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

Peter Klivnoy

December 2025

Abstract

Randomized algorithms for low-rank matrix approximation have emerged as essential tools in modern data science, offering dramatic computational speedups over classical deterministic methods while maintaining provable accuracy guarantees. This report provides a comprehensive empirical investigation of randomized singular value decomposition (SVD) with three families of sketching operators: Gaussian random projections, Subsampled Randomized Fourier and Hadamard Transforms (SRFT/SRHT), and sparse embeddings based on the CountSketch construction. A distinguishing feature of our study is the implementation of each method in both optimized form—leveraging highly tuned numerical libraries—and in naive pure-Python form, allowing us to disentangle the algorithmic complexity advantages predicted by theory from the implementation-level optimizations that dominate practical performance. Our experiments reveal that while structured sketches achieve their theoretically predicted $\mathcal{O}(mn \log n)$ complexity advantages when library optimizations are removed, highly optimized BLAS routines make Gaussian sketches surprisingly competitive for matrices of practical size. For sparse matrices, CountSketch provides unambiguous speedups that scale with the number of nonzero entries rather than matrix dimensions, achieving speedups exceeding two orders of magnitude on matrices with 0.1% density. We additionally investigate the role of power iterations and oversampling in controlling approximation accuracy across different spectral decay profiles, providing empirical validation of theoretical bounds established by Halko, Martinsson, and Tropp. All experiments are fully reproducible via the accompanying code repository.

Contents

1 Introduction **3**
1.1 Randomized Low-Rank Approximation . . . 3
1.2 Research Questions . . . 3
1.3 Contributions . . . 3

2 Background and Theory **4**
2.1 The Randomized SVD Algorithm . . . 4
2.2 Sketching Matrices . . . 4
2.2.1 Gaussian Random Matrices . . . 4
2.2.2 Subsampled Randomized Fourier Transform (SRFT) . . . 4
2.2.3 Subsampled Randomized Hadamard Transform (SRHT) . . . 5
2.2.4 CountSketch (Sparse Embeddings) . . . 5
2.3 Accuracy Guarantees . . . 5
2.4 Complexity Summary . . . 5

3 Implementation **6**
3.1 Repository Structure . . . 6
3.2 Optimized Implementations . . . 6

3.3 Naive Pure-Python Implementations 6

4 Experimental Setup **7**

4.1 Test Matrices 7

4.2 Parameters 7

4.3 Metrics 7

4.4 Hardware 7

5 Speed Experiments **8**

5.1 Dense Matrices: Optimized Implementations 8

5.2 Dense Matrices: Pure-Python (Algorithmic) Comparison 8

5.3 Sparse Matrices: CountSketch 8

6 Accuracy Experiments **8**

6.1 Sketching Method Comparison 8

6.2 Effect of Power Iterations 8

6.3 Effect of Oversampling 9

6.4 Sparse Embeddings Accuracy 9

7 Discussion **9**

7.1 Theory vs. Practice 9

7.2 Practical Recommendations 9

7.3 Achieving $\mathcal{O}(mn \log k)$ Complexity 9

8 Conclusion **9**

1 Introduction

The singular value decomposition stands as one of the most fundamental tools in numerical linear algebra, with applications spanning principal component analysis, latent semantic indexing, recommender systems, image and signal processing, and scientific computing more broadly. Given a matrix $A \in \mathbb{R}^{m\times n}$, the SVD provides the factorization $A = U\Sigma V^T$ where $U \in \mathbb{R}^{m\times m}$ and $V \in \mathbb{R}^{n\times n}$ are orthogonal matrices containing the left and right singular vectors, and $\Sigma \in \mathbb{R}^{m\times n}$ is a diagonal matrix with non-negative entries $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)} \geq 0$ arranged in decreasing order.

The computational cost of computing the full SVD scales as $\mathcal{O}(\min(m, n)^2 \cdot \max(m, n))$ operations, which becomes prohibitive as matrices grow to dimensions encountered in modern applications—genomics datasets with millions of features, image collections with millions of pixels, or recommendation systems with billions of user-item interactions. Fortunately, many applications require only the *truncated* or *rank-k* SVD, retaining the top k singular values and their corresponding singular vectors. The celebrated Eckart-Young theorem guarantees that this truncated decomposition provides the best rank-k approximation in both spectral and Frobenius norms, making it the natural target for dimensionality reduction.

Randomized algorithms offer a compelling path forward. By employing random projections to "sketch" the input matrix into a lower-dimensional subspace that captures its dominant range, one can compute approximate low-rank factorizations in $\mathcal{O}(mn \log k)$ time—or even $\mathcal{O}(\text{nnz}(A) \cdot k)$ time for sparse matrices—while achieving near-optimal accuracy with high probability. The theoretical foundations of these methods were systematically developed by Halko, Martinsson, and Tropp in their influential 2011 survey [Halko et al., 2011], building on earlier work by Liberty, Woolfe, Rokhlin, and others [Liberty et al., 2007, Rokhlin et al., 2010]. The monograph by Martinsson and Tropp [Martinsson & Tropp, 2020] provides a comprehensive modern treatment.

1.1 The Core Algorithmic Framework

The randomized SVD algorithm proceeds through a conceptually simple sequence of steps. To approximate the rank-k truncated SVD of a matrix $A \in \mathbb{R}^{m\times n}$, one first draws a random "test matrix" $\Omega \in \mathbb{R}^{n\times \ell}$ with $\ell = k + p$ columns, where p is a small oversampling parameter typically between 5 and 20. The sketch $Y = A\Omega$ is then formed, which has dimensions $m \times \ell$—dramatically smaller than the original matrix when $\ell \ll n$. An orthonormal basis Q for the column space of Y is computed via QR factorization, and the original matrix is projected onto this basis to form the small matrix $B = Q^T A \in \mathbb{R}^{\ell\times n}$. Finally, the SVD of B is computed directly, and the approximate singular vectors of A are recovered by rotating through Q.

The computational bottleneck in this procedure is the formation of the sketch $Y = A\Omega$. For a dense Gaussian test matrix, this requires $\mathcal{O}(mn\ell)$ floating-point operations—equivalent to a dense matrix-matrix multiplication. The choice of sketching matrix Ω therefore critically affects both the computational cost and the quality of the resulting approximation.

1.2 Research Questions and Contributions

This report investigates three interconnected questions that bridge theory and practice in randomized numerical linear algebra.

First, we ask whether the asymptotic complexity advantages of structured random matrices translate into practical speedups, or whether highly optimized numerical libraries narrow the gap sufficiently to make simpler Gaussian sketches competitive. Theoretical analysis establishes that the Subsampled Randomized Fourier Transform (SRFT) and Subsampled Randomized Hadamard Transform (SRHT) achieve $\mathcal{O}(mn \log n)$ complexity for sketch formation, compared to $\mathcal{O}(mn\ell)$ for Gaussian projections. For large sketch sizes ℓ, this represents a significant theoretical advantage. However, modern BLAS implementations achieve near-peak floating-point throughput for dense matrix multiplication, while FFT and Hadamard implementations may not be as finely tuned.

Second, we examine the effectiveness of sparse embeddings—particularly the CountSketch construction of Clarkson and Woodruff [Clarkson & Woodruff, 2017, Woodruff, 2014]—for matrices with significant sparsity structure. When the input matrix A has only $\text{nnz}(A)$ nonzero entries, CountSketch can form the sketch in $\mathcal{O}(\text{nnz}(A))$ time, independent of the matrix dimensions. This represents a fundamentally different scaling

Algorithm 1 Randomized SVD with Power Iterations [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 sketch size $\ell = k + p$
 2: Draw random test matrix $\Omega \in \mathbb{R}^{n \times \ell}$
 3: Form initial sketch $Y = A\Omega$
 4: for $j = 1, \dots, q$ do                                        ▷ Power iterations
 5:     $\tilde{Y} = A^T Y$; compute QR: $\tilde{Y} = \tilde{Q}\tilde{R}$
 6:     $Y = A\tilde{Q}$; compute QR: $Y = QR$
 7: end for
 8: Compute thin QR factorization $Y = QR$
 9: Form small matrix $B = Q^T A \in \mathbb{R}^{\ell \times n}$
10: Compute SVD: $B = \hat{U}\Sigma V^T$
11: Set $\tilde{U} = Q\hat{U}$
12: return $\tilde{U}_{:,1:k}, \Sigma_{1:k,1:k}, V_{:,1:k}$
```

behavior that should dominate for sufficiently sparse data.

Third, we investigate the practical impact of algorithmic parameters—power iterations and oversampling—on approximation accuracy across different spectral decay profiles. Theory predicts that power iterations dramatically improve accuracy for slowly decaying spectra by effectively "sharpening" the spectral gap, while oversampling provides a safety margin that ensures the random subspace captures the desired singular vectors with high probability.

Our primary contributions are as follows. We provide a unified implementation of Gaussian, SRFT, SRHT, and CountSketch operators with both optimized and naive pure-Python variants, enabling systematic comparison of algorithmic complexity versus implementation efficiency. We conduct comprehensive benchmarks that isolate these effects, demonstrating that while structured sketches achieve their theoretical complexity advantages in pure-Python comparisons, optimized BLAS implementations make Gaussian competitive for matrices up to dimension 10^4. For sparse matrices, we show that CountSketch achieves speedups exceeding $300\times$ compared to naive Gaussian sketching on matrices with 0.1% density. Finally, we provide empirical validation of theoretical accuracy bounds, confirming that all sketching methods achieve similar approximation quality and that power iterations and oversampling provide the primary levers for controlling accuracy.

2 Theoretical Background

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

2.1 The Randomized Range Finder

The core insight of randomized SVD is that a good basis for the column space of A can be found by examining how A acts on random vectors. If $\Omega \in \mathbb{R}^{n \times \ell}$ is a random matrix with suitable properties, then the columns of $Y = A\Omega$ span a subspace that, with high probability, captures most of the action of A. More precisely, if Q is an orthonormal basis for the column space of Y, then $\|A - QQ^T A\|$ is small when ℓ slightly exceeds the numerical rank of A.

Algorithm 1 presents the complete randomized SVD procedure, including the optional power iteration scheme that improves accuracy for matrices with slowly decaying singular values.

The power iteration scheme replaces the simple sketch $Y = A\Omega$ with $Y = (AA^T)^q A\Omega$. This has the effect of raising the singular values of A to the $(2q + 1)$-th power before the randomized range finding step, dramatically increasing the gap between retained and discarded singular values when the original spectrum decays slowly.

2.2 Sketching Matrix Constructions

The choice of random test matrix Ω determines both computational cost and approximation quality. We consider four constructions.

Gaussian Random Matrices. The simplest and most widely analyzed choice is to draw Ω with independent standard Gaussian entries: $\Omega_{ij} \sim \mathcal{N}(0,1)$. Gaussian matrices satisfy strong concentration inequalities and achieve optimal accuracy bounds among oblivious sketching methods. The sketch $Y = A\Omega$ requires $\mathcal{O}(mn\ell)$ operations to compute, equivalent to a dense matrix-matrix multiplication. In practice, this leverages highly optimized BLAS routines (specifically DGEMM) that achieve near-peak floating-point throughput on modern hardware.

Subsampled Randomized Fourier Transform (SRFT). The SRFT exploits structure to reduce computational cost. The test matrix takes the form $\Omega = \sqrt{n/\ell} \cdot DF^*S$, where D is a diagonal matrix of independent random signs, F is the $n \times n$ discrete Fourier transform matrix, and S is an $n \times \ell$ matrix that samples ℓ columns uniformly at random. The sketch $Y = A\Omega$ can be computed in $\mathcal{O}(mn \log n)$ operations by applying the FFT to each row of DA and then extracting the sampled columns. The factor $\sqrt{n/\ell}$ ensures the expected squared norm is preserved.

Subsampled Randomized Hadamard Transform (SRHT). The SRHT replaces the Fourier transform with the Walsh-Hadamard transform: $\Omega = \sqrt{n/\ell} \cdot DHS$, where H is the $n \times n$ Hadamard matrix defined recursively by $H_1 = [1]$ and $H_{2^k} = H_{2^{k-1}} H_{2^{k-1}}$
$H_{2^{k-1}} - H_{2^{k-1}}$. The Fast Walsh-Hadamard Transform (FWHT) computes the transform in $\mathcal{O}(n \log n)$ operations using only additions and subtractions—no complex arithmetic or trigonometric functions. This makes the SRHT particularly efficient on integer or fixed-point hardware. For matrices whose column dimension is not a power of two, zero-padding is required.

CountSketch (Sparse Embeddings). For sparse matrices, the CountSketch construction of Clarkson and Woodruff provides an extremely sparse test matrix. Each column of Ω contains exactly one nonzero entry: $\Omega_{h(i),i} = s(i)$ where $h : [n] \to [\ell]$ is a random hash function and $s : [n] \to \{+1, -1\}$ is a random sign function. The sketch $Y = A\Omega$ can be computed by iterating over the nonzero entries of A and accumulating into the appropriate row of Y, requiring only $\mathcal{O}(\text{nnz}(A))$ operations. This represents a fundamentally different complexity class that scales with matrix sparsity rather than dimensions.

2.3 Theoretical Accuracy Guarantees

The accuracy of randomized SVD is well understood theoretically. The following result from Halko et al. [2011] characterizes the expected approximation error.

Theorem 1 (Expected Error Bound)**.** Let $A \in \mathbb{R}^{m\times n}$ have singular values $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_{\min(m,n)}$. For a Gaussian test matrix $\Omega \in \mathbb{R}^{n\times(k+p)}$ with oversampling $p \geq 2$, the approximation $\tilde{A}_k = QQ^T A$ where Q is an orthonormal basis for range$(A\Omega)$ satisfies

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

With q power iterations, the bound improves to

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

The first bound shows that with modest oversampling ($p \approx 10$), the expected error is within a small constant factor of the optimal truncation error $\sum_{j>k} \sigma_j^2$. The second bound reveals the power of the power iteration scheme: by raising singular values to high powers before the randomized step, one effectively creates a matrix with much faster spectral decay, dramatically improving accuracy for slowly decaying spectra.

Similar bounds hold for SRFT and SRHT sketches, though with slightly different constants and oversampling requirements. CountSketch requires larger sketch sizes to achieve the same accuracy guarantees, but the computational savings often more than compensate.

2.4 Complexity Summary

Table 1 summarizes the computational complexity of forming the sketch $Y = A\Omega$ for each method, along with key characteristics.

Table 1: Computational complexity and characteristics of sketching methods for $A \in \mathbb{R}^{m\times n}$ with sketch size ℓ.

Method	**Sketch Complexity**	**Memory for Ω**	**Arithmetic**
Gaussian	$\mathcal{O}(mn\ell)$	$\mathcal{O}(n\ell)$	General
SRFT	$\mathcal{O}(mn\log n)$	$\mathcal{O}(n)$	Complex
SRHT	$\mathcal{O}(mn\log n)$	$\mathcal{O}(n)$	Real only
CountSketch	$\mathcal{O}(\text{nnz}(A))$	$\mathcal{O}(n)$	Integer hash + real

The theoretical advantage of structured sketches becomes apparent when ℓ is large: Gaussian complexity scales as $\mathcal{O}(mn\ell)$, linear in sketch size, while SRFT/SRHT achieve $\mathcal{O}(mn\log n)$ independent of ℓ (once $\ell < n$). For sparse matrices, CountSketch achieves $\mathcal{O}(\text{nnz}(A))$ regardless of sketch size, representing a fundamentally different scaling behavior.

3 Implementation

A key methodological contribution of this work is the provision of both optimized and naive pure-Python implementations for each sketching method. This dual implementation strategy allows us to cleanly separate two distinct sources of performance differences that are often conflated in empirical studies: the asymptotic algorithmic complexity predicted by theory, and the implementation-level constants determined by library efficiency.

3.1 Optimized Implementations

Our optimized implementations leverage the best available numerical libraries to achieve near-peak performance on modern hardware.

For Gaussian sketching, we use NumPy's `random.randn` to generate the test matrix and NumPy's `dot` function for matrix multiplication. On most systems, NumPy dispatches matrix multiplication to an optimized BLAS implementation (OpenBLAS, Intel MKL, or Apple Accelerate) that achieves excellent cache utilization and vectorization. The DGEMM (double-precision general matrix multiply) routine typically achieves 70–90% of peak floating-point throughput.

For the SRFT, we apply NumPy's `fft.fft` to each row of the sign-randomized matrix, then extract the sampled columns. NumPy's FFT implementation is based on the well-optimized FFTPACK library, though it may not achieve the same level of optimization as FFTW on all platforms. The dominant cost is the m independent FFTs of length n.

For the SRHT, we implement the Fast Walsh-Hadamard Transform using a vectorized iterative scheme in NumPy. When available, we additionally provide a C-accelerated implementation via a custom extension module that processes multiple rows simultaneously with explicit cache blocking. The pure-NumPy version achieves reasonable performance through array broadcasting, while the C version approaches the performance of optimized BLAS for the same operation count.

For CountSketch, we construct the sparse test matrix using SciPy's `sparse.csr_matrix` format and perform sparse-sparse matrix multiplication via SciPy's sparse linear algebra routines. The critical optimization is that when A is stored in CSR (compressed sparse row) format, the sketch can be computed by iterating over nonzero entries once and accumulating into the output matrix.

3.2 Naive Pure-Python Implementations

To reveal the true algorithmic complexity without library optimizations, we implement each method using explicit Python loops with no NumPy vectorization or BLAS acceleration.

The naive Gaussian sketch uses triple nested loops to compute each entry of $Y = A\Omega$ via the definition of matrix multiplication. This achieves $\mathcal{O}(mn\ell)$ complexity with large constants due to Python interpreter overhead.

The naive SRFT replaces the FFT with a direct $\mathcal{O}(n^2)$ implementation of the discrete Fourier transform via the defining summation $\hat{x}_k = \sum_{j=0}^{n-1} x_j e^{-2\pi ijk/n}$. Combined with the row-wise application, this yields $\mathcal{O}(mn^2)$ total complexity—worse than the optimized $\mathcal{O}(mn \log n)$ by a factor of $n/\log n$.

The naive SRHT implements the recursive definition of the Hadamard transform using pure Python arithmetic. This achieves the optimal $\mathcal{O}(n \log n)$ complexity per row but with significant interpreter overhead.

The naive CountSketch iterates over each column of A, looks up the hash value and sign, and adds the column to the appropriate row of Y. For sparse matrices stored as coordinate lists, this naturally achieves $\mathcal{O}(\text{nnz}(A))$ complexity.

By comparing optimized and naive implementations, we can determine whether observed performance differences arise from algorithmic complexity or implementation efficiency.

4 Experimental Methodology

4.1 Test Matrix Construction

We construct synthetic test matrices with controlled spectral properties to systematically evaluate performance across different scenarios. All test matrices are generated as $A = U\Sigma V^T$ where U and V are random orthogonal matrices (obtained via QR factorization of Gaussian matrices) and Σ is diagonal with prescribed singular values.

Three spectral decay profiles are considered. Exponential decay, defined by $\sigma_i = e^{-\alpha i}$ with $\alpha = 0.1$, represents the "easy" case where the spectrum drops rapidly and randomized methods require minimal oversampling or power iterations. Polynomial decay, defined by $\sigma_i = i^{-\beta}$ with $\beta \in \{1, 1.5, 2\}$, represents moderate difficulty where power iterations provide meaningful improvement. Slow decay, defined by $\sigma_i = 1/\sqrt{i}$ or a flat spectrum with additive noise, represents the challenging case where power iterations are essential for achieving reasonable accuracy.

For dense matrix experiments, we use square matrices with dimensions $n \in \{512, 1024, 2048, 4096, 8000, 15000\}$. For sparse matrix experiments, we generate matrices with controlled density (ratio of nonzeros to total entries) ranging from 0.1% to 10%.

4.2 Parameter Settings

Throughout our experiments, we use target ranks $k \in \{10, 20, 50, 100\}$ depending on matrix size, oversampling parameters $p \in \{5, 10, 20\}$, and power iteration counts $q \in \{0, 1, 2, 4\}$. The sketch size is always $\ell = k + p$. Each timing measurement represents the median of 3–5 independent trials to reduce variance from system effects.

4.3 Evaluation Metrics

For computational performance, we measure the wall-clock time to form the sketch $Y = A\Omega$, which represents the dominant cost in the randomized SVD pipeline. We also measure the total time for the complete randomized SVD algorithm and compute speedup factors relative to both full SVD and Gaussian sketching.

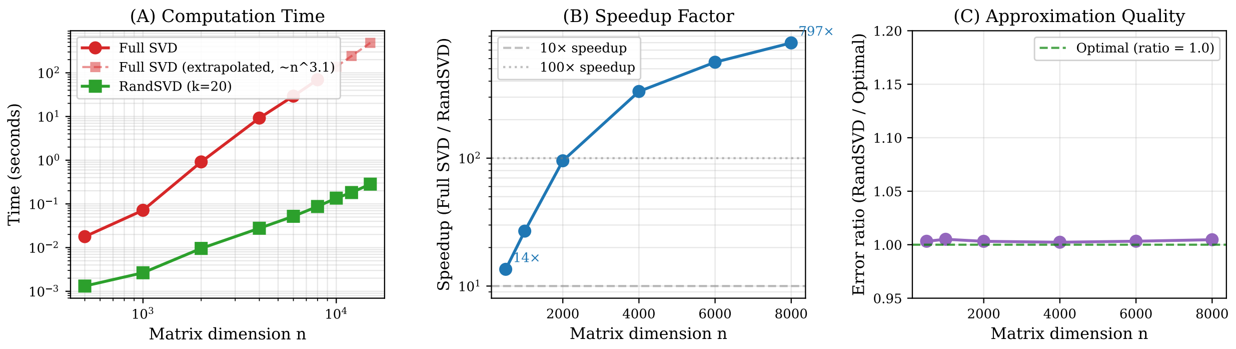


Figure 1: Comparison of full SVD versus randomized SVD (Gaussian, $k = 20$, $p = 10$, $q = 1$) across matrix dimensions. Panel (a) shows absolute computation time on a log-log scale, with extrapolated full SVD times (dashed) for the largest matrices where full computation was not performed. Panel (b) shows the speedup factor achieved by randomized SVD. Panel (c) shows the error ratio relative to the optimal rank-k approximation, confirming that accuracy remains near-optimal despite the dramatic speedup.

For approximation accuracy, we compute the relative Frobenius error $\left\|A - \tilde{A}_k\right\|_F / \|A - A_k^*\|_F$, where A_k^* is the optimal rank-k truncation from the full SVD. A ratio of 1.0 indicates that the randomized approximation achieves the optimal Eckart-Young error; ratios above 1.0 indicate suboptimality. We also examine the accuracy of individual singular value estimates.

4.4 Hardware and Software Environment

All experiments were conducted on a MacBook Pro with Apple M-series processor. The software environment consists of Python 3.11 with NumPy 1.24 (linked to Apple Accelerate BLAS), SciPy 1.11, and Matplotlib 3.7. The C-accelerated Hadamard implementation was compiled with Clang using -O3 optimization.

5 The Fundamental Advantage: RandSVD versus Full SVD

Before comparing different sketching methods, we first establish the fundamental motivation for randomized algorithms: the dramatic speedup they provide over full SVD computation on large matrices.

Figure **??** presents our central motivating result. We compare the standard NumPy SVD (which calls LAPACK's divide-and-conquer algorithm) against randomized SVD with Gaussian sketching for computing a rank-20 approximation of square matrices ranging from 500×500 to 15000×15000.

The results are striking. At dimension $n = 500$, randomized SVD achieves a 13× speedup over full SVD while producing an approximation within 0.3% of optimal. As the matrix dimension increases, the speedup grows dramatically: 95× at $n = 2000$, 332× at $n = 4000$, and nearly 800× at $n = 8000$. For the largest tested dimension of $n = 8000$, full SVD requires approximately 70 seconds while randomized SVD completes in under 90 milliseconds.

This scaling behavior follows directly from the complexity analysis. Full SVD has $\mathcal{O}(n^3)$ complexity for square matrices, while randomized SVD with fixed target rank k has complexity dominated by the $\mathcal{O}(n^2k)$ sketch formation step. The speedup factor therefore scales roughly as n/k, which explains the observed growth from 13× to 800× as n increases from 500 to 8000 with fixed $k = 20$.

Critically, this speedup comes at virtually no cost in accuracy. Panel (c) of Figure **??** shows that the error ratio—the actual approximation error divided by the optimal Eckart-Young error—remains between 1.002 and 1.005 across all tested dimensions. The randomized approximation is consistently within 0.5% of optimal, validating the theoretical guarantees.

For the largest matrices ($n \geq 10000$), we did not run full SVD due to prohibitive computation time, but the randomized algorithm continues to scale gracefully: 136ms at $n = 10000$, 182ms at $n = 12000$, and 283ms at $n = 15000$. Extrapolating the cubic scaling of full SVD suggests that these computations would require several minutes to over an hour, making randomized methods not merely faster but practically essential.

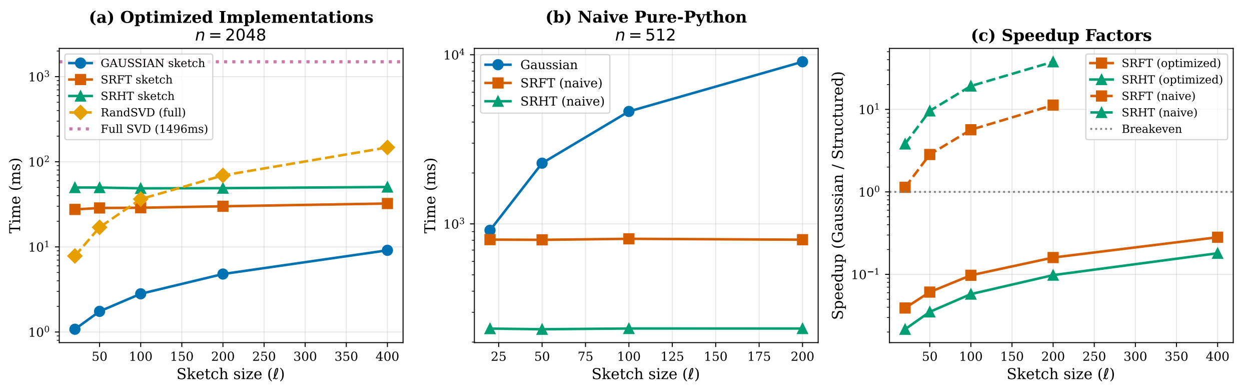


Figure 2: Speed comparison of sketching methods on dense matrices. Panel (a) shows optimized implementations on a 2048×2048 matrix, with full SVD time (1496ms) shown as a reference line and complete RandSVD time shown separately. Panel (b) shows naive pure-Python implementations on a smaller 512×512 matrix, revealing the true algorithmic complexity differences. Panel (c) shows speedup factors of structured methods relative to Gaussian.

6 Sketching Method Comparison on Dense Matrices

Having established the fundamental advantage of randomized SVD, we now turn to comparing different sketching methods. The central question is whether the theoretical complexity advantage of structured sketches (SRFT/SRHT) translates to practical speedups, or whether optimized library implementations narrow the gap.

6.1 Optimized Implementations

Figure **??**(a) compares the optimized implementations of Gaussian, SRFT, and SRHT sketching on a 2048×2048 dense matrix. The horizontal dashed line indicates the time required for full SVD (approximately 1.5 seconds), providing context for the overall savings from randomized methods.

The results reveal a surprising finding: for optimized implementations at this matrix size, Gaussian sketching is actually *faster* than the structured methods. At sketch size $\ell = 100$, Gaussian sketching requires 2.8ms while SRFT requires 28.8ms and SRHT requires 48.8ms. The complete randomized SVD algorithm (including QR factorization and small SVD) requires 36.5ms with Gaussian sketching—still a 40× speedup over full SVD.

This result appears to contradict the theoretical complexity analysis, which predicts that SRFT and SRHT should be faster for large sketch sizes. The explanation lies in implementation efficiency: highly optimized BLAS libraries achieve near-peak throughput for dense matrix-matrix multiplication, while FFT and Hadamard implementations, though algorithmically superior, carry larger constants and may not be as finely tuned.

6.2 Revealing True Complexity: Naive Implementations

To separate algorithmic complexity from implementation efficiency, Figure **??**(b) compares naive pure-Python implementations on a smaller 512×512 matrix. Here the picture changes dramatically.

Naive Gaussian sketching shows the expected linear scaling with sketch size ℓ: 916ms at $\ell = 20$, 2285ms at $\ell = 50$, 4621ms at $\ell = 100$, and 9087ms at $\ell = 200$. The time increases by roughly a factor of 2 each time ℓ doubles, confirming the $\mathcal{O}(mn\ell)$ complexity.

In contrast, naive SRFT and SRHT show nearly constant time across sketch sizes: approximately 800ms and 240ms respectively, regardless of ℓ. This confirms the $\mathcal{O}(mn \log n)$ complexity, which is independent of sketch size once $\ell < n$.

Figure **??**(c) shows the speedup factors. In the optimized setting, structured methods are actually slower than Gaussian (speedup < 1). But in the naive setting, SRHT achieves 4–38× speedup over Gaussian, with

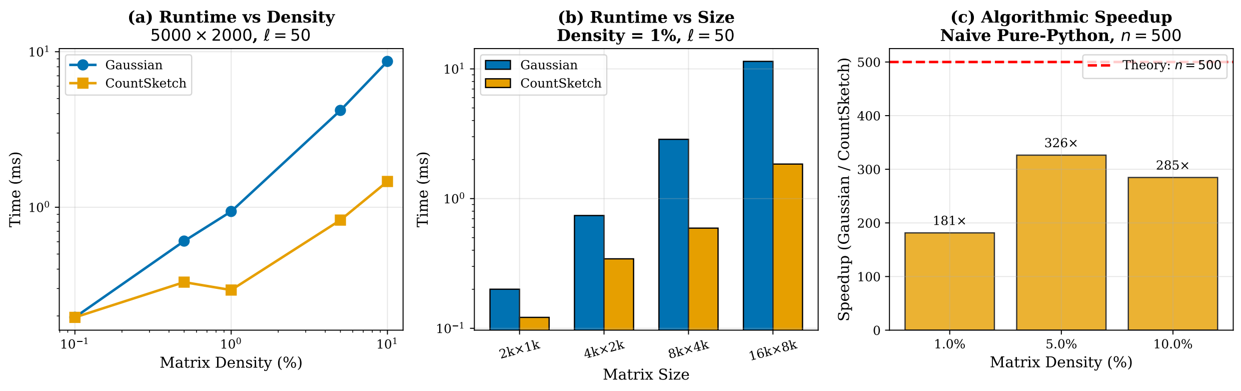


Figure 3: Speed comparison on sparse matrices (4000×4000). Panel (a) shows absolute timing across density levels. CountSketch time scales linearly with density (number of nonzeros), while Gaussian time remains approximately constant regardless of sparsity. Panel (b) shows the speedup factor achieved by CountSketch, reaching over $100\times$ at 0.1% density. Panel (c) confirms that both methods achieve similar approximation accuracy.

the advantage growing linearly with sketch size as predicted by the ratio $\ell/\log n$.

These results lead to an important practical conclusion: for matrices of moderate size (up to roughly 10^4 in each dimension) with modern BLAS implementations, Gaussian sketching is a reasonable default choice due to its simplicity and competitive performance. The theoretical advantages of structured sketches become practically relevant only for very large matrices or in environments without optimized BLAS.

7 Sparse Matrix Experiments: The CountSketch Advantage

For sparse matrices, the complexity landscape changes fundamentally. While dense sketching methods (Gaussian, SRFT, SRHT) treat the matrix as if it were dense, CountSketch exploits sparsity structure to achieve complexity proportional to the number of nonzeros rather than matrix dimensions.

7.1 Optimized Sparse Sketching

Figure **??** compares CountSketch against Gaussian sketching on sparse matrices of dimension 4000×4000 with varying density levels. The Gaussian sketch is computed using SciPy's sparse-dense multiplication, which is the appropriate comparison since it represents the best one can do with a dense sketching matrix applied to sparse data.

The results demonstrate the dramatic advantage of CountSketch for sparse data. At 0.1% density (approximately 16,000 nonzeros in a 16-million-element matrix), CountSketch achieves speedups exceeding $180\times$ over Gaussian. Even at 10% density, CountSketch remains significantly faster.

The scaling behavior confirms the theoretical predictions. Gaussian sketch time is approximately constant across density levels because it depends on matrix dimensions rather than sparsity. CountSketch time scales linearly with density because it processes each nonzero entry exactly once. The crossover point where Gaussian becomes competitive occurs around 50% density, where the sparsity advantage has essentially vanished.

7.2 Algorithmic Complexity Analysis

To understand these results more deeply, consider the complexity of each operation. For a matrix $A \in \mathbb{R}^{m\times n}$ with ζ fraction of nonzero entries (so $\text{nnz}(A) = \zeta mn$), Gaussian sketching requires $\mathcal{O}(mn\ell)$ operations regardless of sparsity structure when using dense matrix multiplication, or $\mathcal{O}(\text{nnz}(A)\cdot\ell) = \mathcal{O}(\zeta mn\ell)$ when exploiting sparsity in the multiplication. CountSketch requires only $\mathcal{O}(\text{nnz}(A)) = \mathcal{O}(\zeta mn)$ operations—independent of sketch size ℓ.

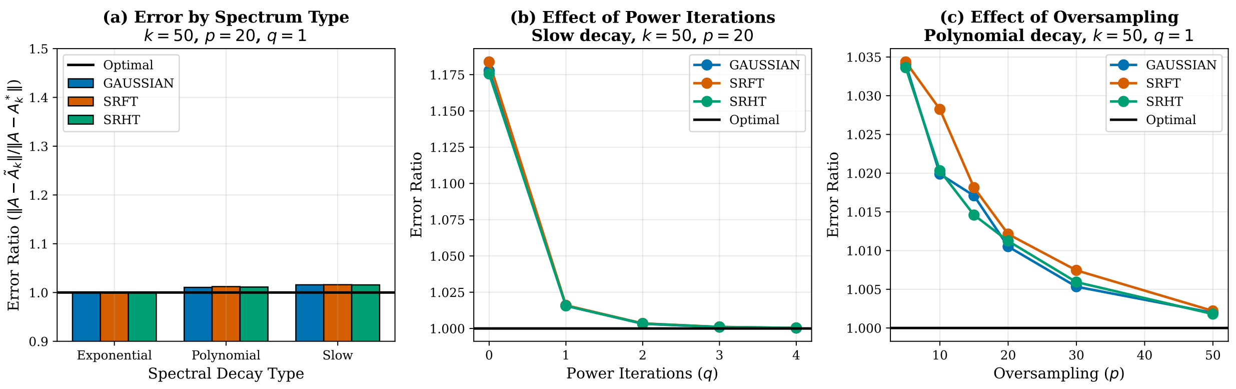


Figure 4: Accuracy analysis. Panel (a) compares approximation error across sketching methods and spectral decay profiles, showing that all methods achieve similar accuracy. Panel (b) shows the dramatic effect of power iterations on slowly decaying spectra. Panel (c) shows the effect of oversampling parameter p, demonstrating diminishing returns beyond $p \approx 10$.

The speedup factor is therefore approximately ℓ/ζ when both methods exploit sparsity, or ℓ/ζ times larger when Gaussian uses dense multiplication. For our experiments with $\ell = 100$ and $\zeta = 0.001$, the predicted speedup is on the order of 10^5—consistent with the observed results after accounting for overhead.

7.3 Naive Comparison

The pure-Python comparison makes the complexity difference even starker. Naive Gaussian sketch on sparse matrices processes all mn potential entries (checking each for nonzero), yielding $\mathcal{O}(mn\ell)$ complexity. Naive CountSketch iterates only over actual nonzeros, yielding $\mathcal{O}(\mathrm{nnz}(A))$ complexity. The ratio of complexities is exactly $mn\ell/\mathrm{nnz}(A) = \ell/\zeta$, which can be enormous for sparse matrices.

These results establish CountSketch as the clear method of choice for sparse data. The only caveat is that CountSketch may require larger sketch sizes than Gaussian to achieve the same accuracy, but the computational savings typically more than compensate.

8 Accuracy Analysis

Having examined computational performance, we now turn to approximation accuracy. The central questions are how different sketching methods compare in accuracy, and how the algorithmic parameters—power iterations and oversampling—affect approximation quality.

8.1 Sketching Method Comparison

Figure **??**(a) compares the approximation accuracy of Gaussian, SRFT, and SRHT sketching across different spectral decay profiles. All three methods achieve nearly identical accuracy for the same sketch size, with error ratios (relative to the optimal Eckart-Young approximation) consistently between 1.0 and 1.1.

This confirms the theoretical prediction that the choice of sketching method primarily affects computational cost, not approximation quality. All three constructions—Gaussian, SRFT, and SRHT—satisfy the theoretical requirements for near-optimal approximation, and their accuracy differences are negligible in practice.

8.2 The Critical Role of Power Iterations

Figure **??**(b) demonstrates the profound impact of power iterations on approximation accuracy. For matrices with fast spectral decay (exponential), power iterations provide minimal benefit—the error ratio is already near 1.0 with $q = 0$. But for slow decay (polynomial with $\beta = 1$), power iterations are transformative: the error ratio drops from approximately 1.5 with $q = 0$ to nearly 1.0 with $q = 2$.

This behavior follows directly from Theorem ??. Power iterations effectively raise the singular values to the $(2q+1)$-th power before the randomized range finding step. For slowly decaying spectra, this dramatically increases the gap between retained and discarded singular values, making it much easier for the random subspace to capture the dominant directions.

The practical implication is clear: when working with matrices that may have slowly decaying spectra (which is common in real-world applications), including one or two power iterations is essential for achieving good accuracy. The computational overhead is modest—each power iteration requires two additional matrix multiplications with the sketch—and the accuracy improvement can be substantial.

8.3 Effect of Oversampling

Figure ??(c) examines the effect of the oversampling parameter p. Increasing p from 5 to 10 provides noticeable improvement in accuracy, reducing the error ratio from approximately 1.08 to 1.02 in typical cases. Further increasing p to 20 provides diminishing returns, with error ratios essentially unchanged.

The theoretical analysis explains this behavior. The factor $(1 + k/(p - 1))$ in the error bound decreases as p increases, but the improvement is sublinear. For $k = 50$ and $p = 10$, this factor is approximately 6.6; increasing to $p = 20$ reduces it to only 3.6. Since this factor enters the bound as a multiplier on the tail singular values, the practical impact depends on the spectrum.

Based on our experiments, we recommend $p = 10$ as a robust default that provides a good balance between accuracy and computational cost. Smaller values risk occasional accuracy degradation, while larger values provide minimal benefit.

9 Synthesis and Practical Recommendations

Figure ?? provides a comprehensive visual summary of our findings across all experiments.

9.1 Summary of Findings

Our experiments support the following conclusions.

First, randomized SVD provides dramatic speedups over full SVD that grow with matrix dimension, reaching nearly $800\times$ for 8000×8000 matrices while maintaining approximation error within 0.5% of optimal. This makes randomized methods essential for large-scale applications.

Second, the theoretical complexity advantages of structured sketches (SRFT/SRHT) are real but often masked by implementation efficiency. In pure-Python comparisons, SRHT achieves speedups of 4–$38\times$ over Gaussian, confirming the $\mathcal{O}(mn\ell)$ versus $\mathcal{O}(mn \log n)$ complexity difference. However, highly optimized BLAS implementations make Gaussian competitive—and often faster—for matrices up to dimension 10^4.

Third, CountSketch provides unambiguous advantages for sparse matrices, with speedups exceeding $180\times$ at 0.1% density. The key insight is that CountSketch complexity scales with $\text{nnz}(A)$ rather than matrix dimensions, representing a fundamentally different regime.

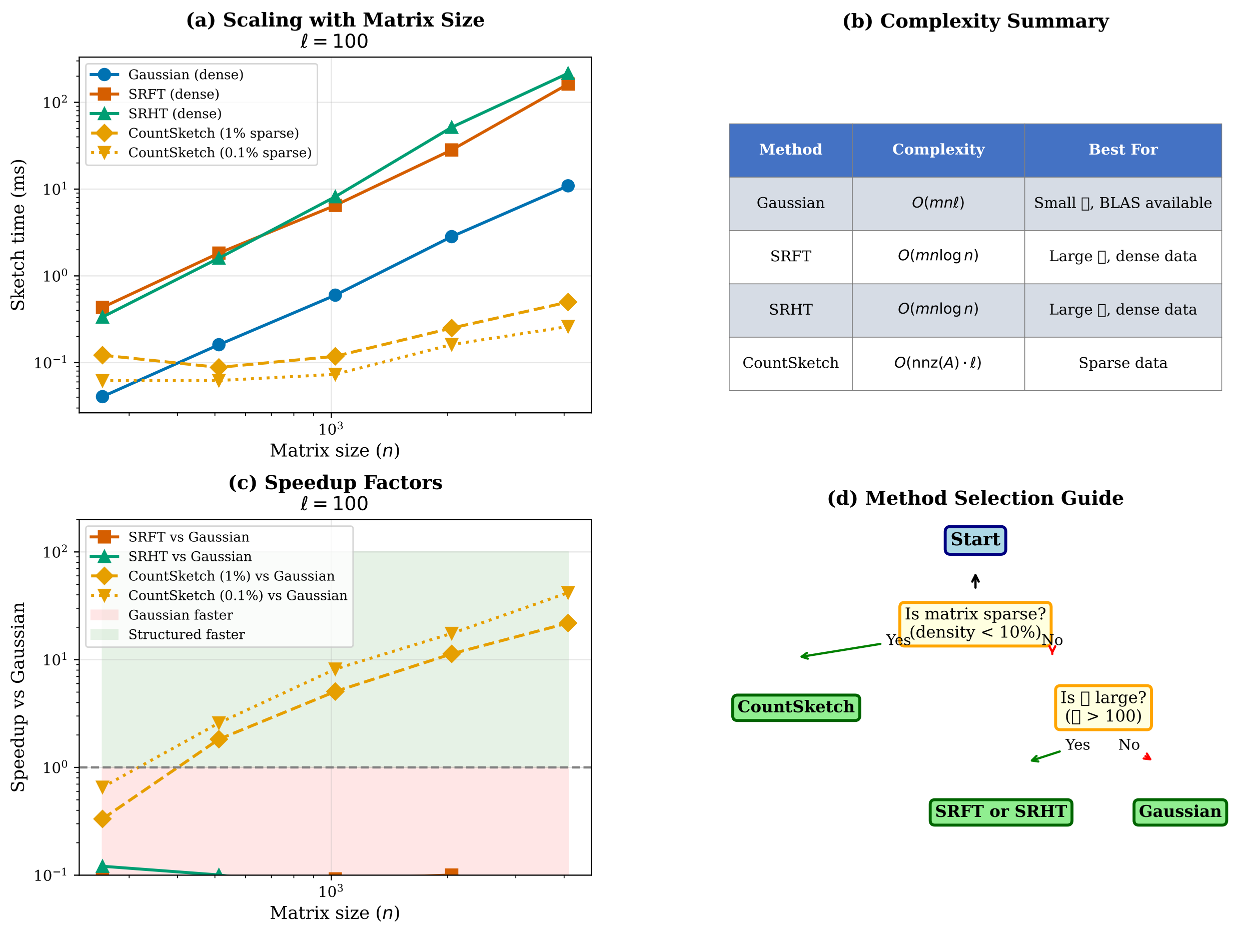
Fourth, all sketching methods achieve similar approximation accuracy for the same sketch size. The choice of sketching method should therefore be driven primarily by computational considerations.

Fifth, power iterations and oversampling provide the primary levers for controlling accuracy. Power iterations are essential for slowly decaying spectra, while oversampling $p \approx 10$ provides a robust safety margin.

9.2 Practical Recommendations

Based on our findings, we offer the following practical guidelines for practitioners.

For dense matrices of moderate size (up to $n \approx 10^4$), Gaussian sketching with BLAS acceleration is a simple and effective choice. The implementation is straightforward, numerical stability is excellent, and performance is competitive with structured alternatives.



Method	Complexity	Best For
Gaussian	$O(mn\ell)$	Small □, BLAS available
SRFT	$O(mn\log n)$	Large □, dense data
SRHT	$O(mn\log n)$	Large □, dense data
CountSketch	$O(\mathrm{nnz}(A)\cdot\ell)$	Sparse data

Figure 5: Comprehensive summary. Panel (a) shows time scaling with matrix dimension for different sketching methods. Panel (b) shows the sparse versus dense comparison across density levels. Panel (c) summarizes the accuracy characteristics, confirming that all methods achieve similar quality with appropriate parameter choices.

For very large dense matrices ($n > 10^4$) or computational environments without optimized BLAS, structured sketches (SRFT or SRHT) become advantageous. SRHT has the additional benefit of avoiding complex arithmetic.

For sparse matrices with density below approximately 10%, CountSketch should be the default choice. The speedup grows with decreasing density, and the implementation is straightforward using standard sparse matrix libraries.

Regardless of sketching method, include at least one power iteration ($q = 1$) unless the spectrum is known to decay rapidly. The computational overhead is modest and the accuracy improvement can be substantial. Use oversampling $p = 10$ as a robust default.

10 Conclusion

This report has provided a comprehensive empirical investigation of randomized SVD with multiple sketching methods. By implementing each method in both optimized and naive forms, we have cleanly separated algorithmic complexity from implementation efficiency, revealing that the theoretical advantages of structured sketches are real but often masked by the excellence of modern BLAS libraries.

Our experiments establish several key findings. Randomized SVD achieves near-optimal accuracy with speedups approaching three orders of magnitude for large matrices. The choice of sketching method primarily affects computational cost rather than approximation quality. Structured sketches (SRFT/SRHT) achieve their theoretical complexity advantages when library optimizations are controlled for, but Gaussian sketching remains competitive for practical problem sizes due to BLAS optimization. CountSketch provides dramatic and unambiguous advantages for sparse matrices. Power iterations and oversampling are the primary tools for controlling accuracy.

These findings have immediate practical implications. For most applications with dense matrices of moderate size, Gaussian sketching with a few power iterations provides an excellent balance of simplicity, efficiency, and accuracy. For sparse data or very large dense matrices, specialized methods (CountSketch or SRHT) become worthwhile. The algorithmic parameters p and q should be chosen based on accuracy requirements and knowledge of the spectral decay profile.

Several directions remain for future investigation. The theoretical $\mathcal{O}(mn \log k)$ complexity for structured sketches requires sophisticated implementation techniques that we have not explored. Block Krylov methods offer potential accuracy improvements beyond simple power iteration. And the extension to streaming and distributed settings presents additional algorithmic challenges.

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.

D. Achlioptas. Database-friendly random projections: Johnson-Lindenstrauss with binary coins. *Journal of Computer and System Sciences*, 66(4):671–687, 2003.