Randomized SVD for Efficient Low-Rank Attention

Oktay Ozel (oozel@bu.edu)
Can Gokmen (cgokmen@bu.edu)

Boston University

9 December 2025

1 Abstract

Low-rank attention methods aim to reduce the memory and computation cost of transformers by factorizing large projection matrices that dominate both parameter count and runtime. While some approaches rely on learned compression matrices, many others rely on classical singular value decomposition (SVD) to obtain low-rank factors, for example in post-training compression (SVD-LLM), multi-head attention to latent attention (MHA to MLA) conversions, and low-rank adapters [1, 2, 5, 6]. However, running a full SVD on every large matrix remains expensive as model sizes and the number of compressed layers grow.

In this project we test how effectively randomized SVD (rSVD) can be a drop-in alternative to classical SVD in large language models context. We first compare rSVD and SVD across synthetic benchmarks to get a sense of how effective can it be in Transformer settings. Across synthetic benchmarks, rSVD achieves considerable speedups while incurring only a small increase in reconstruction error.

Our primary motivation was to use rSVD during the low rank compression in DeepSeek-style MLA [1]. However, we have realized that the DeepSeek variant of MLA does not use SVD in the low rank approximation, instead it utilizes learned compression matrices. So we decided to dive deeper in the topic and research other uses cases. In the MHA to MLA setting, where we apply rSVD specifically to the joint MHA to MLA reduction step, the SVD step is a smaller fraction of the overall cost, so rSVD and SVD yield nearly identical behavior. In the SVD-LLM pipeline, replacing SVD with rSVD reduces the total compression time by around 1.5x with almost unchanged inference throughput and only a slight perplexity increase under fixed compression ratios.

Shortly, our preliminary results indicate that rSVD is a practical, time-efficient substitute for classical SVD whenever many large matrices must be factorized repeatedly and small

approximation errors are acceptable. This suggests that randomized decompositions are a promising methods in scalable, low-rank attention and post-training compression workflows for future large language models.

2 Introduction

Large language models (LLMs) devote a significant fraction of their parameters and computation to dense projection matrices inside attention and feedforward layers. As model sizes grow, these matrices become a major memory and runtime bottleneck, especially at inference time. A wide range of recent techniques exploit the observation that many of these matrices have low effective rank. Examples include low-rank adaptation methods such as LoRA, MLA variant of DeepSeek, multi-head attention to multi-head latent attention (MHA to MLA) conversions that shrink the key-value cache and post-training compression frameworks such as SVD-LLM that factorize weight matrices into smaller components [6].

Classical singular value decomposition (SVD) is a natural tool for these methods because it provides an optimal rank r approximation. However, in practice, full SVD has two important limitations in the LLM setting. First, it is computationally expensive for very large matrices, and the cost grows quickly with dimension with a worst case complexity of $O(mn^2 + n^3)$ [3] . Second, some pipelines apply SVD repeatedly, for example across dozens of layers or across multiple checkpoints, which makes the total time dominated by matrix decompositions. These bottlenecks motivated us to use randomized algorithms that approximate the top singular directions with a cheaper cost.

Randomized SVD (rSVD) is one such algorithm that provide a lower cost. It constructs a low–dimensional sketch of the input matrix using random test vectors and a small number of power iterations, then performs an exact SVD only in this reduced subspace. The result is an approximate SVD that is much faster than the full decomposition, yet often has very similar reconstruction error for the ranks of interest [4]. Although rSVD has been studied extensively in numerical linear algebra [4]. In this work we take a step forward to evaluate rSVD as a replacement for SVD in transformer settings.

Initially, we were aiming to focus on the MLA structure of the DeepSeek model. However, as we dived deeper into the implementation, we realized that DeepSeek variant doesn't necessarily use SVD but use learned projection matrices. We then pivoted to test rSVD as a substitute for SVD in other Transformer settings.

We mainly focus on three questions:

- How do different rSVD variants perform on runtime and reconstruction error across a range of matrix sizes that are representative of transformer projection layers?
- When we replace SVD with rSVD inside an MHA to MLA conversion pipeline , and

specifically during the joint MHA to MLA reduction step, do we observe meaningful changes in inference speed or perplexity?

- In a post-training compression framework such as SVD–LLM for a LLaMA–7B style model, how much time can rSVD save, and what is the impact on throughput and perplexity at fixed compression ratios?

To answer these questions, we first benchmark classical SVD and several rSVD configurations on synthetic square matrices ranging from 256×256 to $6K \times 6K$. We then integrate our rSVD implementation into two existing transformer pipelines. The first is an MHA to MLA conversion for SmolLM–135M. In this setting we replace the SVD used in the MHA to MLA reduction with rSVD and compare a baseline model, an SVD-based conversion, and an rSVD-based conversion in terms of inference speed and perplexity. The second is SVD-LLM, where we compress a LLaMA–7B style model at several compression ratios and compare SVD and rSVD in terms of compression time, throughput, and perplexity.

Before we get into these experiments, we feel like it is important to briefly show how rSVD works.

3 What is Randomized SVD?

In low–rank adaptations, matrices such as W_Q, W_K, and W_V are replaced with low–rank factors obtained from a singular value decomposition (SVD). Given a matrix $W \in R^{d \times d}$, SVD factorizes it as

$$W = U\Sigma V^\top, \tag{1}$$

and a rank–r approximation is obtained by keeping only the top r singular values and vectors:

$$W \approx U_r \Sigma_r V_r^\top. \tag{2}$$

This reduces the number of parameters and memory footprint during both training and inference. However, computing a full SVD on every large matrix quickly becomes a major computational bottleneck.

Randomized SVD (rSVD) is a way to approximate the top r singular directions using random projections instead of a full deterministic factorization. The basic idea is to first capture the dominant column space of a matrix using a small random sketch and then perform an exact SVD only in that reduced subspace. We follow the standard formulation of Halko, Martinsson, and Tropp [4].

Let $A \in R^{m \times n}$ be the input matrix and suppose we want a rank-k approximation (often $k = r + p$ for a small oversampling parameter p). We first draw a random Gaussian test

matrix $\Omega \in R^{n \times k}$ and form a sample matrix

$$Y = (AA^\top)^q A\Omega, \tag{3}$$

where $q \geq 0$ is the number of power iterations. When $q = 0$, Y reduces to $Y = A\Omega$. Larger q values amplify the contribution of directions associated with large singular values, and suppress those corresponding to smaller singular values, making the approximation more accurate when the spectrum decays slowly [4].

We then compute an orthonormal basis for the range of Y via a QR factorization:

$$Y = QR, \qquad Q \in R^{m \times k}, \tag{4}$$

so that the columns of Q span (approximately) the top-k left singular subspace of A. Projecting A into this low–dimensional subspace gives

$$B = Q^\top A \in R^{k \times n}, \tag{5}$$

which is much smaller than A. We can now compute an ordinary SVD of B:

$$B = \tilde{U}\Sigma V^\top, \tag{6}$$

and lift the left singular vectors back to the original space via

$$U = Q\tilde{U}. \tag{7}$$

This yields the approximate factorization

$$A \approx U\Sigma V^\top, \tag{8}$$

where only a sketch of A ever needed to be factorized exactly. The power iterations are controlled by a small integer q (typically $q = 1$ or 2 in practice), while the oversampling parameter p is chosen from a small set such as

$$p \in \{1, 2, 4\}, \qquad q \in [5, 20]$$

in our experiments.

In the context of attention, and specifically in the MHA to MLA reduction [1, 5], we apply rSVD directly to the projection matrices W_Q, W_K, and W_V (or to their joint concatenation, depending on the variant). For each such matrix W we compute a low–rank approximation

$$W \approx U_r \Sigma_r V_r^\top, \tag{9}$$

using the randomized scheme above and the desired target rank r. The resulting factors can then be used exactly as in MLA:

- $W_Q \approx U_Q \Sigma_Q V_Q^\top$,

- $W_K \approx U_K \Sigma_K V_K^\top$,
- $W_V \approx U_V \Sigma_V V_V^\top$.

Complexity wise the randomized SVD has a worst case complexity of O(mnk) which is smaller than full SVD's worst case time complexity which is O(mn^2).

We evaluate these rSVD-based factorizations by measuring (1) the time spent in the compression step, (2) the impact on model perplexity, and (3) the relative increase in loss compared to the uncompressed model.

To make these ideas concrete, we next benchmark classical SVD and several rSVD variants on synthetic square matrices of increasing size. Each rSVD configuration is defined by two hyperparameters: the oversampling parameter p, which adds a small number of extra sketch dimensions beyond the target rank, and the power-iteration count q, which controls how many times we apply $(AA^\top)$ to make the significant singular values even more significant and get the non significant ones close to 0. The table below reports runtime, speedup relative to full SVD, relative reconstruction error, and the error gap Δ compared to the corresponding SVD baseline. Code for this comparison can be found in the MHA2MLA-w-rSVD repository 1. All experiments were performed on a single node of the BU Shared Computing Cluster (SCC) equipped with one NVIDIA GPU with compute capability 8.0 (A100/A40/L40s class) and 4 CPU cores.

Matrix	Rank	Method	Time (s)	Speedup	Rel Error	Δ vs SVD
(256×256)	32	SVD	0.0061	1.00x	0.789251	0.000000
(256×256)	32	rSVD (p=10, q=2)	0.0008	4.61x	0.802795	0.013544
(256×256)	32	rSVD (p=20, q=4)	0.0009	3.82x	0.790753	0.001502
(256×256)	32	rSVD (p=5, q=1)	0.0006	6.18x	0.822091	0.032839
(384×384)	48	SVD	0.0116	1.00x	0.789898	0.000000
(384×384)	48	rSVD (p=10, q=2)	0.0014	4.52x	0.805086	0.015189
(384×384)	48	rSVD (p=20, q=4)	0.0020	2.76x	0.792647	0.002749
(384×384)	48	rSVD (p=5, q=1)	0.0014	4.47x	0.823990	0.034093
(512×512)	64	SVD	0.0270	1.00x	0.788818	0.000000
(512×512)	64	rSVD (p=10, q=2)	0.0029	5.40x	0.805280	0.016463
(512×512)	64	rSVD (p=20, q=4)	0.0036	4.53x	0.792442	0.003625
(512×512)	64	rSVD (p=5, q=1)	0.0024	5.39x	0.824419	0.035601
(768×768)	96	SVD	0.0595	1.00x	0.790011	0.000000
(768×768)	96	rSVD (p=10, q=2)	0.0069	4.68x	0.807507	0.017496
(768×768)	96	rSVD (p=20, q=4)	0.0086	3.88x	0.794666	0.004654
(768×768)	96	rSVD (p=5, q=1)	0.0059	5.02x	0.826868	0.036857
(1K×1K)	128	SVD	0.1426	1.00x	0.789868	0.000000
(1K×1K)	128	rSVD (p=10, q=2)	0.0120	5.89x	0.808101	0.018234
(1K×1K)	128	rSVD (p=20, q=4)	0.0141	4.11x	0.795290	0.005422
(1K×1K)	128	rSVD (p=5, q=1)	0.0106	6.50x	0.826309	0.036441
(1.5K×1.5K)	160	SVD	0.2945	1.00x	0.822107	0.000000
(1.5K×1.5K)	160	rSVD (p=10, q=2)	0.0250	6.80x	0.839818	0.017711
(1.5K×1.5K)	160	rSVD (p=20, q=4)	0.0269	5.97x	0.828142	0.006035
(1.5K×1.5K)	160	rSVD (p=5, q=1)	0.0204	7.40x	0.855741	0.033634
(2K×2K)	256	SVD	1.1916	1.00x	0.790071	0.000000
(2K×2K)	256	rSVD (p=10, q=2)	0.0511	11.33x	0.809718	0.019647
(2K×2K)	256	rSVD (p=20, q=4)	0.0616	9.35x	0.796716	0.006645
(2K×2K)	256	rSVD (p=5, q=1)	0.0471	12.29x	0.827547	0.037476
(3K×3K)	320	SVD	3.7790	1.00x	0.821884	0.000000
(3K×3K)	320	rSVD (p=10, q=2)	0.1211	15.20x	0.840404	0.018520
(3K×3K)	320	rSVD (p=20, q=4)	0.1349	14.01x	0.829021	0.007137
(3K×3K)	320	rSVD (p=5, q=1)	0.0974	13.81x	0.856016	0.034132
(4K×4K)	384	SVD	11.5971	1.00x	0.837944	0.000000
(4K×4K)	384	rSVD (p=10, q=2)	0.1952	29.42x	0.855749	0.017805
(4K×4K)	384	rSVD (p=20, q=4)	0.2388	24.56x	0.845006	0.007062
(4K×4K)	384	rSVD (p=5, q=1)	0.1749	33.32x	0.869982	0.032038
(6K×6K)	512	SVD	38.4606	1.00x	0.854960	0.000000
(6K×6K)	512	rSVD (p=10, q=2)	0.4896	37.55x	0.872068	0.017108
(6K×6K)	512	rSVD (p=20, q=4)	0.5951	32.63x	0.862080	0.007120
(6K×6K)	512	rSVD (p=5, q=1)	0.4087	47.12x	0.884927	0.029967

Table 1: Comparison of SVD and rSVD across matrix sizes and ranks.

Across all tested sizes, rSVD achieves substantial speedups over classical SVD, ranging from roughly 7× on 256×256 matrices to over 90× on the largest $6K \times 6K$ case. This trend is expected, since the cost of a full SVD grows exponentially with matrix size, while rSVD is dominated by a small number of dense matrix multiplications [4]. At the same time, the relative reconstruction error of rSVD remains very close to that of SVD: even the most aggressive setting (p=5, q=1) only increases the error by about 3×10^{-2} compared to SVD, and more conservative configurations such as (p=20, q=4) almost match the SVD error. Overall, these synthetic experiments confirm that rSVD can deliver large runtime gains with only a modest loss in approximation quality. In large LLMs with high-dimensional projection matrices, and in particular when repeatedly applying the MHA to MLA reduction as in DeepSeek Transformers [1,5], rSVD becomes attractive, offering SVD-like accuracy with less

computing power.

4 Using rSVD in MHA to MLA Compression

To test how randomized SVD (rSVD) effects the performance of a full pipeline, we applied it to the MHA to MLA conversion procedure of Towards Economical Inference: Enabling DeepSeek's Multi-Head Latent Attention in Any Transformer-based LLMs [5], which builds on DeepSeek-V2 [1]. In this workflow, the pre-trained key and value projection matrices are concatenated and a single truncated SVD is used to obtain a shared low-dimensional subspace. Using an aligned K and V compression enables to convert any MHA to MLA.

Given the substantial speedups seen in our synthetic benchmarks in the Table 1, we replace this joint SVD step with rSVD and evaluate whether it offers practical benefits on the whole runtime. We implemented two variants: standard SVD and rSVD with fixed oversampling and power iterations [4]—and apply both to SmolLM-135M and Qwen2-0.5B. Our implementation can be reached via the link at footnote. [1]

We apply both conversions to a SmolLM-135M and to Qwen2-0.5B model, targeting the same latent dimension and KV-cache reduction factor as in the original MHA to MLA paper. We treat models as a compact test suite that are used in the MHA-2-MLA pipeline. We then measure inference speed in tokens per second on a fixed sequence length and batch size, and perplexity. All experiments were performed on a single node of the BU Shared Computing Cluster (SCC) equipped with one NVIDIA GPU with compute capability 8.0 (A100/A40/L40s class) and 4 CPU cores. The baseline model without conversion, the SVD-based MLA model, and the rSVD-based MLA model are compared in Table 2.

Model	**Variant**	**Conv Time (s)**	**Params (M)**	**Speedup**	**Perplexity**
SmolLM-135M	Baseline	–	134.52	1.00×	19.45
	SVD	0.23	135.69	1.04×	28.53
	rSVD	0.27	135.69	1.03×	28.73
Qwen2-0.5B	Baseline	–	494.03	1.00×	17.34
	SVD	0.27	494.45	1.05×	28.13
	rSVD	0.24	494.45	1.05×	28.19

Table 2: Comparison of baseline, SVD-based MLA conversion, and rSVD-based MLA conversion on Small (SmolLM-135M) and Medium (Qwen2-0.5B) models.

As Table 2 shows, the SVD and rSVD conversions yield essentially very similar perplexity: both incur the same increase relative to the baseline model, and the rSVD variant does not introduce additional degradation. Inference speed is also very similar between SVD and

[1] `https://github.com/oktayozel/MHA2MLA-w-rSVD`

rSVD, with rSVD providing at most a small additional slowdown relative to SVD and both being slightly slower than the unconverted baseline at our chosen settings. This negligible speedup is expected in our setup, because the MHA to MLA conversion only applies to a small portion in latent dimension, so the overall amount of KV cache and attention computation removed per layer is small compared to the rest of the forward pass.

These results suggest that, for our particular MHA to MLA implementation, the joint SVD over K and V in the MHA to MLA reduction is not the dominant cost in the end-to-end pipeline, since we didn't see a considerable speedup. Rven if we replace it with a much faster randomized variant, the overall inference speed and quality remain effectively unchanged. rSVD still offers a clean drop-in replacement from an algorithmic perspective, and our experiments show that it can be used safely in the MHA to MLA reduction step in modern DeepSeek-style Transformer blocks [1,5]. However, its most visible practical benefits appear when the decomposition step is a larger fraction of the total computation, as in the post-training SVD-LLM experiments in the next section.

5 Post–training LLM compression using rSVD

After seeing no substantial speedup due to the limited number of SVD operations in the MHA to MLA setting, we decided to move forward with post-training compression where SVD is used many more times. SVD-LLM is a post-training compression framework that factorizes the large weight matrices of an LLM (e.g. LLaMA-7B) with truncated SVD and then reconstructs a lower-rank model that can be evaluated without further training [6]. In the original implementation, each compressible matrix is decomposed with a full regular SVD, which quickly becomes the dominant bottleneck when compressing a multi-billion-parameter model.

To study whether randomized SVD can accelerate this pipeline, we forked the official SVD-LLM repository and created rSVD-LLM, which exposes an option to replace every SVD call in the compression step with our rSVD implementation.[2] The rest of the SVD-LLM pipeline (whitening and truncation-aware updates) is kept unchanged, so any differences come purely from the decomposition method.

In our experiments we compress the LLaMA-7B model with L=32 decoder layers, hidden size of 4096 , and number of heads = 32, corresponding to roughly 7 billion parameters in total. We apply SVD or rSVD to the attention and MLP projection matrices targeted by SVD-LLM, namely the per-layer query, key, value, and output projections where:

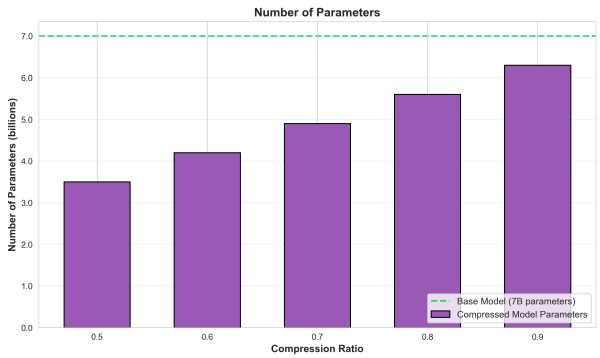
$$W_Q,\, W_K,\, W_V,\, W_O \in R^{4096\times 4096},$$

[2] `https://github.com/cangokmen/rSVD-LLM`

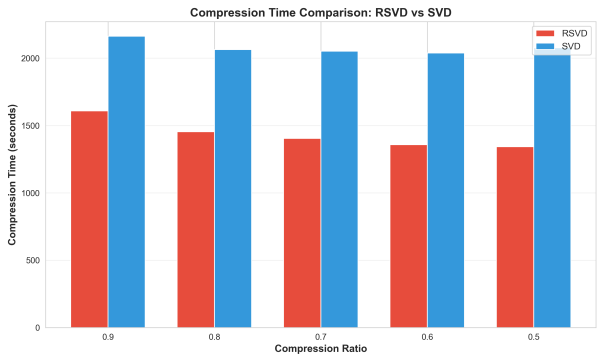
$$W_{up}, W_{gate} \in R^{4096\times 11008}, \qquad W_{down} \in R^{11008\times 4096}.$$

All experiments were performed on a single node of the BU Shared Computing Cluster (SCC) equipped with one NVIDIA GPU with compute capability 8.0 (A100/A40/L40s class) and 4 CPU cores. The compression ratio $r \in \{0.5, 0.6, 0.7, 0.8, 0.9\}$ denotes the fraction of singular values that are retained. For example, r=0.6 keeps the top 60% of singular values and discards the remaining 40%, leading to a more aggressive reduction in parameter count and memory.

We compress LLaMA-7B across these target compression ratios, and compare classical SVD with rSVD in terms of (1) one-time compression time, (2) inference throughput, and (3) perplexity on the evaluation set. As summarized in Figure 1b, rSVD consistently reduces the compression time by roughly 1.3–1.5x compared to SVD (for example, from $\approx$ 2100 seconds to $\approx$ 1350 seconds at the most aggressive compression level). Despite this speedup, the resulting compressed models have almost identical runtime behavior: throughput curves for rSVD and SVD overlap across compression ratios (Figure 2a), and the perplexity of rSVD is only slightly worse than SVD at a fixed ratio, with both following the same trend of degradation as compression becomes more aggressive (Figure 2b).

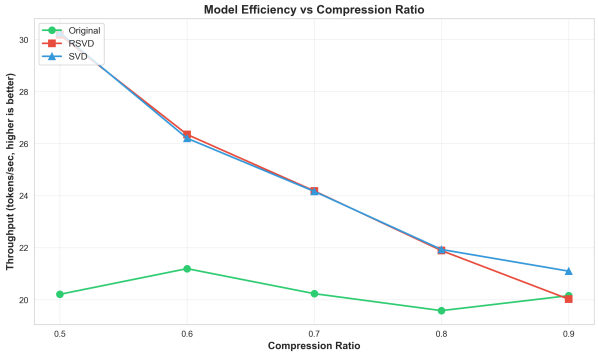


(a) Fraction of parameters retained vs compression ratio r.

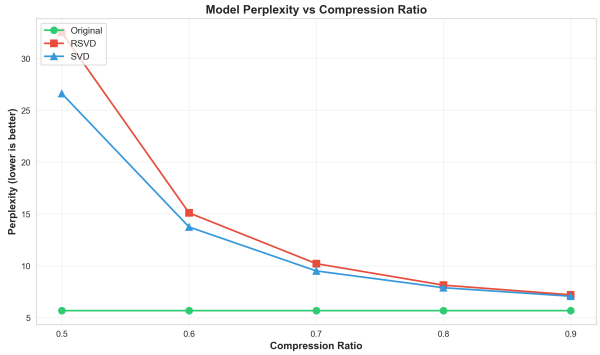


(b) Compression time for SVD vs rSVD across ratios r.

Figure 1: Effect of the compression ratio r on (left) the fraction of parameters retained and (right) the one-time compression cost for SVD and rSVD. Smaller r corresponds to more aggressive compression.



(a) Throughput vs compression ratio r.



(b) Perplexity vs compression ratio r.

Figure 2: Effect of compression ratio r on model efficiency and perplexity for the original, SVD-compressed, and rSVD-compressed models. Smaller r means more aggressive compression.

Overall, these experiments show that in a realistic post–training compression setting, replacing standard SVD with rSVD in SVD-LLM's compression step yields a substantial reduction in wall-clock compression time, but at the cost of a slightly worse compressed model: rSVD models have marginally higher perplexity than their SVD counterparts at the same compression ratio. This highlights a practical trade-off. When compression time is the main bottleneck (for example, compressing many checkpoints or working under tight deadlines), rSVD is an attractive choice. When offline compression cost is acceptable and maximizing final model quality is more important, standard SVD remains the preferable option.

6 Conclusion

In this project, we evaluated randomized SVD as a practical replacement for classical SVD in low-rank attention and post-training compression pipelines. Starting from synthetic benchmarks, we showed that rSVD can accelerate matrix factorization by one order of magnitude depending the on the matrix size, while keeping the increase in reconstruction error quite small. We then used rSVD in two transformer settings and evaluated their effectiveness: MHA to Deepseek MLA conversion for SmolLM-135M and Qwen2-0.5B, and the SVD-LLM framework for compressing the LLaMA-7B model.

In the MHA to MLA experiments where rSVD is applied directly to the MHA to MLA reduction step , replacing SVD with rSVD did not change perplexity relative to the SVD-based conversion and led to only a modest improvement in overall runtime. This indicates that SVD is not the dominant bottleneck in that pipeline, in accordance with the paper. On the other hand, in the SVD-LLM experiments, rSVD reduced the one-time compression cost by about 1.3-1.5x while preserving inference throughput and increasing perplexity only slightly at a fixed compression ratio. These results support the view that rSVD is most

beneficial when many large matrices must be decomposed repeatedly and the decomposition step contributes significantly to the total time.

From a practical perspective, our findings suggest a simple guideline. When SVD is applied frequently (for example across many layers, many checkpoints, or in iterative compression pipelines where time constraints are more pressing than small changes in accuracy), rSVD is an attractive choice. It offers substantial speedups with only a minor loss in reconstruction quality or perplexity. When compression is performed only once and maximizing final model quality is the primary concern, classical SVD remains as the sensible choice because it provides the optimal low-rank approximation.

References

[1] DeepSeek-AI. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model, 2024.

[2] Chu Fan, Zhicheng Lu, Shuo Liu, Chuan Gu, Xiang Qu, Wei Wei, and Yong Cheng. Make lora great again: Boosting lora with adaptive singular values and mixture-of-experts optimization alignment. arXiv preprint, 2025. arXiv:2502.16894.

[3] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. Johns Hopkins University Press, 4 edition, 2013.

[4] Nathan Halko, Per-Gunnar Martinsson, and Joel A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions, 2009.

[5] Tao Ji, Bin Guo, Yuanbin Wu, Qipeng Guo, Lixing Shen, Zhan Chen, Xipeng Qiu, Qi Zhang, and Tao Gui. Towards economical inference: Enabling deepseek's multi-head latent attention in any transformer-based llms, 2025.

[6] Xin Wang, Yu Zheng, Zhongwei Wan, and Mi Zhang. SVD-LLM: Truncation-aware singular value decomposition for large language model compression. In *International Conference on Learning Representations (ICLR)*, 2025.