximate document vector by summing the assigned partition center and the decoded residual. This reconstructed vector serves as the approximation for RelApprox variants, which subsequently quantize the remaining secondary residual. The following observations are based on Tables 6 and 7, which show the recall for quantized residual vectors used in candidate rescoring:

⁴Slm4 (Scaled Lloyd-Max) is a codec similar to Rslm4, but without the rotation.

⁵For the large bigann dataset, this means using 2-level hierarchical partitioning tree.

Table 5: Recall@20@30 Evaluation Results for Non-Relative Codecs.

Codec	Bits/Dim	wiki_full ($D = 3072$)	wiki_pca ($D = 384$)	bigann ($D = 128$)	glove ($D = 100$)	openai ($D = 1536$)
SQ8	8	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%	98.8% $+0.4\%$ -0.5%	100.0% $+0.0\%$ -0.0%	99.6% $+0.1\%$ -0.2%
TurboQuant	4+s	100.0% $+0.0\%$ -0.0%	99.3% $+0.2\%$ -0.2%	92.2% $+0.6\%$ -0.7%	96.9% $+0.4\%$ -0.4%	99.3% $+0.1\%$ -0.2%
Rslm4	4+s	99.9% $+0.1\%$ -0.1%	99.2% $+0.2\%$ -0.2%	92.3% $+0.6\%$ -0.7%	97.3% $+0.4\%$ -0.4%	98.9% $+0.2\%$ -0.2%
Rslm4Lite	4	100.0% $+0.0\%$ -0.0%	99.2% $+0.2\%$ -0.2%	89.5% $+0.8\%$ -0.9%	95.5% $+0.5\%$ -0.5%	99.0% $+0.2\%$ -0.2%
Slm4	4+s	79.1% $+1.0\%$ -0.9%	98.6% $+0.2\%$ -0.2%	60.9% $+1.6\%$ -1.7%	93.6% $+0.6\%$ -0.6%	57.5% $+1.3\%$ -1.4%
SSQ4Lite	4	89.8% $+0.7\%$ -0.8%	98.1% $+0.3\%$ -0.3%	60.2% $+1.4\%$ -1.6%	93.3% $+0.5\%$ -0.6%	60.6% $+1.3\%$ -1.2%
Rslm3	3+s	99.0% $+0.2\%$ -0.2%	94.6% $+0.4\%$ -0.5%	75.3% $+1.0\%$ -1.1%	87.2% $+0.8\%$ -0.9%	94.7% $+0.5\%$ -0.5%
Rslm2	2+s	94.2% $+0.5\%$ -0.5%	82.2% $+0.8\%$ -0.9%	49.7% $+1.2\%$ -1.2%	67.9% $+1.4\%$ -1.4%	82.2% $+0.9\%$ -1.0%
Rslm1	1+s	79.2% $+0.9\%$ -0.9%	56.2% $+1.1\%$ -1.2%	25.4% $+0.9\%$ -0.9%	39.2% $+1.7\%$ -1.6%	61.0% $+1.2\%$ -1.4%

Note: ‘+s’ indicates that an extra scaling factor is stored per vector.

Table 6: Recall@20@30 Evaluation Results for Relative Codecs.

Codec	Bits/Dim	wiki_full ($D = 3072$)	wiki_pca ($D = 384$)	bigann ($D = 128$)	glove ($D = 100$)	openai ($D = 1536$)
RelApprox-SQ8	8	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%	99.9% $+0.1\%$ -0.1%	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%
RelApproxGlobal-Rslm4	4+s	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%
RelApproxGlobal-Rslm4Lite	4	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%	99.9% $+0.0\%$ -0.1%	99.8% $+0.1\%$ -0.1%	100.0% $+0.0\%$ -0.0%
RelApprox-Rslm4	4+s	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%	99.0% $+0.2\%$ -0.3%	99.8% $+0.1\%$ -0.1%	100.0% $+0.0\%$ -0.0%
RelApprox-TurboQuant	4+s	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%	98.8% $+0.3\%$ -0.3%	99.8% $+0.1\%$ -0.1%	100.0% $+0.0\%$ -0.0%
RelApprox-Slm4	4+s	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.1%	97.1% $+0.4\%$ -0.5%	99.5% $+0.1\%$ -0.1%	98.2% $+0.3\%$ -0.2%
RelApprox-Rslm4Lite	4	100.0% $+0.0\%$ -0.0%	100.0% $+0.0\%$ -0.0%	98.1% $+0.4\%$ -0.4%	99.6% $+0.1\%$ -0.1%	100.0% $+0.0\%$ -0.0%
RelApprox-SSQ4Lite	4	100.0% $+0.1\%$ -0.0%	99.9% $+0.1\%$ -0.1%	94.8% $+0.5\%$ -0.6%	99.2% $+0.2\%$ -0.2%	99.9% $+0.1\%$ -0.1%
RelApproxGlobal-Rslm3	3+s	100.0% $+0.0\%$ -0.0%	99.6% $+0.1\%$ -0.1%	99.6% $+0.1\%$ -0.1%	98.9% $+0.2\%$ -0.2%	100.0% $+0.0\%$ -0.0%
RelApprox-Rslm3	3+s	100.0% $+0.0\%$ -0.0%	99.4% $+0.2\%$ -0.1%	94.0% $+0.6\%$ -0.8%	98.2% $+0.3\%$ -0.3%	99.7% $+0.1\%$ -0.1%
RelApproxGlobal-Rslm2	2+s	100.0% $+0.0\%$ -0.0%	97.2% $+0.3\%$ -0.3%	97.2% $+0.3\%$ -0.4%	94.2% $+0.6\%$ -0.6%	99.7% $+0.1\%$ -0.1%
RelApprox-Rslm2	2+s	100.0% $+0.0\%$ -0.0%	96.5% $+0.4\%$ -0.4%	80.9% $+1.0\%$ -1.1%	91.7% $+0.6\%$ -0.7%	97.4% $+0.3\%$ -0.3%
RelApproxGlobal-Rslm1	1+s	99.5% $+0.2\%$ -0.2%	86.9% $+0.8\%$ -0.7%	87.7% $+0.8\%$ -0.8%	82.1% $+1.1\%$ -1.0%	97.3% $+0.3\%$ -0.4%
RelApprox-Rslm1	1+s	99.2% $+0.2\%$ -0.2%	85.1% $+0.8\%$ -0.8%	57.6% $+1.1\%$ -1.1%	76.0% $+1.2\%$ -1.2%	88.6% $+0.7\%$ -0.7%

Note: ‘+s’ indicates that one or two extra scaling factors are stored per vector.

Table 7: Recall@20@40 Evaluation Results for Selected Codecs.

Codec	Bits/Dim	wiki_full ($D = 3072$)	wiki_pca ($D = 384$)	bigann ($D = 128$)	glove ($D = 100$)	openai ($D = 1536$)
RelApproxGlobal-Rslm2	2+s	100.0% $+0.0\%$ -0.0%	99.4% $+0.2\%$ -0.2%	99.3% $+0.2\%$ -0.2%	97.7% $+0.3\%$ -0.4%	100.0% $+0.0\%$ -0.0%
RelApproxGlobal-Rslm1	1+s	100.0% $+0.0\%$ -0.0%	93.1% $+0.5\%$ -0.5%	93.5% $+0.5\%$ -0.6%	88.2% $+0.9\%$ -0.9%	99.4% $+0.2\%$ -0.2%

Note: ‘+s’ indicates that one or two extra scaling factors are stored per vector.

- (1) **Superiority to non-relativized encodings:** Across all datasets and bit rates, having the additional information

(approximate vector to which we quantize relatively) drastically reduces recall loss. Our relativized 4-bit Rslm encodings achieve 100% recall, and 3-bit Rslm encodings achieve

$\geq 98.9\%$ recall across all datasets, offering substantial efficiency gains for the current state-of-the-art ANN systems.

- (2) **Extreme bandwidth efficiency for RAG systems:** In production vector databases, where downstream ranking systems ingest a larger candidate pool, it makes more sense to also look at metrics like Recall@20@40. We observe that our relativized 2-bit Rslm encoding achieves 99.3%+ recall@20@40 across four out of five datasets (only for the lowest 100 dimensional glove it is 97.7%); For high-dimensional datasets, our relativized 1-bit Rslm encoding achieves 100% recall@20@40 on 3072-dim wiki_full and 99.4% recall@20@40 on 1536-dim openai. This enables 4x to 8x reduction in rescoring vectors DRAM footprint and bandwidth.
- (3) **Fixing the norm is crucial:** RelApproxGlobal significantly outperforms RelApprox encodings, experimentally confirming that fixing the norm of the full reconstructed vector (approximate + residual) is critical for MIPS ranking.

5.3 Other Recall Metrics

We provide additional recall metrics (defined in Section 4.1) in Table 8. They trend as expected, with the exception that the bigann dataset achieves high 1% Tolerant recall@20, even though recall@20@X metrics are worse. Looking closer, this is explained by bigann having clusters of documents with very similar vectors (magnified by discretized per-dimension values – integers between 0 and 255).

Table 8: Various recall metrics comparison across selected datasets and codecs.

Codec	Metric	openai	glove	bigann
SQ8	R@20@20	91.4%	98.0%	92.4%
	R@20@30	99.6%	100.0%	98.8%
	R@20@40	100.0%	100.0%	99.4%
	Tolerant	100.0%	100.0%	100.0%
SSQ4Lite	R@20@20	49.9%	81.0%	49.4%
	R@20@30	60.6%	93.3%	60.2%
	R@20@40	68.0%	97.3%	67.6%
	Tolerant	82.5%	87.4%	81.1%
Rslm4	R@20@20	90.5%	86.8%	79.6%
	R@20@30	98.9%	97.3%	92.3%
	R@20@40	99.9%	99.3%	96.6%
	Tolerant	100.0%	92.7%	99.6%
RelApproxGlobal-Rslm4	R@20@20	97.5%	94.9%	95.5%
	R@20@30	100.0%	100.0%	100.0%
	R@20@40	100.0%	100.0%	100.0%
	Tolerant	100.0%	99.3%	100.0%
RelApproxGlobal-Rslm2	R@20@20	92.9%	82.9%	86.4%
	R@20@30	99.7%	94.2%	97.2%
	R@20@40	100.0%	97.7%	99.3%
	Tolerant	100.0%	89.0%	99.8%

5.4 Performance Impact and Memory Savings

We measure dot-product throughput and codec operation throughput using a standardized microbenchmark suite on both AMD Milan (x86_64 AVX2) and Intel Cascade Lake (x86_64 AVX-512) server platforms to capture different capacity and utilization of L1/L2/L3 CPU caches by those platforms.

To achieve peak speed, vector quantization algorithms (like Rslm) perform table lookups directly inside CPU registers using byte-shuffle instructions (VPSHUFFB). However, because AVX2 hardware restricts these shuffles to 16-byte lanes, codebooks cannot exceed 16 elements without suffering severe performance penalties [2].

The main focus of our benchmarks is OneToMany Dot-product operation, i.e., computing the dot-product of one query vector with many document vectors. Figure 2 shows the throughput if the scored vectors are contiguous in memory (Plot 1) or whether they are randomly spread within the large pool of the data (Plot 2). The latter is modeling the real-world rescoring use-case better, while the former shows the headroom if DRAM bandwidth was not an issue. The plots clearly show:

- A smaller RAM footprint not only saves cost (as less RAM is needed), but it also leads, because of higher throughput, to much better end-to-end system performance.
- The cacheline-optimized Rslm4Lite significantly outperforms Rslm4 on the LargePool benchmark.
- Even though Rslm4Lite uses the fixed Lloyd-Max codebook, in contrast to SSQ4Lite not having any codebook, it can compete on the throughput thanks to highly optimized SIMD CPU instructions.

We also evaluated the throughput of Compress, Decompress, and Precompute-Query operations. Compress is applied on each vector once at index building time, making it less critical, but it can still be a substantial cost. For codecs utilizing rotations, it makes a big difference whether it is full random rotation, FWHT on the whole vector, or very efficient per-block approach we apply in Rslm codecs family. This rotation needs to be applied also at serving time, once per query, so that we can then directly compute dot-products in the rotated space instead of decompressing all the vectors on the fly. Being able to have the fast rotation, whatever the vector dimensionality is, was one of the reasons that, even though Rslm4 is similar to 4-bit TurboQuant, we created an Rslm4 version (see Table 1). In production, non-standard dimensionalities arise (e.g., $D = 2049$, where an extra dimension encodes domain-specific metadata). While performing comparably at $D = 256$, TurboQuant experiences significant slowdown during Compress and Precompute-Query at $D = 2049$ due to its reliance on full-dimensional rotations – on AMD Milan Turboquant can compress 3.5k items per second, while Rslm4 compresses 128k items per second.

5.5 End-to-End Comparison with Faiss and ScaNN

In this section, we perform an end-to-end comparison of the Rslm codecs with the quantization methods available in the Faiss⁶ and ScaNN⁷ libraries on the full glove and openai datasets over 500

⁶<https://github.com/facebookresearch/faiss>

⁷<https://github.com/google-research/google-research/tree/master/scann>

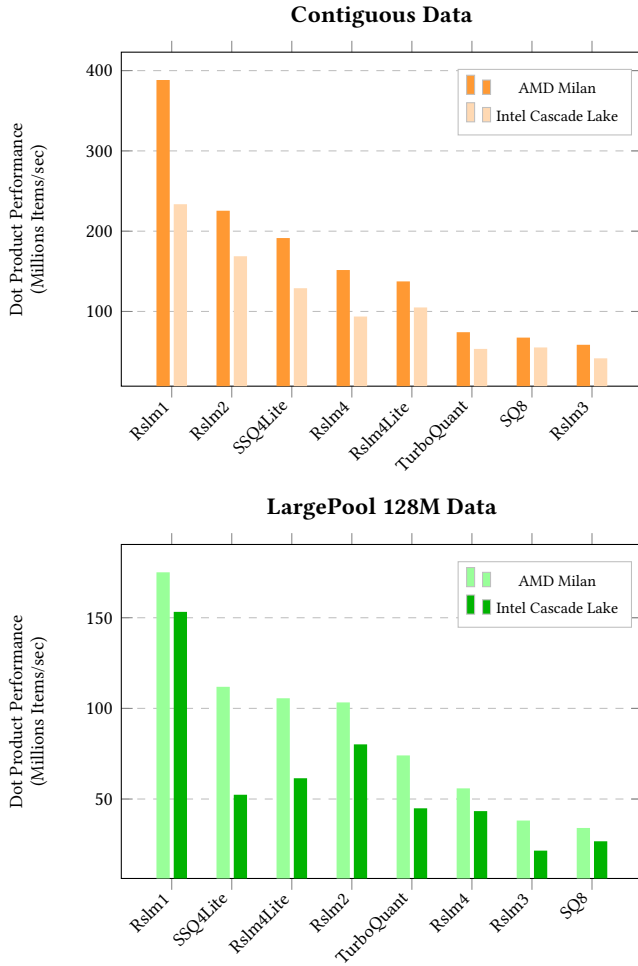


Figure 2: Dot product performance on random 256 dimensional embeddings on AMD Milan and Intel Cascade Lake.

queries. The goal of this comparison is to evaluate how these quantization methods perform in realistic vector search scenarios. In contrast to previous subsections, where we focused on rescoring phase (i.e., comparing encodings of vectors relative to approximate vectors), here we show the benefits also for approximate scoring (i.e., comparing encodings relative to partition centers).

5.5.1 Setup. The evaluation pipeline consists of three phases:

- (1) **Dataset Partitioning:** The datasets are clustered to define coarse search partitions.
- (2) **Residual Quantization:** We quantize the residual vectors (the difference between each vector and its assigned partition centroid) using the respective codecs.
- (3) **Query Routing and Scanning:** At query time, we identify the n_{probe} closest partitions (defining the search budget) based on the dot product to the centroids. We then scan the quantized residual vectors within these active partitions to retrieve the top- k nearest neighbors.

For the partitioning step, we configure both Faiss (using an IVF4096 Flat index) and ScaNN (using a single-level k-means tree) to partition the dataset into 4,096 partitions for glove and 8,192 for openai.⁸ To ensure a fair comparison, both partitioners are trained using identical parameters: 25 k-means iterations, a training sample size of 1,048,576 vectors for glove and 2,097,152 for openai (corresponding to 256 points per centroid on average), and random initialization. To evaluate each baseline under its optimal settings, Faiss and ScaNN codecs are run on their respective default native partitions (spherical k-means for Faiss, and standard Euclidean k-means for ScaNN), while our proposed codecs are evaluated directly on ScaNN’s partitions.

Faiss Codecs. We briefly describe the Faiss codecs which we use in our evaluation. For more details, please see [17].

- **Scalar Quantization (Faiss SQ):** Maps floating-point values to equal-sized bins within a dynamically determined range. We evaluate both the default *non-uniform* version, which maintains an independent range per dimension (e.g., Faiss SQ4), and the *uniform* version, which shares a single range across all dimensions (e.g., FaissSQ4 Uniform).
- **Product Quantization (Faiss PQ):** Decomposes the vector into M sub-vectors of dimension D/M and quantizes each subspace independently using a codebook trained via k-means. For instance, FaissPQ50x4 partitions the 100-dimensional GloVe vector into 50 2-dimensional sub-vectors, quantizing each with a 4-bit ($K = 16$ centroids) codebook to yield an overall rate of 2 bits/dimension.

ScaNN Codecs. From the ScaNN library, we evaluate Asymmetric Hashing (AH), which is ScaNN’s implementation of trained PQ. While the codebook itself is trained using standard k-means, the vectors can be encoded based on either MSE or anisotropic loss [23]. Unlike MSE, anisotropic loss prioritizes MIPS ranking by penalizing parallel error, i.e., the quantization error component along the vector’s direction that directly alters inner product values. A threshold $t \in [0, 1)$ specifies the expected similarity regime: a higher t assumes closer nearest neighbors and penalizes parallel error more aggressively, whereas a lower t converges toward MSE.

Faiss and ScaNN also provide the option to apply a random orthogonal projection matrix to rotate the vectors before quantization. This distributes coordinate variance uniformly, similar in goal to Rslm’s FWHT-based pipeline, but requires dense $O(D^2)$ matrix multiplication. We denote by *Faiss Rotated* and *ScaNN Rotated* the codecs for which such a rotation was applied prior to quantization.

Results. We analyze recall as a function of the search budget n_{probe} , testing 0.1%, 0.3%, 1%, 10%, and 100% of the total partitions (i.e., $n_{\text{probe}} \in \{4, 12, 41, 410, 4096\}$ for glove and $n_{\text{probe}} \in \{8, 25, 82, 819, 8192\}$ for openai). Figure 3 illustrates recall@20@30 obtained for different search budgets with 4- to 1-bit codecs.

ScaNN results obtained with anisotropic vector quantization (AVQ) using threshold t are marked with the suffix “Anisot”. In our experiments, we evaluated the thresholds $t \in \{0.1, 0.2, 0.3\}$ for glove, and included in the plots the value that led to the best recall. Note that using the same thresholds for openai, ScaNN AVQ led

⁸This corresponds to $4 \times \sqrt{N}$ rounded up to the next power of 2, where N is the size of the considered dataset.

to considerably worse results than standard ScaNN PQ at 2- and especially 1-bit levels (e.g., for $n_{\text{probe}} = 82$ and 1-bit level, ScaNN PQ Aniso0.2 has 18.7% recall@20@30, while ScaNN PQ has 61.66% recall@20@30). This is because in the anisotropic loss function, the parameter that is used to penalize the parallel error depends on both the threshold and the vector dimension.⁹ Thus, configuring t in ScaNN AVQ requires tuning and depends on the dataset. We evaluated the equivalent thresholds for `openai` that imply the same parallel multiplier as the thresholds used for `glove` and included in the plots the best one. In what follows, we discuss the `glove` results, noting that the `openai` results exhibit similar behavior.

In an end-to-end ANN search pipeline, retrieval quality is determined by both coarse partitioning errors (missing the correct cluster) and quantization errors (distance distortion during scoring). The relative impact of these two error sources depends on the codec’s bitrate. At higher bitrates (such as 4-bits), quantization error is negligible (see Table 5), meaning retrieval quality depends primarily on partitioning errors. In this regime, increasing the search budget yields substantial recall gains (e.g., from 54.1% at $n_{\text{probe}} = 4$ to 95.8% at $n_{\text{probe}} = 410$ for `Rslm4_Global`). Conversely, at lower bitrates, quantization noise becomes the primary bottleneck, capping the maximum achievable recall even when searching the entire database (54.0% recall for `Rslm1_Global` at $n_{\text{probe}} = 4096$).

Similar Recall Without Codebook Training. Unlike standard PQ in Faiss and ScaNN, which requires running k-means to train dataset-specific codebooks, Rslm utilizes fixed Lloyd-Max codebooks. This training-free approach is made possible by Rslm’s rotation pipeline. Our evaluation across all bit levels shows that Rslm codecs achieve comparable or superior recall to the trained Faiss and ScaNN configurations. By eliminating the k-means training phase, Rslm enables fast deployment on new datasets (where we do not have enough samples for training) and considerably reduces indexing complexity.

Low-Bitrate Superiority of Rslm. At highly compressed rates (1- and 2-bit, which are also the default settings in ScaNN), Rslm_Global codecs significantly outperform ScaNN and Faiss. For instance, for $n_{\text{probe}} = 4096$, Rslm1_Global reaches 54.0% recall, outperforming ScaNN’s AVQ (45.6%) and Faiss’s PQ (42.1%). At 2-bits, Rslm2_Global outperforms ScaNN’s best variant by 4.3% (Faiss by 8.7%).

The superior results of Rslm low-bit codecs can be attributed to fixing the norm of the vector for ranking. While ScaNN AVQ improves the results compared to simple ScaNN PQ, it requires tuning. Rslm achieves a similar effect by strictly enforcing norm preservation in the reconstructed vector. Although this requires storing an auxiliary scalar per vector, it avoids complex anisotropic optimization. For low-dimensional datasets like `glove`, this extra scalar increases the memory requirements by 15% for 1-bit codecs (Faiss PQ requires 13, while Rslm1 requires 15 bytes per vector), while for higher-dimensional datasets like `openai`, the increase is modest, only 1% (192 in Faiss PQ vs. 194 in Rslm1). Furthermore, globally scaling the combined centroid-plus-residual vector yields an additional increase in recall (this extra scalar means that two scalars are 31% increase in size for 1-bit codecs for `glove` while only 2% increase for high-dimensional `openai`). This allows us to achieve

similar recall while searching significantly fewer partitions (if this is the final scoring), or rescoring significantly fewer documents (if this is approximate scoring phase). We acknowledge that storing an extra scalar on low-bit codecs makes it slightly unfair for comparisons on lower dimensional datasets like `glove` but is negligible for higher-dimensional datasets like `openai` or `wiki_full`.

Codebook Training Differences. Finally, we note that while both ScaNN and Faiss implement PQ, they differ in their codebook training initialization. ScaNN utilizes k-means++ initialization, whereas Faiss relies on standard random initialization. Consequently, the recall results for these structurally equivalent PQ configurations are close but not identical. For instance, at $n_{\text{probe}} = 41$, 4-bit Faiss PQ slightly outperforms ScaNN PQ (81.4% recall vs. 80.1%). At 2 bits/dimension, the difference in recall is 1.3% (66.1% in Faiss, 64.8% in ScaNN), while at 1 bit/dimension, the results are very close (41.5% in ScaNN, 41.1% in Faiss). We also note that applying random orthogonal rotations improves SQ, but it generally does not improve PQ. This happens because random rotations spread variance evenly across dimensions, helping SQ’s rigid coordinate ranges avoid clipping high-variance coordinates, whereas PQ already handles multi-dimensional spaces by clustering sub-vectors directly.

6 Conclusions

We introduced Rslm, a family of training-free vector quantization codecs spanning 1 to 4 bits per dimension, designed to save memory resources and eliminate memory-bandwidth bottlenecks in large-scale ANN search. We optimized them for encoding the residual vectors and demonstrated that the current state-of-the-art systems could benefit from $2\times$ to $4\times$ improvements. One can take these as direct savings. Another option is to fund with them other optimizations like SOAR [40] which needs extra memory but leads to a much smaller number of partitions that need to be searched.

Our major contribution is not only to correct the norm of the residual vector, but of the final reconstructed vector. This is a simple and yet highly effective solution to preserve geometric integrity for MIPS, experimentally outperforming complex anisotropic loss formulations at low-bitrate compressions of residual vectors.

We achieved highly scalable, hardware-efficient processing, by focusing on details of the whole end-to-end pipeline: (1) We proposed block-wise cascaded FWHT which effectively approximates dense global rotations with linear-like complexity even for strange input dimensionalities. (2) We kept codebooks of size at most 16 to be able to utilize fast AVX SIMD instructions. (3) We introduced Rslm4Lite codec which hides a 16-bit scale factor to achieve perfect cache-line alignment providing a major throughput advantage for production vector databases.

And finally, our empirical results demonstrate that completely training-free codecs can match or exceed the semantic recall of highly optimized, data-dependent industry standards (Faiss PQ, ScaNN AH) without the burden of offline k-means training. Having training-free codecs both for the approximate scoring phase (as shown in Section 5.5) and the rescoring phase (as shown in Subsection 5.2) of ANN paves the way for simplified and lower-cost production vector database architectures.

⁹Specifically, the parameter is $\eta = \frac{t^2(D-1)}{1-t^2}$ and the anisotropic loss is $\mathcal{L} = \eta \|e_{\parallel}\|^2 + \|e_{\perp}\|^2$, where $e_{\parallel}$ and $e_{\perp}$ denote the parallel and perpendicular errors, respectively.

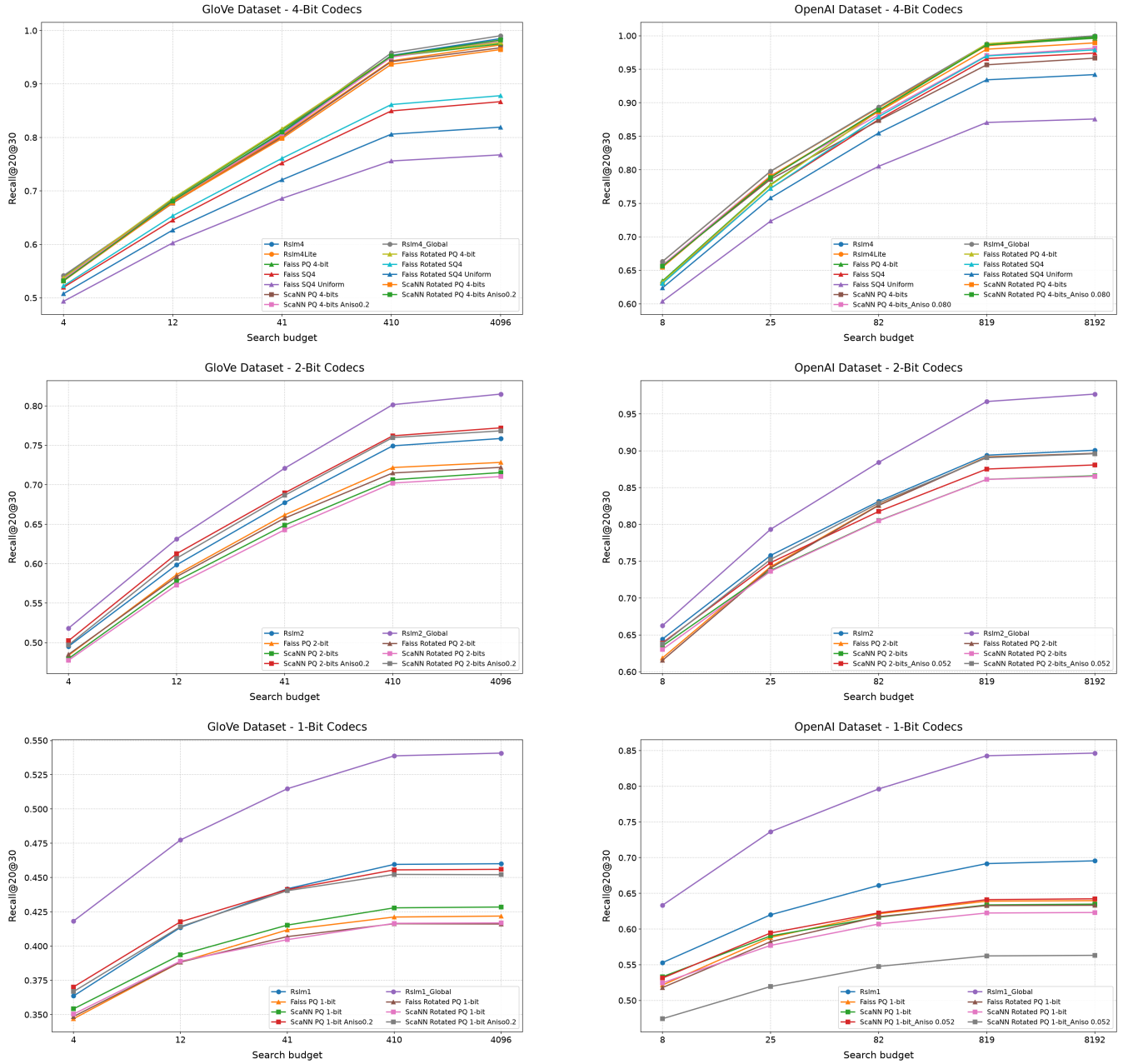


Figure 3: Recall@20@30 results for Faiss, ScaNN, and Rslm codecs across different search budgets.

7 Reproducibility

We provide a reference Python implementation of our codecs within a Colab notebook which reproduces quality metrics on full glove dataset, both for non-relative and relative quantizations.

See <https://github.com/google-research/google-research/tree/master/rslm>.

References

- [1] Cecilia Aguerrebere, Ishwar Singh Bhati, Mark Hildebrand, Mariano Tepper, and Theodore Willke. 2023. Similarity Search in the Blink of an Eye with Compressed Indices. *Proceedings of the VLDB Endowment* 16, 11 (2023), 3433–3446. <https://doi.org/10.14778/3611479.3611537>
- [2] Fabien André, Anne-Marie Kermarrec, and Nicolas Le Scouarnec. 2015. Cache locality is not enough: High-performance nearest neighbor search with product quantization fast scan. *Proceedings of the VLDB Endowment* 9, 4 (2015), 288–299.
- [3] Heesang Ann, Joongkyu Lee, and Min-hwan Oh. 2026. Block-Sphere Vector Quantization. *arXiv preprint arXiv:2605.19972* (2026).

- [4] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefer, and James Hensman. 2024. Quarot: Outlier-free 4-bit inference in rotated llms. *Advances in Neural Information Processing Systems* 37 (2024), 100213–100240.
- [5] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. 2020. ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms. *Information Systems* 87 (2020), 101374.
- [6] Alex Auvolat, Sarath Chandar, Pascal Vincent, Hugo Larochelle, and Yoshua Bengio. 2015. Clustering is Efficient for Approximate Maximum Inner Product Search. arXiv:1507.05910 [cs.LG] <https://arxiv.org/abs/1507.05910>
- [7] Jon Louis Bentley. 1975. Multidimensional binary search trees used for associative searching. *Commun. ACM* 18, 9 (1975), 509–517.
- [8] Ella Bingham and Heikki Mannila. 2001. Random projection in dimensionality reduction: applications to image and text data. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*. 245–250.
- [9] Sebastian Bruch. 2024. *Foundations of Vector Retrieval*. Springer Nature Switzerland. <https://doi.org/10.1007/978-3-031-55182-6>
- [10] Jerry Chee, Yaohui Cai, Volodymyr Kuleshov, and Christopher M De Sa. 2023. Quip: 2-bit quantization of large language models with guarantees. *Advances in neural information processing systems* 36 (2023), 4396–4429.
- [11] Qi Chen, Bing Zhao, Haidong Wang, Mingqin Li, Chuanjie Liu, Zengzhong Li, Mao Yang, and Jingdong Wang. 2021. Spann: Highly-efficient billion-scale approximate nearest neighborhood search. *Advances in Neural Information Processing Systems* 34 (2021), 5199–5212.
- [12] Yongjian Chen, Tao Guan, and Cheng Wang. 2010. Approximate nearest neighbor search by residual vector quantization. *Sensors* 10, 12 (2010), 11259–11273.
- [13] Arman Cohan, Franck Dernoncourt, Doo Soon Kim, Trung Bui, Seokhwan Kim, Walter Chang, and Nazi Goharian. 2018. A Discourse-Aware Attention Model for Abstractive Summarization of Long Documents. *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)* (2018). <https://doi.org/10.18653/v1/n18-2097>
- [14] Stuart Coles. 2001. *An Introduction to Statistical Modeling of Extreme Values*. Springer London, London. <https://doi.org/10.1007/978-1-4471-3675-0>
- [15] Xinyan Dai, Xiao Yan, Kelvin KW Ng, Jie Liu, and James Cheng. 2020. Norm-Explicit Quantization: Improving Vector Quantization for Maximum Inner Product Search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 34. 3702–3709.
- [16] Sanjoy Dasgupta and Kaushik Sinha. 2013. Randomized partition trees for exact nearest neighbor search. In *Conference on learning theory*. PMLR, 317–337.
- [17] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2025. The faiss library. *IEEE Transactions on Big Data* (2025).
- [18] Jianyang Gao, Yutong Gou, Yuxuan Xu, Yongyi Yang, Cheng Long, and Raymond Chi-Wing Wong. 2025. Practical and asymptotically optimal quantization of high-dimensional vectors in euclidean space for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–26.
- [19] Jianyang Gao and Cheng Long. 2024. RaBitQ: Quantizing high-dimensional vectors with a theoretical error bound for approximate nearest neighbor search. *Proceedings of the ACM on Management of Data* 2, 3 (2024), 1–27.
- [20] Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Meng Wang, and Haofen Wang. 2024. Retrieval-Augmented Generation for Large Language Models: A Survey. arXiv:2312.10997 [cs.CL] <https://arxiv.org/abs/2312.10997>
- [21] Tiezheng Ge, Kaiming He, Qifa Ke, and Jian Sun. 2014. Optimized Product Quantization. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 36, 4 (2014), 744–755. <https://doi.org/10.1109/TPAMI.2013.240>
- [22] Robert M. Gray and David L. Neuhoff. 1998. Quantization. *IEEE Transactions on Information Theory* 44, 6 (1998), 2325–2383.
- [23] Ruiqi Guo, Philip Sun, Erik Lindgren, Quan Geng, David Simcha, Felix Chern, and Sanjiv Kumar. 2020. Accelerating large-scale inference with anisotropic vector quantization. In *International Conference on Machine Learning*. PMLR, 3887–3896.
- [24] Liulu He, XuanAng Liu, Juntao Liu, Taolue Feng, Ting Lu, Chunsheng Gan, Zhiyong Peng, Yuan Du, Huanrui Yang, Yijiang Liu, et al. 2026. LiftQuant: Continuous Bit-Width LLM via Dimensional Lifting and Projection. arXiv preprint arXiv:2606.04050 (2026).
- [25] Suhas Jayaram Subramanya, Fnu Devvrit, Harsha Vardhan Simhadri, Ravishankar Krishnawamy, and Rohan Kadekodi. 2019. DiskANN: Fast accurate billion-point nearest neighbor search on a single node. *Advances in neural information processing systems* 32 (2019).
- [26] Hervé Jégou, Romain Tavenard, Matthijs Douze, and Laurent Amsaleg. 2011. Searching in One Billion Vectors: Re-evaluating the State of the Art. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 1817–1824.
- [27] Hervé Jégou, Matthijs Douze, and Cordelia Schmid. 2011. Product Quantization for Nearest Neighbor Search. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 33, 1 (2011), 117–128. <https://doi.org/10.1109/TPAMI.2010.57>
- [28] Vladimir Karpukhin, Barlas Öğüz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. arXiv:2004.04906 [cs.CL] <https://arxiv.org/abs/2004.04906>
- [29] Leonardo Kuffo, Ioanna Tsakalidou, Roberta De Viti, Albert Angel, Jiri Iša, and Rastislav Lenhardt. 2026. Semantic Recall for Vector Search. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval* (Melbourne, VIC, Australia) (SIGIR '26). Association for Computing Machinery, New York, NY, USA. <https://doi.org/10.1145/3805712.3809894>
- [30] Donghyun Lee, Jitesh Chavan, Duy Nguyen, Sam Huang, Liming Jiang, Priyadarshini Panda, Timo Mertens, and Saurabh Shukla. 2026. OrbitQuant: Data-Agnostic Quantization for Image and Video Diffusion Transformers. arXiv:2607.02461 [cs.CV] <https://arxiv.org/abs/2607.02461>
- [31] Yang Li, Kangbo Liu, Ranjan Satapathy, Suhang Wang, and Erik Cambria. 2023. Recent Developments in Recommender Systems: A Survey. arXiv:2306.12680 [cs.IR] <https://arxiv.org/abs/2306.12680>
- [32] Zirui Liu, Jiayi Yuan, Hongye Jin, Shaochen Zhong, Zhaozhuo Xu, Vladimir Braverman, Beidi Chen, and Xia Hu. 2024. Kivi: A tuning-free asymmetric 2bit quantization for kv cache. arXiv preprint arXiv:2402.02750 (2024).
- [33] Le Ma, Ran Zhang, Yikun Han, Shirui Yu, Zaitian Wang, Zhiyuan Ning, Jinghan Zhang, Ping Xu, Pengjiang Li, Ziyue Qiao, Wei Ju, Chong Chen, Dongjie Wang, Kunpeng Liu, Pengyang Wang, Pengfei Wang, Yanjie Fu, Chunjiang Liu, Yuanchun Zhou, and Chang-Tien Lu. 2026. A Comprehensive Survey on Vector Database: Storage and Retrieval Technique, Challenge. arXiv:2310.11703 [cs.DB] <https://arxiv.org/abs/2310.11703>
- [34] Solmaz Seyed Monir, Irene Lau, Shubing Yang, and Dongfang Zhao. 2024. VectorSearch: Enhancing Document Retrieval with Semantic Embeddings and Optimized Search. arXiv:2409.17383 [cs.IR] <https://arxiv.org/abs/2409.17383>
- [35] Alexander Novikov, Ngan Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. 2025. AlphaEvolve: A coding agent for scientific and algorithmic discovery. arXiv (2025). <https://doi.org/10.48550/arxiv.2506.13131>
- [36] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). Association for Computational Linguistics, Doha, Qatar, 1532–1543. <https://doi.org/10.3115/v1/D14-1162>
- [37] Shaina Raza, Mizanur Rahman, Saifullah Kamawal, Armin Toroghi, Ananya Raval, Farshad Navah, and Amirmohammad Kazemeini. 2025. A Comprehensive Review of Recommender Systems: Transitioning from Theory to Practice. arXiv:2407.13699 [cs.IR] <https://arxiv.org/abs/2407.13699>
- [38] Chaitanya Sharma. 2025. Retrieval-Augmented Generation: A Comprehensive Survey of Architectures, Enhancements, and Robustness Frontiers. arXiv:2506.00054 [cs.IR] <https://arxiv.org/abs/2506.00054>
- [39] Jifan Shi, Jianyang Gao, James Xia, Tamás Béla Fehér, and Cheng Long. 2026. GPU-Native Approximate Nearest Neighbor Search with IVF-RaBitQ: Fast Index Build and Search. arXiv preprint arXiv:2602.23999 (2026).
- [40] Philip Sun, David Simcha, Dave Dopson, Ruiqi Guo, and Sanjiv Kumar. 2023. SOAR: Improved Indexing for Approximate Nearest Neighbor Search. In *Advances in Neural Information Processing Systems*, Vol. 36. Curran Associates, Inc., 3189–3204.
- [41] Théophane Vallaeys, Matthew J Muckley, Jakob Verbeek, and Matthijs Douze. 2025. Qinc02: Vector compression and search with improved implicit neural codebooks. In *International Conference on Learning Representations*, Vol. 2025. 85467–85484.
- [42] Shay Vargaftik, Ran Ben-Basat, Amit Portnoy, Gal Mendelson, and Yaniv Ben-Itzhak. 2022. EDEN: Communication-Efficient and Robust Distributed Mean Estimation for Federated Learning. In *International Conference on Machine Learning (ICML) (Proceedings of Machine Learning Research)*, Vol. 162. 21984–22014. <https://proceedings.mlr.press/v162/vargaftik22a/vargaftik22a.pdf>
- [43] Thomas Vecchiato. 2024. Learning Cluster Representatives for Approximate Nearest Neighbor Search. arXiv:2412.05921 [cs.IR] <https://arxiv.org/abs/2412.05921>
- [44] Jianguo Wang, Xiaomeng Yi, Rui Mao, et al. 2021. Milvus: A Purpose-Built Vector Data Management System. In *Proceedings of the 2021 International Conference on Management of Data (SIGMOD '21)*. Association for Computing Machinery, 2614–2627. <https://doi.org/10.1145/3448016.3457550>
- [45] Yang Wang, Lu Yu, Jinbin Zhang, and Qiyuan Zhang. 2026. Pyramid Product Quantization for Approximate Nearest Neighbor Search. *Applied Sciences* 16, 2 (2026), 853.
- [46] Bernard Widrow. 1956. A study of rough amplitude quantization by means of Nyquist sampling theory. *IRE Transactions on Circuit Theory* 3, 4 (1956), 266–276.
- [47] Bernard Widrow and István Kollár. 2008. *Quantization Noise: Roundoff Error in Digital Computation, Signal Processing, Control and Communications*. Cambridge

- University Press, Cambridge.
- [48] Songhao Wu et al. 2025. PolarQuant: Leveraging Polar Transformation for Efficient Key Cache Quantization and Decoding Acceleration. *arXiv preprint arXiv:2502.00527* (2025).
- [49] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. 2023. Smoothquant: Accurate and efficient post-training quantization for large language models. In *International conference on machine learning*. PMLR, 38087–38099.
- [50] Yuzhuang Xu, Shiyu Ji, Qingfu Zhu, and Wanxiang Che. 2025. CRVQ: Channel-relaxed vector quantization for extreme compression of LLMs. *Transactions of the Association for Computational Linguistics* 13 (2025), 1488–1506.
- [51] Artur Zagitov, Gleb Molodtsov, and Aleksandr Beznosikov. 2026. HARP: Hadamard-Preconditioned Adaptive Rotation Processor for Extreme LLM Quantization. *arXiv preprint arXiv:2605.29843* (2026).
- [52] Amir Zandieh, Majid Daliri, Majid Hadian, and Vahab Mirrokni. 2026. TurboQuant: Online Vector Quantization with Near-optimal Distortion Rate. In *International Conference on Learning Representations (ICLR)*.
- [53] Amir Zandieh, Majid Daliri, and Insu Han. 2025. Qjl: 1-bit quantized jl transform for kv cache quantization with zero overhead. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 39. 25805–25813.
- [54] Xi Zhao, Bolong Zheng, Xiaomeng Yi, Xiaofan Luan, Charles Xie, Xiaofang Zhou, and Christian S Jensen. 2023. FARGO: Fast maximum inner product search via global multi-probing. *Proceedings of the VLDB Endowment* 16, 5 (2023), 1100–1112.