Clark Hash

Stateless Sparse Johnson-Lindenstrauss Quantization for Neural Embeddings

Clark Labs Inc. | Autoresearch | Stanislav Kirdey

Vector bytes	Compression	Memory saved
1536 -> 48	**32x**	**96.875%**
384-dim f32 to default cosine code	0.03125 storage ratio	same vector count, smaller footprint

Scope and claim

Clark Hash packages a specific stateless engineering design: deterministic sparse signed JL sketches, fixed scalar quantization, and asymmetric sketch-space scoring. It does not claim first invention of JL transforms, feature hashing, scalar quantization, or compressed vector retrieval.

1. Abstract

Clark Hash is a compact embedding-storage package for systems that need to encode vectors online. A database embedding is normalized, projected through a deterministic sparse signed Johnson-Lindenstrauss sketch, rescaled, clipped, and stored as a fixed-width scalar-quantized code. Queries stay in floating point and are scored asymmetrically against database-side sketches. The default 384-dimensional profile stores a cosine-search vector in 48 bytes instead of 1536 bytes for dense `f32` storage. The design is intentionally simple: no corpus fitting, learned codebooks, rotations, or calibration tables are required before new vectors can be encoded.

2. Symbols

Symbol	Meaning
$x \in \mathbb{R}^d$	Input database embedding
$r \in \mathbb{R}^d$	Input query embedding
d	Input dimension
m	Sketch dimension
s	Hashes per input coordinate
b	Bits per quantized sketch coordinate
c	Symmetric clipping range
$h_{j(i)}$	Bucket hash for coordinate i and repetition j
$\sigma_{j(i)} \in \{-1, +1\}$	Sign hash

3. Sparse Signed JL Projection

Clark Hash uses a sparse random matrix $R \in \mathbb{R}^{m \times d}$. Each input coordinate has s non-zero contributions. Bucket locations and signs are derived deterministically from the seed, input coordinate, and repetition index:

$$R_{k,i} = \frac{1}{\sqrt{s}} \sum_{j=1}^{s} \sigma_{j(i)} \cdot 1\left[h_{j(i)} = k\right]$$

Equivalently, the projected coordinate is:

$$y_k = \sum_{i=1}^{d} \sum_{j=1}^{s} 1\left[h_{j(i)} = k\right] \cdot \sigma_{j(i)} \cdot \frac{x_i}{\sqrt{s}}$$

This is data-oblivious: the projection is independent of the corpus and does not need fitting before new vectors can be encoded.

3.1. Inner-product centering

For fixed vectors u and v, the signed-hash estimator is centered around the original inner product:

$$E[\langle Ru, Rv \rangle] = \langle u, v \rangle$$

The variance depends on sketch dimension, sparsity, and collisions. Increasing m reduces projection noise at the cost of storage. Increasing s usually reduces sparse projection noise at the cost of encode CPU.

4. Direction Normalization

For cosine search, Clark Hash sketches the unit direction:

$$\|x\|_2 = \sqrt{\sum_i x_i^2}, \quad u = \frac{x}{\|x\|_2}$$

The raw sketch is:

$$y = Ru$$

For a unit vector, each coordinate of Ru has scale roughly $\frac{1}{\sqrt{m}}$. Clark Hash rescales by $\sqrt{m}$ before quantization:

$$z = \sqrt{m} Ru$$

That makes sketch coordinates roughly unit scale, so a fixed clipping range such as $[-3, 3]$ is practical across sketch sizes.

5. Fixed Scalar Quantization

Let $L = 2^b - 1$. Each scaled coordinate is clipped and uniformly quantized:

$$z'_k = \text{clamp}(z_k, -c, c)$$

$$q_k = \text{round}\left(L \cdot \frac{z'_k + c}{2c} \right)$$

The database-side dequantizer is:

$$\hat{z}_k = \left(2c \cdot \frac{q_k}{L} \right) - c$$

The integer code stores exactly b bits per coordinate. The quantization step is:

$$\Delta = 2\frac{c}{L}$$

Without clipping, scalar quantization error per coordinate is bounded by:

$$|\hat{z}_k - z_k| \le \frac{\Delta}{2}$$

Clipping adds the residual term:

$$\text{clip\_error}_k = z_k - \text{clamp}(z_k, -c, c)$$

The implementation exposes `clip` so workloads with heavier-tailed sketch coordinates can trade quantization resolution against clipping frequency.

6. Asymmetric Cosine Scoring

Queries stay in floating point. For query r, Clark Hash computes:

$$v = \frac{r}{\|r\|_2}, \quad a = \sqrt{m}Rv$$

The database vector stores the quantized and dequantized sketch $\hat{z}$ for the database vector x. The asymmetric cosine estimate is:

$$\widehat{\cos}(r, x) = \frac{1}{m}\sum_{k=1}^{m} a_k \cdot \hat{z}_k$$

Without quantization, this estimator maps back to cosine scale:

$$\frac{1}{m}\langle\sqrt{m}Rv, \sqrt{m}Ru\rangle = \langle Rv, Ru\rangle \approx \langle v, u\rangle = \cos(r, x)$$

Quantization adds scalar distortion and clipping error on the database side while preserving the query in floating point. A simple quantization-only score perturbation bound is:

$$|\text{score\_error}| \le \frac{1}{m}\sum_{k=1}^{m}|a_k| \cdot \frac{\Delta}{2}$$

plus the analogous clipping term using $|\text{clip\_error}_k|$.

7. Dot-Product Mode

Cosine mode stores only the normalized direction sketch. Dot-product mode also stores a two-byte log-norm side channel:

$$\ell = \text{clamp}(\log_2(\|x\|_2), \ell_{\min}, \ell_{\max})$$

$$n = \text{round}\left(65535 \cdot \frac{\ell - \ell_{\min}}{\ell_{\max} - \ell_{\min}}\right)$$

Decode:

$$\hat{\ell} = \ell_{\min} + n \cdot \frac{\ell_{\max} - \ell_{\min}}{65535}$$

$$\widehat{\|x\|_2} = 2^{\hat{\ell}}$$

Final dot-product estimate:

$$\text{dot\_hat}(r, x) = \text{cos\_hat}(r, x) \cdot \|r\|_2 \cdot \widehat{\|x\|_2}$$

The norm channel costs 2 bytes per vector.

8. Storage Math

For cosine mode:

$$\text{bytes}_{\text{cosine}} = \text{ceil}\left(m \cdot \frac{b}{8}\right)$$

For dot-product mode:

$$\text{bytes}_{\text{dot}} = \text{ceil}\left(m \cdot \frac{b}{8}\right) + 2$$

For the default benchmark configuration:

$$d = 384, \quad m = 96, \quad b = 4, \quad s = 4$$

$$\text{dense\_bytes} = 384 \cdot 4 = 1536$$

$$\text{clark\_hash\_bytes} = \text{ceil}\left(96 \cdot \frac{4}{8}\right) = 48$$

$$\text{compression\_ratio} = \frac{48}{1536} = 0.03125$$

Operational result

The default profile stores each 384-dimensional vector in 48 bytes instead of 1536 bytes. That is 32x smaller, or 96.875% less vector memory, while keeping vectors searchable through asymmetric sketch-space scoring.

9. Operational Contract

1. Choose d, m, b, s, c, and `seed`.
2. Keep the same config for a collection.
3. Encode every database vector independently.
4. Sketch each query in floating point.
5. Score compressed vectors with asymmetric dot products in sketch space.

Clark Hash is an online, deterministic, and compact vector representation that can be used directly for evaluation, flat compressed scans, or as a storage layer inside larger retrieval systems.

10. MLA Citation

Clark Labs Inc., Autoresearch, and Stanislav Kirdey. “Clark Hash: Stateless Sparse Johnson-Lindenstrauss Quantization for Neural Embeddings.” Clark Labs Inc., 2026, GitHub, https://github.com/clark-labs-inc/clark-hash.