Randomized SVD: Theory, Implementation, and Empirical Analysis of Sketching Methods

Peter Klivnoy

December 2025

Abstract

Randomized algorithms for low-rank matrix approximation have become essential tools in modern data science, offering dramatic speedups over classical deterministic methods. This report provides a comprehensive empirical study of randomized SVD with three families of sketching operators: Gaussian random projections, Subsampled Randomized Fourier/Hadamard Transforms (SRFT/SRHT), and sparse embeddings (CountSketch). We implement each method in both optimized (using BLAS/FFT libraries) and naive pure-Python forms to disentangle algorithmic complexity gains from implementation-level optimizations. Our experiments demonstrate that while structured sketches (SRFT/SRHT) achieve the theoretically predicted $\mathcal{O}(mn \log \ell)$ complexity advantages in pure-Python comparisons, highly optimized BLAS routines make Gaussian sketches competitive for practical problem sizes. For sparse matrices, CountSketch provides dramatic speedups scaling with the number of nonzeros rather than matrix dimensions. We also investigate the role of power iterations and oversampling in controlling approximation accuracy, validating theoretical predictions from Halko et al. [2011]. All experiments are fully reproducible via the accompanying code repository.

Contents

1 Introduction

The singular value decomposition (SVD) is a foundational tool in numerical linear algebra with applications spanning principal component analysis, latent semantic indexing, recommender systems, image compression, and scientific computing. Given a matrix $A \in \mathbb{R}^{m \times n}$, the SVD factorizes it as

$$A = U \Sigma V^T \tag{1}$$

where $U \in \mathbb{R}^{m \times m}$ and $V \in \mathbb{R}^{n \times n}$ are orthogonal matrices and $\Sigma \in \mathbb{R}^{m \times n}$ is diagonal with non-negative entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$.

Computing the full SVD requires $\mathcal{O}(\min(m,n)^2 \cdot \max(m,n))$ operations, which becomes prohibitive for large-scale problems. In many applications, however, we only need the *truncated* or *rank-k* SVD, retaining only the top k singular values and corresponding singular vectors. The Eckart-Young theorem guarantees that this truncated SVD provides the best rank-k approximation in both spectral and Frobenius norms.

1.1 Randomized Low-Rank Approximation

Randomized algorithms offer a compelling alternative: by using random projections to "sketch" the matrix into a lower-dimensional subspace, we can compute approximate low-rank factorizations in $\mathcal{O}(mn \log k)$ or even $\mathcal{O}(\text{nnz}(A) \cdot k)$ time for sparse matrices, with high probability of achieving near-optimal accuracy [Halko et al., 2011, Woodruff, 2014, Martinsson & Tropp, 2020].

The core idea is elegantly simple. To find a rank-k approximation of A:

1. **Sketch:** Form $Y = A\Omega$ where $\Omega \in \mathbb{R}^{n \times \ell}$ is a random "test matrix" with $\ell = k + p$ columns (where p is a small oversampling parameter).
2. **Orthogonalize:** Compute an orthonormal basis Q for the column space of Y.
3. **Project and factor:** Form the small matrix $B = Q^T A$ and compute its SVD.
4. **Reconstruct:** The approximate SVD of A is recovered from Q and the SVD of B.

The computational bottleneck is Step 1: forming the sketch $Y = A\Omega$. The choice of sketching matrix Ω determines both the cost of this step and the quality of the resulting approximation.

1.2 Research Questions

This report investigates three fundamental questions:

1. **Algorithmic complexity vs. implementation:** Structured sketches (SRFT/SRHT) have better asymptotic complexity than Gaussian sketches. Does this translate to practical speedups, or do highly optimized BLAS libraries close the gap?
2. **Sparse data:** For sparse matrices, do sparse embeddings like CountSketch provide the theoretically predicted advantages over dense sketching methods?
3. **Accuracy control:** How do power iterations and oversampling affect approximation quality across different spectral decay profiles?

1.3 Contributions

Our main contributions are:

- A unified implementation of Gaussian, SRFT, SRHT, and CountSketch operators with both optimized and pure-Python variants.

- Systematic benchmarks isolating algorithmic complexity from library optimizations.
- Empirical validation of theoretical accuracy bounds across diverse spectral decay profiles.
- Practical guidelines for practitioners choosing among sketching methods.

2 Background and Theory

We briefly review the theoretical foundations underlying randomized SVD, focusing on the properties that different sketching matrices must satisfy and their computational costs.

2.1 The Randomized SVD Algorithm

Algorithm 1 presents the basic randomized SVD procedure.

Algorithm 1 Randomized SVD [Halko et al., 2011]

Require: Matrix $A \in \mathbb{R}^{m \times n}$, target rank k, oversampling p, power iterations q
Ensure: Approximate rank-k SVD: $\tilde{U}, \tilde{\Sigma}, \tilde{V}$
1: Set $\ell = k + p$
2: Draw random test matrix $\Omega \in \mathbb{R}^{n \times \ell}$
3: Form sketch $Y = (AA^T)^q A\Omega$ ▷ Power iteration for accuracy
4: Compute QR factorization $Y = QR$
5: Form $B = Q^T A \in \mathbb{R}^{\ell \times n}$
6: Compute SVD of small matrix: $B = \hat{U}\Sigma V^T$
7: Set $\tilde{U} = Q\hat{U}$
8: **return** $\tilde{U}, \tilde{\Sigma}, \tilde{V} = V$

2.2 Sketching Matrices

The choice of Ω critically affects both computational cost and approximation quality.

2.2.1 Gaussian Random Matrices

The simplest choice is $\Omega_{ij} \sim \mathcal{N}(0, 1)$ i.i.d. Gaussian entries.

- **Complexity:** $\mathcal{O}(mn\ell)$ to form $Y = A\Omega$
- **Accuracy:** Optimal among oblivious sketches; well-understood error bounds
- **Implementation:** Leverages highly optimized BLAS (DGEMM)

2.2.2 Subsampled Randomized Fourier Transform (SRFT)

The SRFT uses structured randomness:

$$\Omega = \sqrt{\frac{n}{\ell}} DF^* S \tag{2}$$

where D is a diagonal matrix of random signs, F is the DFT matrix, and S samples ℓ columns uniformly at random.

- **Complexity:** $\mathcal{O}(mn \log n)$ using FFT, or $\mathcal{O}(mn \log \ell)$ with more sophisticated implementations
- **Accuracy:** Comparable to Gaussian with slightly larger oversampling
- **Implementation:** Uses FFT libraries (FFTW, NumPy FFT)

2.2.3 Subsampled Randomized Hadamard Transform (SRHT)

The SRHT replaces the DFT with the Walsh-Hadamard transform:

$$\Omega = \sqrt{\frac{n}{\ell}} DHS \tag{3}$$

where H is the Hadamard matrix (requires n to be a power of 2, or padding).

- **Complexity:** $\mathcal{O}(mn \log n)$ using Fast Walsh-Hadamard Transform
- **Accuracy:** Similar to SRFT
- **Implementation:** Pure integer/real arithmetic; no complex numbers

2.2.4 CountSketch (Sparse Embeddings)

For sparse matrices, CountSketch [Woodruff, 2014] uses an extremely sparse Ω:

$$\Omega_{ij} = \begin{cases} +1 & \text{if } h(i) = j \text{ and } s(i) = +1 \\ -1 & \text{if } h(i) = j \text{ and } s(i) = -1 \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

where $h : [n] \to [\ell]$ is a random hash function and $s : [n] \to \{+1, -1\}$ is a random sign function.

- **Complexity:** $\mathcal{O}(\text{nnz}(A) \cdot \ell)$ — depends on sparsity, not dimensions
- **Accuracy:** Requires larger ℓ than Gaussian but still practical
- **Implementation:** Sparse matrix-matrix multiplication

2.3 Accuracy Guarantees

The fundamental accuracy result from Halko et al. [2011] states:

Theorem 1 (Halko et al., 2011)**.** Let $A \in \mathbb{R}^{m \times n}$ with singular values $\sigma_1 \geq \sigma_2 \geq \cdots$. For a Gaussian test matrix $\Omega \in \mathbb{R}^{n \times (k+p)}$ with $p \geq 2$, the rank-k approximation $\tilde{A}_k$ satisfies

$$\mathbb{E}\left[\left\|A - \tilde{A}_k\right\|_F^2\right] \leq \left(1 + \frac{k}{p-1}\right) \sum_{j>k} \sigma_j^2 \tag{5}$$

and with q power iterations:

$$\mathbb{E}\left[\left\|A - \tilde{A}_k\right\|_F^2\right] \leq \left(1 + \frac{k}{p-1}\right)^{1/(2q+1)} \left(\sum_{j>k} \sigma_j^{4q+2}\right)^{1/(2q+1)} \tag{6}$$

Remark 1. Power iterations dramatically improve accuracy when the singular value spectrum decays slowly. Each power iteration "raises" the spectrum to a higher power, effectively creating a faster-decaying spectrum.

2.4 Complexity Summary

Table 1 summarizes the computational complexity of forming the sketch $Y = A\Omega$ for each method.

Table 1: Complexity of sketch formation $Y = A\Omega$ for $A \in \mathbb{R}^{m \times n}$, sketch size ℓ

Method	Complexity	Notes
Gaussian	$\mathcal{O}(mn\ell)$	Optimal BLAS; no structure
SRFT	$\mathcal{O}(mn \log n)$	FFT-based; complex arithmetic
SRHT	$\mathcal{O}(mn \log n)$	FWHT; real arithmetic only
CountSketch	$\mathcal{O}(\text{nnz}(A) \cdot \ell)$	Sparse only; scales with nonzeros

3 Implementation

A key contribution of this work is providing *both* optimized and naive pure-Python implementations for each sketching method. This allows us to disentangle two distinct sources of performance differences:

1. **Algorithmic complexity:** The asymptotic scaling predicted by theory (e.g., $\mathcal{O}(mn\ell)$ vs. $\mathcal{O}(mn \log n)$).
2. **Implementation constants:** The efficiency of underlying libraries (BLAS, FFT) which can differ by orders of magnitude.

3.1 Repository Structure

Our implementation is organized as follows:

```
src/
  randsvd_algorithm.py    # Main randomized SVD
  structured_sketch.py    # SRFT and SRHT operators
  sparse_sketching.py     # CountSketch and sparse embeddings
  naive_sketching.py      # Pure-Python baselines
  hadamardKernel.py       # C-accelerated Hadamard (optional)
  utils.py                # Spectrum generation, utilities
tests/
  run_benchmark_*.py      # All experimental scripts
figures/                  # Generated plots
```

3.2 Optimized Implementations

Gaussian: Uses `numpy.random.randn` and `numpy.dot`, which dispatch to optimized BLAS (DGEMM).

SRFT: Uses `numpy.fft.fft` applied to each row, followed by column subsampling.

SRHT: Uses either a C-accelerated Fast Walsh-Hadamard Transform (when available) or a pure NumPy iterative implementation.

CountSketch: Constructs a sparse Ω using `scipy.sparse.csr_matrix` and uses sparse-sparse matrix multiplication.

3.3 Naive Pure-Python Implementations

To isolate algorithmic complexity, we implement each method using explicit Python loops with no library optimizations:

Naive Gaussian: Triple nested loop computing $Y_{ij} = \sum_k A_{ik}\Omega_{kj}$.

Naive FFT (DFT): $\mathcal{O}(n^2)$ discrete Fourier transform via the definition.

Naive FWHT: $\mathcal{O}(n \log n)$ recursive Hadamard using pure Python arithmetic.

Naive CountSketch: Direct column extraction based on hash indices.

These naive implementations are slow but reveal the *true* algorithmic complexity ratios without BLAS confounds.

4 Experimental Setup

4.1 Test Matrices

We construct synthetic matrices with controlled spectral decay profiles:

- **Exponential decay:** $\sigma_i = e^{-\alpha i}$ with $\alpha = 0.1$. Fast decay; easy for randomized methods.
- **Polynomial decay:** $\sigma_i = i^{-\beta}$ with $\beta = 1$ or $\beta = 2$. Moderate difficulty.
- **Slow decay:** $\sigma_i = 1/\sqrt{i}$ or flat spectrum with noise. Challenging; benefits from power iterations.

Matrix sizes: $n = m \in \{1024, 2048, 4096\}$ for square matrices; $m = 8000, n = 2000$ for tall matrices.

For sparse experiments: density $\in \{0.1\%, 1\%, 5\%, 10\%\}$.

4.2 Parameters

- Target rank: $k \in \{50, 100, 200\}$
- Oversampling: $p \in \{5, 10, 20\}$
- Sketch size: $\ell = k + p$
- Power iterations: $q \in \{0, 1, 2, 4\}$
- Trials: 5–10 random seeds per configuration

4.3 Metrics

Speed:

- Time to form sketch $Y = A\Omega$ (the computational bottleneck)
- Total randomized SVD runtime
- Speedup factor: $t_{\text{Gaussian}}/t_{\text{structured}}$

Accuracy:

- Relative Frobenius error: $\left\|A - \tilde{A}_k\right\|_F / \|A - A_k^*\|_F$ where A_k^* is the optimal truncated SVD
- Relative spectral error: $\left\|A - \tilde{A}_k\right\|_2 / \|A - A_k^*\|_2$
- Singular value errors: $|\sigma_i - \tilde{\sigma}_i|/\sigma_i$ for $i = 1, \ldots, k$

4.4 Hardware

All experiments run on [TODO: specify hardware]. Python 3.x with NumPy (linked to [TODO: BLAS version]), SciPy, and Matplotlib.

5 Speed Experiments

5.1 Dense Matrices: Optimized Implementations

Figure **??** shows runtime versus sketch size ℓ for optimized implementations on dense matrices.

Key observation: Despite having worse asymptotic complexity, Gaussian sketching is often *faster* than SRFT/SRHT for practical problem sizes due to highly optimized BLAS libraries. The crossover point where structured methods win depends on matrix size and the ratio ℓ/n.

5.2 Dense Matrices: Pure-Python (Algorithmic) Comparison

To isolate the algorithmic complexity advantage, we compare naive pure-Python implementations with no library optimization.

Figure **??** shows the "fair" algorithmic comparison.

Key observation: In the pure-Python setting, structured sketches show clear advantages:

- Naive Gaussian: $\mathcal{O}(mn\ell)$ — linear in ℓ
- Naive SRFT/SRHT: $\mathcal{O}(mn \log n)$ — nearly constant in ℓ

The speedup factor approaches $\ell/\log n$ as predicted by theory.

5.3 Sparse Matrices: CountSketch

For sparse matrices, CountSketch provides dramatic speedups over dense methods.

Figure **??** compares CountSketch against Gaussian on matrices of varying density.

Key observations:

- CountSketch time scales with $\text{nnz}(A)$, not matrix dimensions
- For 0.1% density, CountSketch achieves 10–100× speedup
- Advantage grows with matrix size and decreasing density

Pure-Python comparison confirms the algorithmic advantage is $\mathcal{O}(n)$ — CountSketch is fundamentally n times faster than Gaussian for the sketching step on sparse data.

6 Accuracy Experiments

6.1 Sketching Method Comparison

Figure **??** compares approximation quality across Gaussian, SRFT, and SRHT for different spectral decay profiles.

Key observation: All three sketching methods achieve nearly identical accuracy, validating the theoretical prediction that they differ primarily in computational cost, not approximation quality.

6.2 Effect of Power Iterations

Figure **??** shows how power iterations (q) affect accuracy for slow-decaying spectra.

Key observations:

- For fast decay (exponential), $q = 0$ suffices
- For slow decay (polynomial), $q = 1$ or $q = 2$ dramatically improves accuracy
- Diminishing returns beyond $q = 2$ for most practical cases
- "Burn-in" phenomenon: first iterations may show limited improvement before accuracy converges

6.3 Effect of Oversampling

Figure **??** shows how oversampling parameter p affects accuracy.

Key observation: Oversampling $p = 5$ to 10 provides a good balance between computational cost and accuracy. Larger p gives diminishing returns.

6.4 Sparse Embeddings Accuracy

Figure **??** confirms that CountSketch achieves comparable accuracy to Gaussian sketching for the same sketch size ℓ.

Key observation: Despite using a much sparser random matrix, CountSketch produces approximations within 1–3% of the optimal Eckart-Young bound, comparable to Gaussian.

7 Discussion

7.1 Theory vs. Practice

Our experiments reveal an important gap between theoretical complexity and practical performance:

1. **BLAS dominance:** For dense matrices at practical sizes ($n \lesssim 10000$), highly optimized BLAS makes Gaussian competitive despite its worse asymptotic complexity.
2. **Algorithmic wins are real:** Pure-Python comparisons confirm the $\mathcal{O}(mn\ell)$ vs. $\mathcal{O}(mn \log n)$ gap exists. It simply gets hidden by implementation constants.
3. **Sparse is different:** For sparse data, CountSketch provides unambiguous wins because the complexity depends on $\text{nnz}(A)$ rather than mn.

7.2 Practical Recommendations

Based on our experiments, we offer the following guidelines:

Table 2: Recommended sketching method by problem characteristics

Data Type	**Sketch Size ℓ**	**Recommended Method**
Dense, small–medium	Any	Gaussian (BLAS-optimized)
Dense, very large	$\ell \gg \log n$	SRHT or SRFT
Sparse ($\text{nnz} \ll mn$)	Any	CountSketch
Slow spectral decay	Any	Add power iterations ($q = 1, 2$)

7.3 Achieving $\mathcal{O}(mn \log k)$ Complexity

The literature suggests that SRFT/SRHT can achieve $\mathcal{O}(mn \log k)$ rather than $\mathcal{O}(mn \log n)$ complexity through careful implementation that avoids computing the full transform. Our current implementation achieves $\mathcal{O}(mn \log n)$. Implementing the $\mathcal{O}(mn \log k)$ variant is left for future work.

8 Conclusion

We have presented a comprehensive empirical study of randomized SVD with multiple sketching methods. Our key findings are:

1. **Algorithmic complexity matters, but so do constants:** Structured sketches (SRFT/SRHT) have provably better asymptotic complexity than Gaussian, but highly optimized BLAS libraries narrow the gap significantly for practical problem sizes.

2. **Pure-Python comparisons reveal true scaling:** By implementing naive versions of all methods, we confirm the theoretical complexity ratios without library confounds.

3. **CountSketch wins for sparse data:** For sparse matrices, CountSketch provides dramatic speedups that scale with nonzeros rather than matrix dimensions.

4. **Power iterations control accuracy:** For slowly decaying spectra, power iterations ($q = 1, 2$) are essential for achieving near-optimal accuracy.

5. **All methods achieve similar accuracy:** The choice of sketching method primarily affects speed, not approximation quality.

All code and experiments are available at `https://github.com/peterKlivnoy/RandSVD` for full reproducibility.

References

N. Halko, P.-G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

E. Liberty, F. Woolfe, P.-G. Martinsson, V. Rokhlin, and M. Tygert. Randomized algorithms for the low-rank approximation of matrices. *Proceedings of the National Academy of Sciences*, 104(51):20167–20172, 2007.

P.-G. Martinsson and J. A. Tropp. Randomized numerical linear algebra: Foundations and algorithms. *Acta Numerica*, 29:403–572, 2020.

D. P. Woodruff. Sketching as a tool for numerical linear algebra. *Foundations and Trends in Theoretical Computer Science*, 10(1–2):1–157, 2014.

J. A. Tropp. Improved analysis of the subsampled randomized Hadamard transform. *Advances in Adaptive Data Analysis*, 3(01n02):115–126, 2011.

K. L. Clarkson and D. P. Woodruff. Low-rank approximation and regression in input sparsity time. *Journal of the ACM*, 63(6):1–45, 2017.

V. Rokhlin, A. Szlam, and M. Tygert. A randomized algorithm for principal component analysis. *SIAM Journal on Matrix Analysis and Applications*, 31(3):1100–1124, 2010.

C. Musco and C. Musco. Randomized block Krylov methods for stronger and faster approximate singular value decomposition. *Advances in Neural Information Processing Systems*, 28, 2015.

A. K. Saibaba. Randomized subspace iteration: Analysis of canonical angles and unitarily invariant norms. *SIAM Journal on Matrix Analysis and Applications*, 40(1):23–48, 2019.