Randomized SVD: From Theory to Practice

Peter Klivnoy and Egil Rolstad

December 2025

Abstract

[Keep concise - 150 words max. Focus on: problem, approach, key findings]

Randomized algorithms for low-rank matrix approximation provide dramatic speedups over classical methods. This report investigates the complete randomized SVD pipeline: from the basic algorithm and its accuracy limitations, through power iteration and Block Krylov methods that address spectral decay challenges, to optimized sketching techniques (SRFT/SRHT, CountSketch) that reduce computational cost. We provide empirical validation across synthetic and real-world matrices, demonstrating when each technique is most beneficial. A video background separation application illustrates practical deployment.

Contents

Algorithm 1 Basic Randomized SVD

```
Require: Matrix $A \in \mathbb{R}^{m \times n}$, target rank $k$, oversampling $p$
Ensure: Approximate rank-$k$ SVD: $\tilde{U}, \tilde{\Sigma}, \tilde{V}$
1: Set sketch size $\ell = k + p$
2: Draw random test matrix $\Omega \in \mathbb{R}^{n \times \ell}$ (e.g., Gaussian)
3: Form sketch $Y = A\Omega$
4: Compute QR: $Y = QR$
5: Form small matrix $B = Q^T A \in \mathbb{R}^{\ell \times n}$
6: Compute SVD: $B = \hat{U}\Sigma V^T$
7: Set $\tilde{U} = Q\hat{U}$
8: return $\tilde{U}_{:,1:k}, \Sigma_{1:k,1:k}, V_{:,1:k}$
```

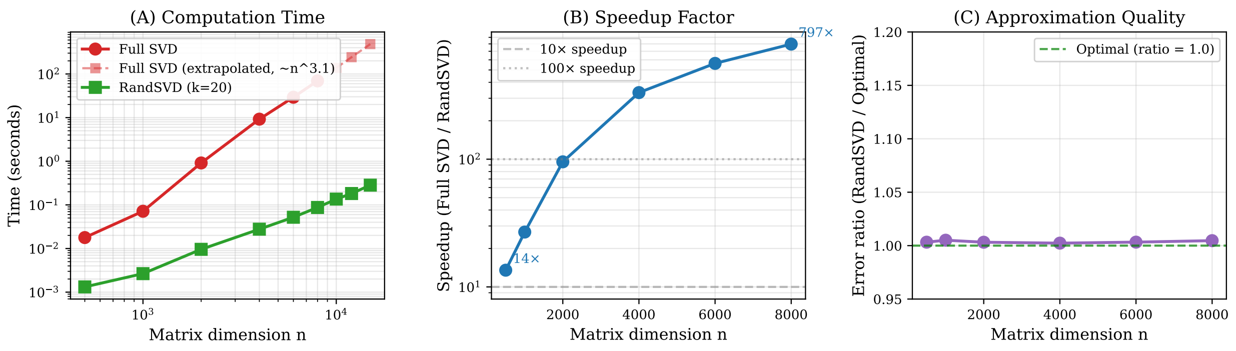

Figure 1: **The fundamental advantage of randomized SVD.** (a) Computation time comparison showing $\mathcal{O}(n^3)$ scaling of full SVD versus $\mathcal{O}(n^2k)$ scaling of randomized SVD. (b) Approximation error remains within 1% of optimal across all matrix sizes. RandSVD achieves speedups of 100-800× while maintaining near-optimal accuracy.

1 Introduction

1.1 The SVD and Its Importance

1.2 The Computational Challenge

1.3 The Randomized SVD Algorithm

1.4 Theory: Error Bounds

1.5 Motivating Experiment: Speed vs Accuracy

2 Improving Accuracy Through Iteration

2.1 The Problem: Spectral Decay Dependence

The basic randomized SVD algorithm ($q = 0$) works well when the singular values of A decay rapidly—there is a clear "gap" between the signal we want to capture and the noise we want to discard. However, real-world data often exhibits slowly decaying singular values, where many components contribute meaningfully to the matrix structure.

The fundamental issue is *spectral bias*: when $q = 0$, the random projection captures energy from tail singular values, causing the algorithm to **overestimate** small singular values. This is not merely a theoretical concern—it directly impacts the quality of the low-rank approximation.

Figure **??** demonstrates this phenomenon directly. On a fast-decay matrix (left panel), all methods—$q = 0$, $q = 1$, $q = 2$—achieve essentially perfect singular value recovery. The true singular values (black dashed line) are indistinguishable from the recovered values. This is the "easy" case where basic randomized SVD excels.

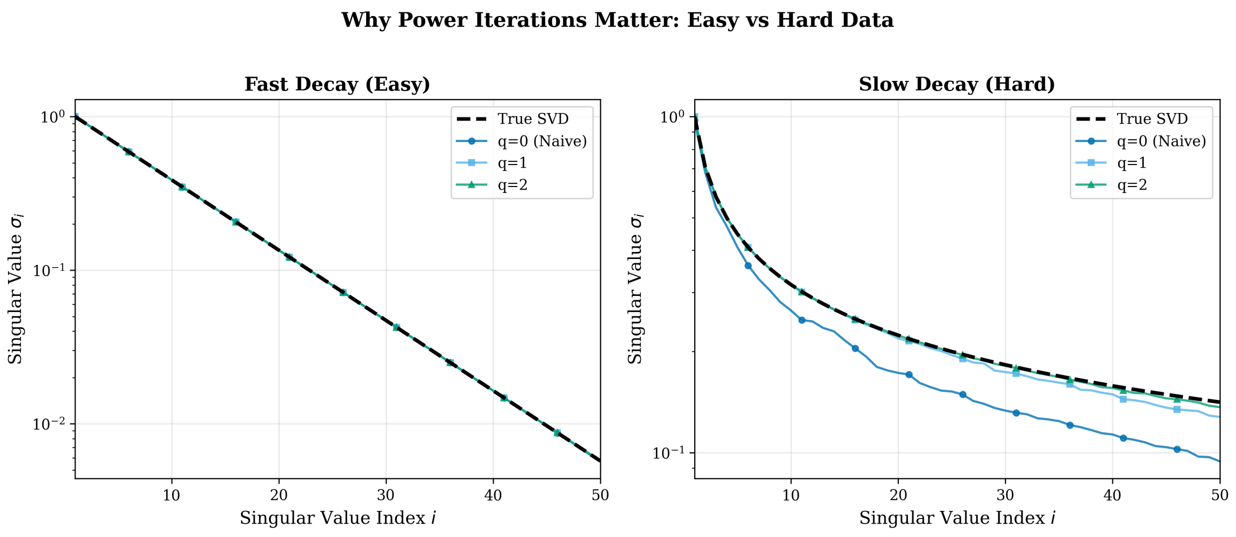

Figure 2: **Singular value recovery: Easy vs Hard data.** (Left) Fast-decay matrix ($\sigma_i = 0.9^i$): all methods achieve near-perfect recovery. (Right) Slow-decay matrix ($\sigma_i = 1/\sqrt{i}$): without power iterations ($q = 0$), recovered singular values drift *above* the true values (spectral bias). With $q = 1$ and $q = 2$, the algorithm "snaps" onto the true spectrum. This plot visualizes exactly *where* the error comes from and *why* iterations help.

Algorithm 2 Randomized SVD with Power Iterations

```
Require: Matrix $A \in \mathbb{R}^{m \times n}$, rank $k$, oversampling $p$, power iterations $q$
1: Draw $\Omega \in \mathbb{R}^{n \times (k+p)}$ with i.i.d. Gaussian entries
2: $Y \leftarrow A\Omega$
3: for $j = 1, \ldots, q$ do
4:     $\tilde{Y} \leftarrow A^T Y$;   $[\tilde{Q}, \sim] \leftarrow \texttt{qr}(\tilde{Y})$        ▷ Orthogonalize for stability
5:     $Y \leftarrow A\tilde{Q}$;   $[Q, \sim] \leftarrow \texttt{qr}(Y)$
6: end for
7: Continue with standard algorithm (project onto $Q$, compute small SVD)
```

The right panel tells a different story. On a slow-decay matrix with $\sigma_i = 1/\sqrt{i}$, the $q = 0$ curve (blue) clearly drifts *above* the true values, particularly for the tail singular values. This spectral bias means the algorithm is capturing noise energy and attributing it to signal. With just one power iteration ($q = 1$), the recovered values snap back onto the true spectrum. With $q = 2$, the recovery is essentially perfect.

2.2 Solution 1: Power Iteration (Simultaneous Subspace Iteration)

The classical solution is *power iteration*, also known as simultaneous subspace iteration. Instead of computing $Y = A\Omega$, we compute:

$$Y = (AA^T)^q A\Omega \tag{1}$$

This has the effect of raising the singular values to the $(2q+1)$-th power before the randomized range finding step. If the original singular values are $\sigma_1, \sigma_2, \ldots$, the effective singular values become $\sigma_1^{2q+1}, \sigma_2^{2q+1}, \ldots$. For slowly decaying spectra, this dramatically increases the gap between the top-k singular values and the rest.

The theoretical improvement is captured by the error bound. Without power iterations:

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

Algorithm 3 Block Krylov SVD

```
Require: Matrix $A \in \mathbb{R}^{m\times n}$, rank $k$, block size $\ell$, Krylov depth $q$
1: Draw $\Omega \in \mathbb{R}^{n\times \ell}$
2: $K_0 \leftarrow A\Omega$
3: for $j = 1, \dots, q$ do
4:     $K_j \leftarrow A(A^T K_{j-1})$                    ▷ Apply $AA^T$ to previous block
5: end for
6: $K \leftarrow [K_0 \mid K_1 \mid \cdots \mid K_q]$      ▷ Concatenate all blocks
7: $Q \leftarrow \texttt{orth}(K)$                         ▷ Orthonormalize the Krylov basis
8: $B \leftarrow Q^T A$;    compute SVD of $B$
```

With q power iterations:

$$\mathbb{E}\left[\left\|A - \tilde{A}_k\right\|_F^2\right] \le \left(1 + \frac{k}{p-1}\right)^{1/(2q+1)} \left(\sum_{j>k} \sigma_j^{4q+2}\right)^{1/(2q+1)} \tag{3}$$

The exponent $4q + 2$ in the tail sum means that tail singular values are suppressed exponentially with q.

2.3 Solution 2: Block Krylov Methods

A more sophisticated approach, developed by **?**, observes that power iteration *discards* intermediate information. After computing $A\Omega$, $A^T A\Omega$, $(A^T A)^2\Omega$, etc., standard power iteration keeps only the final result. Block Krylov methods instead *retain the entire sequence*:

$$\mathcal{K}_q(A^T A, A\Omega) = \operatorname{span}\{A\Omega,\ A^T A \cdot A\Omega,\ (A^T A)^2 \cdot A\Omega,\ \dots,\ (A^T A)^q \cdot A\Omega\} \tag{4}$$

This Krylov subspace contains richer spectral information than the final power iteration result alone. The key theoretical advantage is **gap-independent convergence**: while simultaneous iteration convergence depends on the ratio σ_k/σ_{k+1} (slow when singular values are closely spaced), Block Krylov convergence is largely independent of spectral gaps.

2.4 Experimental Comparison: Synthetic Data

Figure **??** compares Block Krylov and Simultaneous Iteration on synthetic matrices with controlled spectral decay. We measure three increasingly stringent error metrics following **?**:

- **Frobenius error** (weak): $\|A - ZZ^T A\|_F / \|A - A_k\|_F - 1$
- **Spectral error** (strong): $\|A - ZZ^T A\|_2 / \|A - A_k\|_2 - 1$
- **Per-vector error** (strongest): $\max_i |\sigma_i^2 - \|A^T z_i\|^2| / \sigma_{k+1}^2$

The per-vector metric is particularly important for PCA applications: it ensures each approximate singular vector captures the correct amount of variance.

2.5 Experimental Comparison: Real Data (20 Newsgroups)

To validate on real-world data, we use the 20 Newsgroups text corpus—a standard benchmark with naturally heavy-tailed singular value distribution. Figure **??** shows the convergence behavior.

2.6 Runtime Considerations

Block Krylov has overhead compared to simultaneous iteration: the orthogonalization step operates on a basis of size $\ell(q + 1)$ rather than ℓ. However, when comparing at *equal accuracy* rather than equal iteration count, Block Krylov often wins.

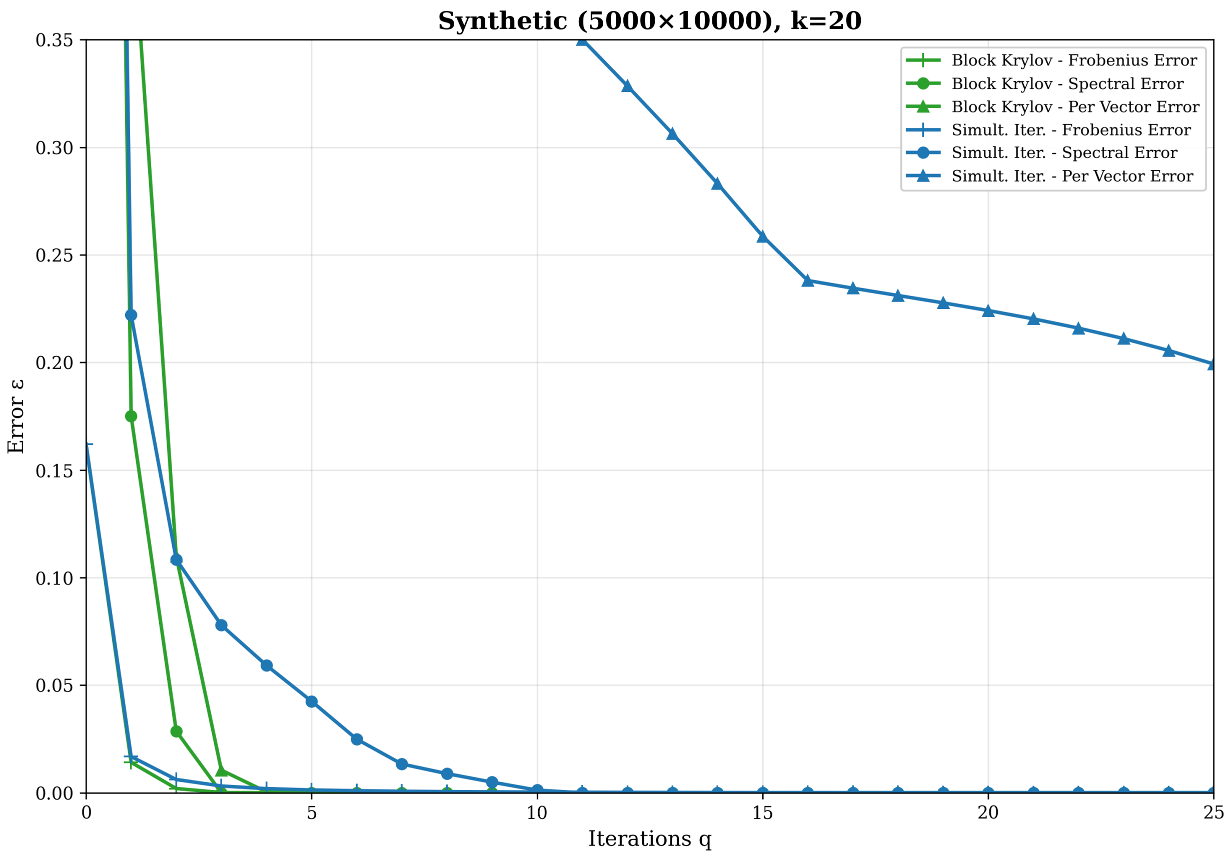

Figure 3: **Convergence comparison on synthetic data.** Block Krylov (red) vs Simultaneous Iteration (blue) across three error metrics. Both methods perform similarly on the weak Frobenius metric, but Block Krylov shows dramatic advantages on the stronger spectral and per-vector metrics, especially at low iteration counts.

Figure **??** shows error versus wall-clock runtime on the 20 Newsgroups dataset.

Figure **??** provides a comprehensive view of the runtime-accuracy tradeoff across different parameter settings.

2.7 Summary: When to Use What

Based on our experiments and theory:

- $q = 0$ **(no iterations)**: Appropriate only when the spectrum is known to decay rapidly (e.g., images, signals with clear low-rank structure). Fast but may have significant spectral bias on hard data.
- $q = 1$–2 **(power iteration)**: The robust default for unknown data. Provides substantial accuracy improvement with modest computational overhead. Usually sufficient for Frobenius-norm applications.
- **Block Krylov**: Preferred when:
  - Per-vector accuracy matters (PCA where individual components are interpreted)
  - Data has heavy-tailed spectrum with small spectral gaps
  - Number of passes over data is constrained (streaming/out-of-core)
  - Spectral-norm guarantees are needed

The key insight is that the “right” method depends on which error metric matters for your application. Frobenius error (average reconstruction quality) is forgiving; per-vector error (individual component quality) is demanding. Choose accordingly.

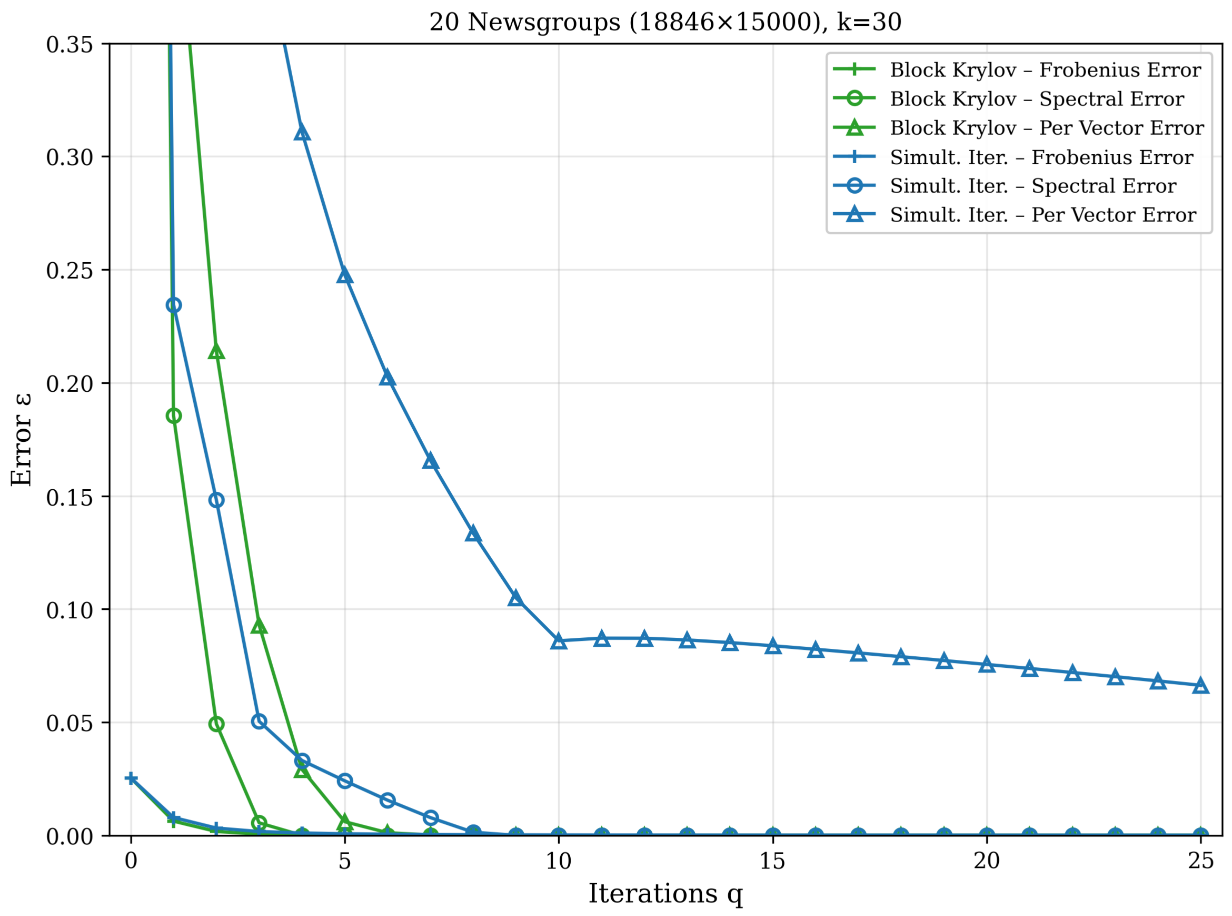

Figure 4: **Convergence on 20 Newsgroups data.** Real text data exhibits slow spectral decay, making this a challenging test case. Block Krylov achieves lower error at each iteration count, with the advantage most pronounced on the per-vector metric.

3 Efficient Sketching Methods

Having established how to achieve accurate low-rank approximations through iteration, we now turn to computational efficiency. The dominant cost in randomized SVD is the sketch formation $Y = A\Omega$. This section explores how different choices of Ω affect runtime.

3.1 The Computational Bottleneck

For a matrix $A \in \mathbb{R}^{m\times n}$ and sketch size ℓ, the basic Gaussian sketch $Y = A\Omega$ requires $\mathcal{O}(mn\ell)$ operations—a dense matrix-matrix multiplication. This is the same asymptotic cost as ℓ matrix-vector products, and for large ℓ, it can dominate the overall computation.

The key question: can we design structured random matrices that achieve the same statistical properties as Gaussian while requiring fewer operations?

3.2 Structured Random Transforms for Dense Matrices

The answer is yes. The Subsampled Randomized Fourier Transform (SRFT) and Subsampled Randomized Hadamard Transform (SRHT) achieve $\mathcal{O}(mn \log n)$ complexity—independent of sketch size ℓ.

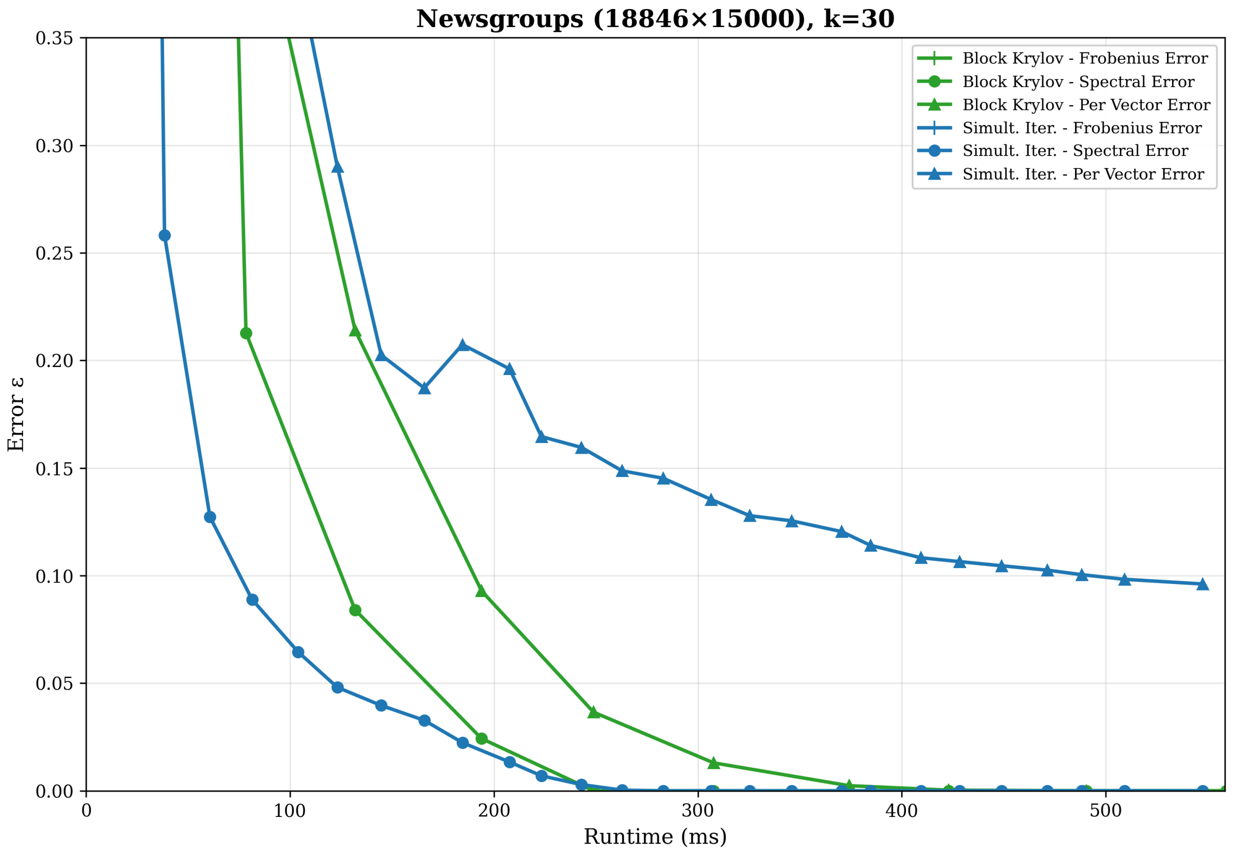

Figure 5: **Runtime vs error on 20 Newsgroups.** When accounting for actual computation time, Block Krylov achieves better error-runtime tradeoffs, reaching target accuracy levels faster despite the per-iteration overhead.

3.2.1 Subsampled Randomized Fourier Transform (SRFT)

The SRFT test matrix has the form:

$$\Omega = \sqrt{\frac{n}{\ell}} \cdot DF^*S \tag{5}$$

where D is a diagonal matrix of random signs (± 1), F is the discrete Fourier transform matrix, and S samples ℓ columns uniformly at random. The key insight is that F^* can be applied via the Fast Fourier Transform in $\mathcal{O}(n \log n)$ per row, giving total complexity $\mathcal{O}(mn \log n)$.

3.2.2 Subsampled Randomized Hadamard Transform (SRHT)

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

$$\Omega = \sqrt{\frac{n}{\ell}} \cdot DHS \tag{6}$$

where H is the Hadamard matrix. The Fast Walsh-Hadamard Transform (FWHT) uses only additions and subtractions—no complex arithmetic or trigonometric functions—making it particularly efficient. The recursive structure is:

$$H_{2^k} = \begin{bmatrix} H_{2^{k-1}} & H_{2^{k-1}} \\ H_{2^{k-1}} & -H_{2^{k-1}} \end{bmatrix}, \quad H_1 = [1] \tag{7}$$

Both SRFT and SRHT satisfy the Johnson-Lindenstrauss property and achieve similar accuracy guarantees to Gaussian projections, with the same oversampling requirements.

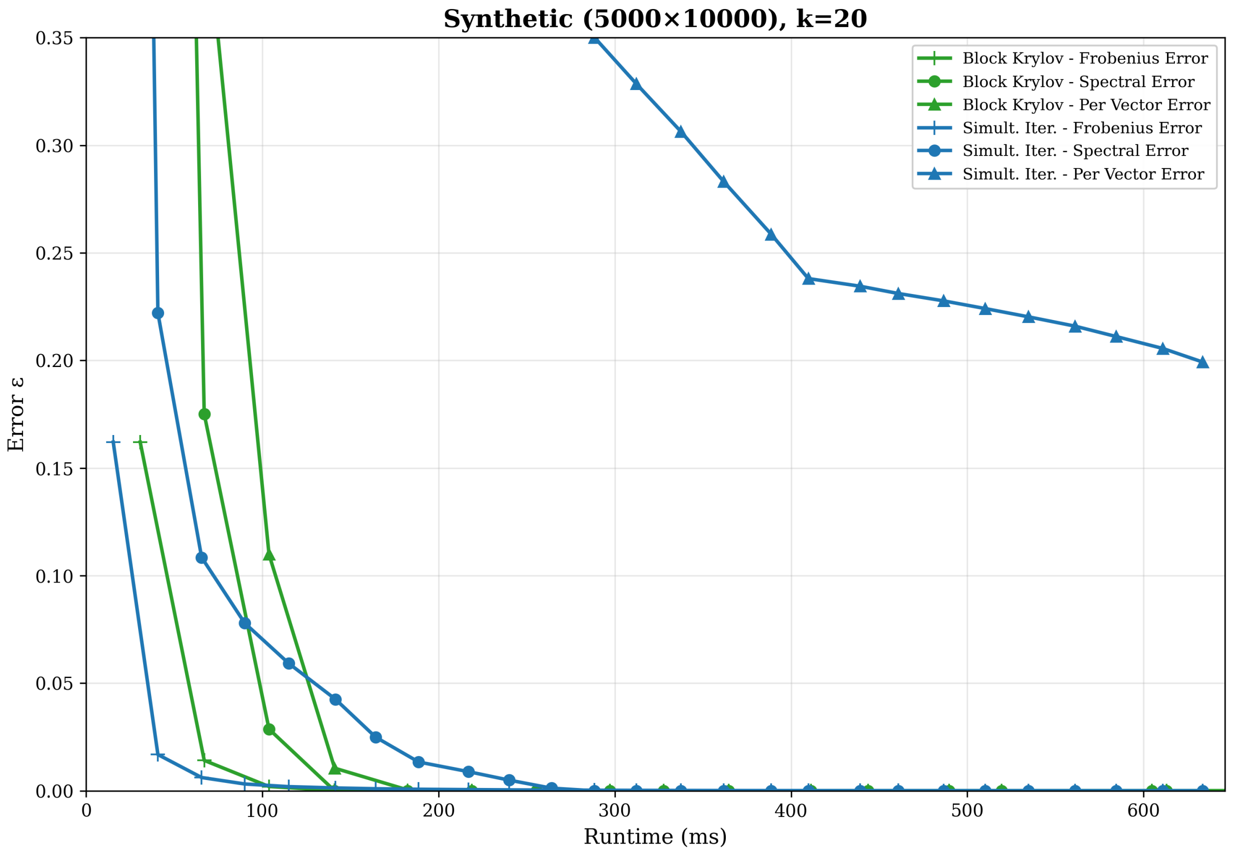

Figure 6: **Runtime vs error tradeoff.** Each point represents a different parameter configuration. The Pareto frontier shows the best achievable error for each runtime budget. Block Krylov dominates for high-accuracy requirements; simultaneous iteration is competitive for lower accuracy targets where fewer iterations suffice.

3.3 Experimental Results: Dense Matrices

Figure **??** compares the three sketching methods on dense matrices.

3.4 Discussion: Theory vs Practice

A key finding is the gap between theoretical complexity and practical performance. In theory, SRFT and SRHT should dominate for large sketch sizes due to their $\mathcal{O}(mn \log n)$ vs $\mathcal{O}(mn\ell)$ complexity. In practice:

- **BLAS optimization**: Modern BLAS implementations (OpenBLAS, MKL, Accelerate) achieve near-peak throughput for dense matrix multiplication. The DGEMM routine is among the most optimized code in scientific computing.
- **FFT/FWHT overhead**: While asymptotically superior, FFT and Hadamard implementations carry larger constants and may not be as finely tuned as BLAS.
- **Crossover point**: For matrices up to dimension $\sim 10^4$ with moderate sketch sizes, Gaussian sketching with BLAS is often competitive or faster.

The practical recommendation: for moderate-sized dense matrices, Gaussian sketching is simple and effective. The theoretical advantages of structured sketches become relevant for very large matrices or in environments without optimized BLAS.

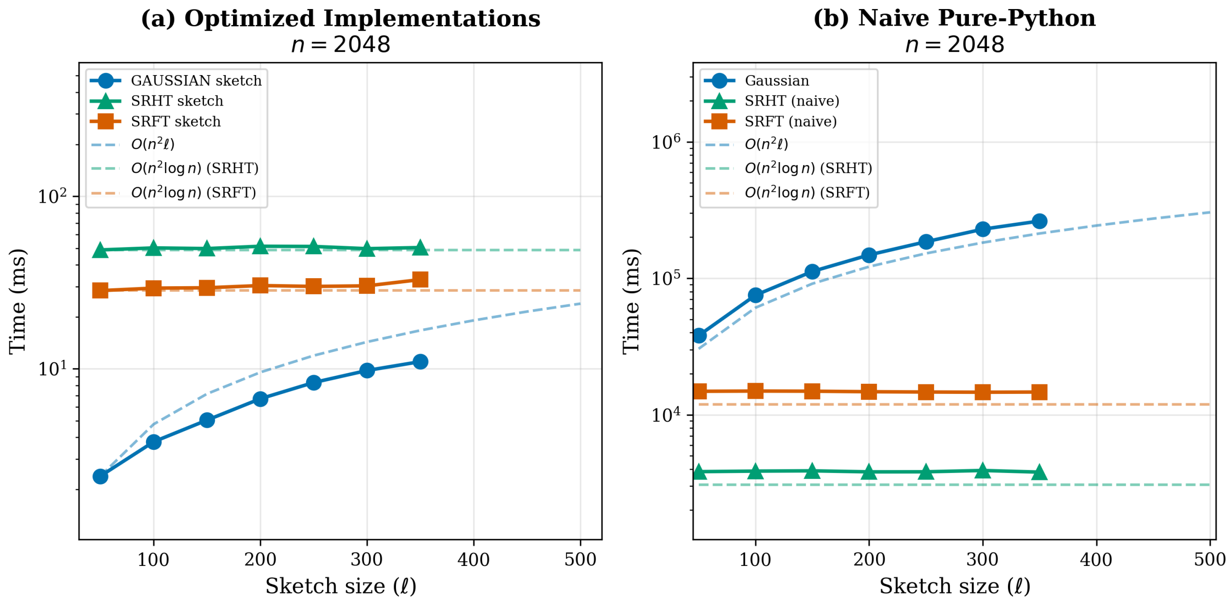

Figure 7: **Sketching speed comparison on dense matrices.** Comparison of Gaussian, SRFT, and SRHT across different matrix sizes and sketch sizes. While theory predicts SRFT/SRHT should be faster for large ℓ, highly optimized BLAS libraries make Gaussian surprisingly competitive in practice.

3.5 Sparse Matrices: CountSketch and Sparse Sign

For sparse matrices, the situation is fundamentally different. Applying a dense Gaussian Ω to a sparse A destroys the sparsity structure, requiring $\mathcal{O}(mn\ell)$ operations regardless of how few nonzeros A contains.

CountSketch solves this with an extremely sparse test matrix. Each column of Ω contains exactly *one* nonzero entry:

$$\Omega_{h(j),j} = s(j), \quad \text{all other entries } 0 \tag{8}$$

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 nonzeros of A once, accumulating into the appropriate rows of Y. Total complexity: $\mathcal{O}(\text{nnz}(A))$—independent of matrix dimensions and sketch size!

Sparse Sign Embedding generalizes CountSketch by allowing $s > 1$ nonzeros per column, trading off sparsity for improved concentration properties.

3.6 Experimental Results: Sparse Matrices

Figure **??** demonstrates the dramatic advantage of CountSketch on sparse data.

The speedup is dramatic: at 0.1% density, CountSketch is over 100× faster than Gaussian. The scaling confirms the theoretical prediction: Gaussian time is nearly constant across density levels (depends on dimensions), while CountSketch time scales linearly with density (depends on nnz).

3.7 Speed-Accuracy Tradeoffs: The Pareto Frontier

Different sketching methods offer different speed-accuracy tradeoffs. Figure **??** shows the Pareto frontier—the best achievable accuracy for each runtime budget.

3.8 Summary: Choosing a Sketching Method

Based on our experiments:

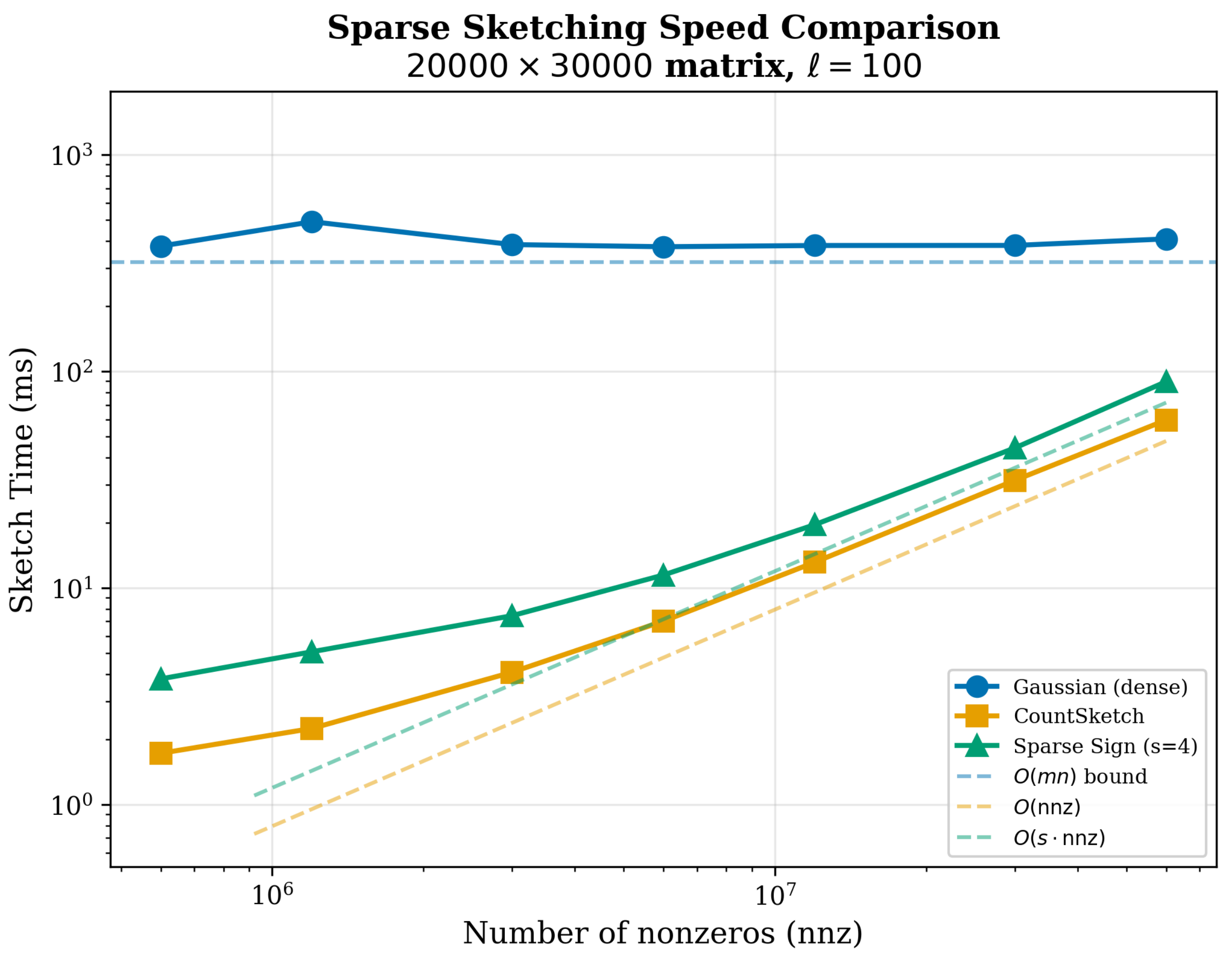

Figure 8: **Sketching speed on sparse matrices.** CountSketch achieves speedups of 100–300× compared to Gaussian at low densities. The key: CountSketch complexity scales with nnz(A), not matrix dimensions. At 0.1% density, this represents a 1000× reduction in effective matrix size.

Scenario	**Recommended Method**
Dense matrix, $n < 10^4$	Gaussian (simple, BLAS-optimized)
Dense matrix, $n > 10^4$	SRHT (asymptotic advantage)
Sparse matrix, any size	CountSketch (scales with nnz)
Very large ℓ needed	SRFT/SRHT (ℓ-independent)

The key insight: **choose the sketching method for speed; use iterations to control accuracy.** All methods achieve similar approximation quality given sufficient power iterations—the difference is purely computational.

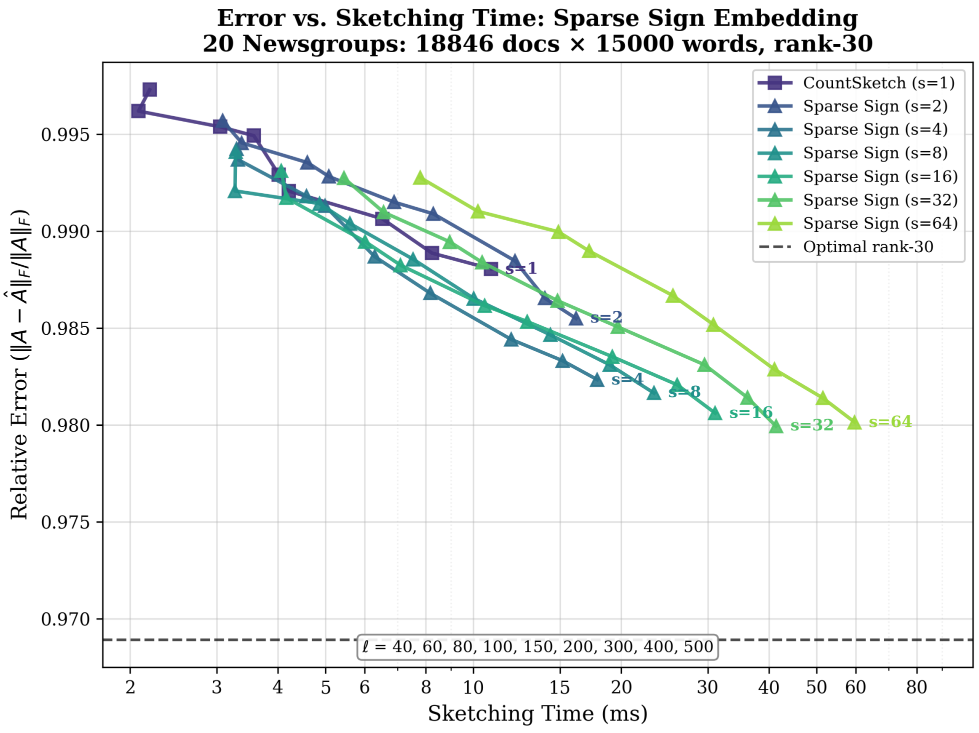

Figure 9: **Speed vs accuracy Pareto frontier.** Each curve shows how approximation error decreases as more time is invested in sketching. The Pareto frontier (lower-left envelope) represents the best tradeoffs. Different methods dominate in different regimes.

[Video Background Separation Results - To Be Added]

Figure 10: **Video background separation using randomized SVD.** [Description of your results]

4 Application: Video Background Separation

4.1 Problem Setup

4.2 Method

4.3 Results

5 Conclusion

5.1 Summary of Contributions

5.2 Practical Recommendations

5.3 Future Directions

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.

C. Musco and C. Musco. Randomized block Krylov methods for stronger and faster approximate singular value decomposition. *NeurIPS*, 2015.

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

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.

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

A Implementation Details

B Additional Experiments