FAST DIFFERENTIABLE SVD ON GPU VIA POLAR DECOMPOSITION

Anonymous authors
Paper under double-blind review

ABSTRACT

We present a fully GPU-oriented SVD pipeline based on polar decomposition, motivated by iterative methods that rely solely on matrix multiplications, such as the Newton-Schulz iteration. We show that this approach enables up to a $2\times$ speedup compared to standard implementations. Furthermore, we derive a numerically stable backward pass for the polar decomposition and leverage it to obtain a fully differentiable SVD. Our methods are released as open-source implementations in both PyTorch and JAX.

1 INTRODUCTION

Singular value decomposition (SVD) is a basic primitive in numerical linear algebra and a standard tool in machine learning. It underpins PCA Abdi & Williams (2010); Jolliffe & Cadima (2016), global covariance pooling Wang et al. (2017; 2020), tensor decompositions Oseledets (2011); Vannieuwenhoven et al. (2012), and matrix-factorization-based recommender systems Cremonesi et al. (2010); Yuan et al. (2019); Rodpysh et al. (2023). In large scale deep learning, SVD is used for model compression and low-rank adaptation in large language models Meng et al. (2024); Wang et al. (2025); Parkina & Rakhuba (2025), optimization methods for neural networks Jordan et al. (2024); Wang et al. (2018), and regularization techniques Shi et al. (2024).

Standard algorithms for computing SVD, such as QR algorithm Demmel & Kahan (1990) and Jacobi-type methods Hestenes (1958), and their implementations in LAPACK Anderson et al. (1999) and NVIDIA libraries NVIDIA (2025) follow classical numerical linear algebra designs. These approaches rely on Householder reflections, Givens rotations, and QR factorizations, which scale poorly on modern GPUs. Many production implementations also target single precision arithmetic, which limits performance in lower precision regimes.

A factorization closely related to the SVD is the polar decomposition. In recent years, the polar decomposition has become increasingly popular in machine learning pipelines Jordan et al. (2024), although it has not been fully explored in this context. We therefore turn our attention to this decomposition and develop a faster SVD algorithm via polar decomposition, using iterative methods for computing the polar factor, such as the Chebyshev-accelerated Newton–Schulz (CANS) iteration Grishina et al. (2025) and related approaches Amsel et al. (2026); Chen & Chow (2014). These methods use only matrix multiplications and can run in lower precision, such as tensorfloat-32 (tf32) or bfloat-16 (bf16).

Another important challenge is that existing pipelines suffer from numerical issues when directly implementing formulas for differentiating the full SVD. For instance, while PyTorch Paszke et al. (2019) supports differentiation through the SVD, bounded gradient norms are guaranteed only for differentiation through the singular values, which limits the use of backpropagation through the full decomposition despite its potential usefulness in deep learning Qinsi et al. (2025); Wang et al. (2021). We address this issue by introducing a pseudo-gradient at degenerate points and deriving regularized differentiation formulas, showing that the polar decomposition is useful in this context as well. To the best of our knowledge, we are the first to derive a backward formula for the polar decomposition and show that it does not rely on spectral gaps. In contrast, standard SVD differentiation formulas become ill-conditioned when singular values approach each other. Our approach yields a regularized differentiation scheme with controlled gradients for ill-conditioned or nearly rank-deficient matrices. We also propose a regularization to stabilize the backward pass of the eigenvalue decomposition (EVD). Combining these components provides a backward pass through the full SVD. Finally,

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

experiments on a practical LLM compression task demonstrate improved optimization outcomes and model quality.

Our main contributions are as follows:

- We introduce a new algorithm for computing the SVD via polar decomposition, achieving up to a $2\times$ speedup by using matrix multiplications in bf16/tf32 precision. We show that polar decomposition, combined with recent advances Grishina et al. (2025), leads to a practical, end-to-end SVD method with meaningful performance gains on modern hardware.
- We develop a stable regularized approach for differentiating the polar decomposition and the SVD. We formally analyze gradient stability under perturbations and provide theoretical guarantees on how regularization improves the stability of the backward pass and controls the gradient norm, while converging to the unperturbed gradient.

The paper is organized as follows. Section 2 reviews related work, existing approaches to the problem, and their limitations. Section 3 introduces basic definitions, the connection between the polar decomposition and the SVD, and briefly describes iterative methods underlying the proposed computational routines. Sections 4 and 5 present the SVD computation and differentiation methods, respectively, and report separate experiments for each component.

2 Related Work

Computing the SVD is a standard task in numerical linear algebra. GPU-based approaches typically include QR-based algorithms, Jacobi-type methods, and polar-decomposition-based algorithms. QR-based methods Golub & Van Loan (2013) first reduce a matrix to bidiagonal form via Householder reflections or Givens rotations and then apply implicit QR iterations. They are robust, stable, and underlie LAPACK SVD routines Demmel & Kahan (1990), but bidiagonalization creates long dependency chains, limiting scalability on GPU Struski et al. (2024). Jacobi-based methods Hestenes (1958); de Rijk (1989) iteratively apply plane rotations to column pairs to reduce off-diagonal entries of the Gram matrix. Since independent pairs can be processed in parallel, these methods can outperform QR-based algorithms on GPUs for small and medium matrices NVIDIA (2025). Methods based on polar decomposition Higham & Papadimitriou (1994); Nakatsukasa & Higham (2013) compute the SVD by first finding the unitary polar factor and then applying an EVD to the remaining factor. Existing approaches, however, typically compute the polar factor using QR-based iterations, which limits their GPU applicability Nakatsukasa & Higham (2013).

Remark 1. *A simpler approach to computing the SVD of a matrix $M \in \mathbb{R}^{m\times n}$ is to form the Gram matrix $M^\top M$ and compute its EVD, taking singular values as the square roots of the eigenvalues. However, this approach is numerically unstable: forming $M^\top M$ squares the condition number of M, so the relative error in the singular values scales as $\mathcal{O}(\sqrt{\varepsilon_{mach}})$. This makes Gram matrix-based methods unreliable for ill-conditioned or nearly rank-deficient matrices, especially in differentiable settings where gradient stability is essential. See (Demmel, 1997, p. 241) for more details and experimental validation in Appendix B.3.*

Differentiating matrix decompositions appears to be a much less studied problem than computing them. Most of the issues comes from their non-uniqueness for repeated eigenvalues or low-rank cases. This problem has been addressed in the literature with analytic formulas for the reverse mode differentiation of the SVD presented in Ionescu et al. (2015); Townsend (2016); Kanchi & He (2025). However, these expressions have well-known failure modes: gradients from these formulas explode for rank-deficient matrices or when singular values are close, which makes their application problematic in practical settings. Several works have aimed to address these issues. First, Wang et al. (2021) proposed to address the small spectral gap issue via a Taylor approximation. This allows us to bound the gradient norm by choosing an appropriate number of terms and to prevent blow-up when singular values are close. However, that work has focused predominantly on symmetric matrices, and the proposed method can yield biased gradients due to the approximation error from the Taylor remainder. Second, Zhang & Hu (2024) proposed to handle both rank-deficient and small gap cases using the Moore–Penrose pseudoinverse. For full-rank matrices with a large gap this method is equivalent to the analytic solution from Townsend (2016). However, we observe that their derivation contains an error (see Appendix C), and their final gradient formula is incorrect.

3 PRELIMINARIES

Throughout this paper, we focus only on reverse-mode automatic differentiation Griewank & Walther (2008); Baydin et al. (2018), and unless stated otherwise, the term *differentiation* refers to computing the backward pass.

3.1 NOTATION

For any vector $x \in \mathbb{R}^n$, $\|x\|_p = \left(\sum_{i=1}^n |x_i|^p\right)^{1/p}$ denotes the ℓ_p norm ($p \geq 1$). For any matrix $M \in \mathbb{R}^{m\times n}$, $M_{:,i}$ denotes the i-th column of M, $\sigma_i(M)$ denotes the i-th singular value of M, and $\mathrm{vec}(M)$ denotes the vectorization of M. We denote by M^+ the Moore–Penrose pseudoinverse of M, defined via the SVD $M = U\Sigma V^\top$ as $M^+ = V\Sigma^+ U^\top$, $\Sigma^+ = \mathrm{diag}\big(\sigma_1^{-1}, \ldots, \sigma_r^{-1}, 0, \ldots, 0\big)$, where $r = \mathrm{rank}(M)$. For any square invertible matrix M, let $\kappa_2(M) = \|M\|_2\|M^{-1}\|_2$ be the condition number of M, where $\|\cdot\|_2$ is the spectral norm. For matrices $A, B \in \mathbb{R}^{m\times n}$, the Hadamard (elementwise) product is denoted by $A \odot B$, $(A \odot B)_{ij} = A_{ij}B_{ij}$. Given a scalar-valued loss function $\mathcal{L}$, we denote by $\overline{X} = \partial\mathcal{L}/\partial X$ the gradient of $\mathcal{L}$ with respect to a variable X. Notice that $\overline{X}$ has the same shape as X and arises in the differential expression with the increment $\mathrm{d}X$ Edelman & Johnson (2022): $\mathrm{d}\mathcal{L} = \langle \overline{X}, \mathrm{d}X\rangle = \mathrm{tr}(\overline{X}^\top \mathrm{d}X)$. For a square matrix $A \in \mathbb{R}^{n\times n}$, we denote its symmetric part by $A_{\mathrm{sym}} = \frac{1}{2}\big(A + A^\top\big)$. We denote by $A_{\text{off-diag}}$ the matrix A with its diagonal set to 0.

3.2 POLAR DECOMPOSITION

Definition 3.1 (Polar decomposition)**.** *Any matrix $M \in \mathbb{R}^{m\times n}$ with $m \geq n$ admits a (right) polar decomposition $M = WH$, where $W \in \mathbb{R}^{m\times n}$ has orthonormal columns, i.e., $W^\top W = I_n$, and $H \in \mathbb{R}^{n\times n}$ is symmetric positive semidefinite.*

The polar decomposition is closely related to the SVD. Let $M = U\Sigma V^\top$ be the SVD of $M \in \mathbb{R}^{m\times n}, m \geq n$, where $U \in \mathbb{R}^{m\times n}$ and $V \in \mathbb{R}^{n\times n}$ have orthonormal columns and $\Sigma \in \mathbb{R}^{n\times n}$ is diagonal with singular values $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$. Using the SVD, M can be rewritten as

$$M = \underbrace{UV^\top}_{W}\underbrace{V\Sigma V^\top}_{H}, \tag{1}$$

which is exactly the Definition 3.1. The matrix H is symmetric positive semidefinite with eigenvalues being equal to singular values of M. Moreover, its eigenvectors are the right singular vectors of M.

4 SVD COMPUTATION

While the polar decomposition can be obtained directly from the SVD, our interest lies in the opposite direction: computing the SVD via the polar decomposition Higham & Papadimitriou (1994); Nakatsukasa & Higham (2013); Higham (2015).

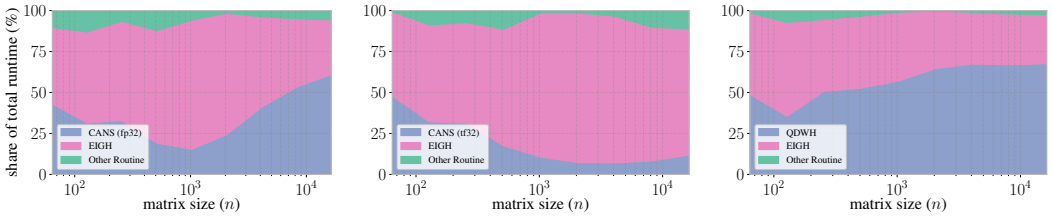
Our algorithm. We first compute the polar factor W and then recover the remaining SVD components from the symmetric factor $H = W^\top M$ using an EVD: $H = V\Sigma V^\top$. The last step is obtained by setting $U = WV$, and finally

$$M = (WV)\Sigma V^\top = U\Sigma V^\top.$$

There exist several iterative methods for computing the polar factor (e.g., Newton-Schulz iteration Kovarik (1970); Björck & Bowie (1971)). Recent works have proposed modified polynomial iterations, such as CANS Grishina et al. (2025) and Polar-Express Amsel et al. (2026), which improve the practical efficiency of such methods. We use the CANS iteration in our algorithm based on the results of an ablation study (see Appendix B.4 for details). A detailed description of the algorithm is provided in Algorithm 1.

Remark 2. *Although Muon Jordan et al. (2024) is often described as using Newton–Schulz iterations, this is not entirely accurate. Muon uses a different polynomial iteration, which is typically run for a small fixed number of steps and does not converge precisely to the polar factor, since an accurate polar decomposition is not required for the optimizer. As our goal is to compute the polar factor accurately, we cannot directly use the Muon polynomial for orthogonalization.*

Numerical stability of such iterations has been discussed in Amsel et al. (2026); Chen & Chow (2014). The stability of CANS follows a pattern similar to that of classical Newton-Schulz-type iterations Nakatsukasa & Higham (2012). In addition, the parameters of CANS control a trade-off between convergence speed and finite-precision error: faster convergence comes at the cost of lower accuracy. We analyze this trade-off for different arithmetic precisions and choose the parameters that give the best balance in our experiments (see Appendix B.6 for details).



(a) CANS SVD (fp32) algorithm (b) CANS SVD (tf32) algorithm (c) QDWH SVD algorithm

Figure 1: Comparison of different parts of the SVD computation by their share of the total runtime for a random matrix $M \in \mathbb{R}^{n\times n}$ with $\kappa_2(M) = 10$ on an NVIDIA B200 GPU. We report the shares of polar-factor computation (in blue), symmetric EVD (EIGH) (in pink), and other routines (in green) for different polar-decomposition-based SVD algorithms.

As an alternative to Newton–Schulz-type iterations, the QDWH iteration Nakatsukasa et al. (2010) has been widely used in practice in recent decades for computing the polar factor due to its cubic convergence rate. However, QDWH relies heavily on repeated QR factorizations, which scale poorly on GPUs (see Appendix B.1). As a result, in our experiments, the QDWH computation accounts for more than half of the total SVD runtime on modern GPUs (see Figure 1c). Additionally, CANS uses only matrix multiplications, which allows us to exploit efficient bf16 and tf32 matrix multiplication routines, substantially accelerating the SVD and shifting the bottleneck to the EVD step (see Figure 1b). This makes CANS more suitable than QDWH for GPU oriented pipelines.

Algorithm 1 CANS SVD. Steps marked in blue color can be executed in half precision

Require: $M \in \mathbb{R}^{m\times n}$ $(m \geq n)$, tolerance $\varepsilon_{\text{QR}} > 0$
Ensure: $U, \Sigma, V^\top$ – SVD of M
1: Normalize M before NS iterations ▷ see Appendix A.1
2: Perform CANS preprocessing on M
3: Compute polar factor W using CANS
4: $H \leftarrow W^\top M$
5: Compute EVD of $H = V\Sigma V^\top$
6: $U \leftarrow WV$
7: **if** $\exists\, i$ such that $\left|\|U_{\cdot,i}\|_2 - 1\right| > \varepsilon_{\text{QR}}$ **then**
8: Orthogonalize U using QR ▷ see Appendix A.2
9: **end if**
10: **return** $U, \Sigma, V^\top$

4.1 Experiments

In this section, we present an empirical evaluation of the proposed algorithm for computing the SVD. We compare the following methods:

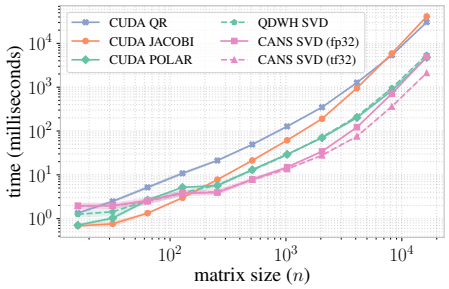
- **CANS SVD (fp32 and tf32[1] versions).** Algorithm 1 computed in fp32 precision and in mixed precision (fp32 and tf32, wherever possible).
- **QDWH SVD.** Our implementation of SVD via polar decomposition using the QDWH iteration proposed in Nakatsukasa & Higham (2013).
- **CUDA POLAR.** cuSOLVER implementation of SVD via polar decomposition using the QDWH iteration (*driver=gesvdp*).
- **CUDA QR.** cuSOLVER implementation of QR-based SVD (*driver=gesvd*).
- **CUDA JACOBI.** cuSOLVER implementation of Jacobi-based SVD (*driver=gesvdj*).

Importantly, CUDA POLAR and CUDA QR do not provide any hyperparameters according to the cuSOLVER documentation[2]. CUDA JACOBI provides hyperparameters controlling accuracy

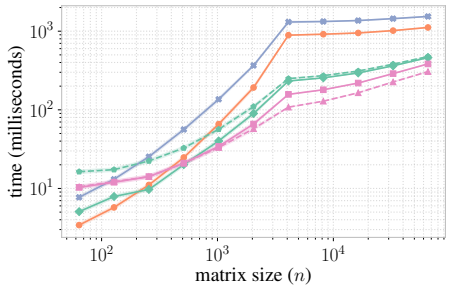
[1]We additionally experiment with the bf16 version, but it provides only negligible speedup over tf32, see Appendix B.7.

[2]https://docs.nvidia.com/cuda/cusolver/index.html

when called from CUDA (see Appendix B.5 for details on choosing hyperparameters). Notably, JAX Bradbury et al. (2018) and PyTorch Paszke et al. (2019) do not provide API access to set these parameters. The hyperparameters chosen for CANS SVD are provided in Appendix A.4.



(a) Square matrices M with $\kappa_2(M) = 10$.



(b) Rectangular matrices $M \in \mathbb{R}^{4096\times n}$ with $\sigma_1(M)/\sigma_{\min(4096,n)}(M) = 10$.

Figure 2: Comparison of SVD algorithms on random matrices. The total wall-clock time is averaged over 100 trials and reported with the standard deviation. **Note:** The algorithms have different accuracy levels, see Tables 1 and 2 for a more comprehensive analysis. The experiments are conducted on an NVIDIA B200 GPU.

Square matrices. We compare the accuracy of the algorithms for various condition numbers using fixed-size random square matrices in Tables 1 and 2. In Figure 2a, we measure the performance of all algorithms on random square matrices $M \in \mathbb{R}^{n\times n}$ with $\kappa_2(M) = 10$, as the size n increases. The algorithms can be split into two groups by their accuracy: (i) less accurate algorithms (reconstruction error $\sim 10^{-3}$) that are fast - CANS SVD in mixed precision and CUDA JACOBI, and (ii) more accurate algorithms (reconstruction error $\sim 10^{-6}$) that are slow - CANS SVD in single precision, CUDA POLAR, and CUDA QR.

For the first group, CANS SVD (tf32 version) is preferable for medium-sized and large matrices, providing up to a $2\times$ speedup compared to CUDA POLAR and up to a $10\times$ speedup compared to CUDA JACOBI and CUDA QR. For small matrices, however, CUDA JACOBI is faster.

For the second group, CANS SVD in single precision is preferable for medium-sized and large matrices. For medium-sized matrices, it is up to $2\times$ faster than the other baselines, while for $n = 2^{14}$, QDWH SVD and CUDA POLAR start to match the performance of CANS SVD. For small matrices, CUDA POLAR is faster than the other algorithms. However, CUDA POLAR failed to compute the SVD for matrices with condition numbers greater than 10^3, while our re-implementation of this algorithm (QDWH SVD) performs well in this setting.

We note that our algorithms are implemented in JAX for benchmarking, so writing custom kernels may substantially speed them up and increase the gap with CUDA baselines for small matrices. Moreover, we expect that, with new generations of NVIDIA GPUs, CANS SVD can become even faster due to the trends observed in Appendix B.1.

Table 1: Comparison of the accuracy of SVD algorithms on a random square matrix $M \in \mathbb{R}^{4096\times 4096}$ for different condition numbers. The reported metric is the relative reconstruction error $\|M - U\Sigma V^\top\|_F/\|M\|_F$, with standard deviation. Results are averaged over 100 sampled matrices. Red indicates cases with significantly large errors.

Method	10	10^2	10^3	10^4	10^6
CANS SVD (fp32)	$4.9_{\pm.0}\cdot 10^{-6}$	$4.2_{\pm.0}\cdot 10^{-6}$	$4.3_{\pm.1}\cdot 10^{-6}$	$4.4_{\pm.0}\cdot 10^{-6}$	$5.3_{\pm.1}\cdot 10^{-6}$
CANS SVD (tf32)	$9.7_{\pm.0}\cdot 10^{-4}$	$8.2_{\pm.0}\cdot 10^{-4}$	$8.5_{\pm.0}\cdot 10^{-4}$	$8.7_{\pm.0}\cdot 10^{-4}$	$1.2_{\pm.0}\cdot 10^{-3}$
QDWH SVD	$3.4_{\pm.0}\cdot 10^{-6}$	$3.3_{\pm.0}\cdot 10^{-6}$	$3.7_{\pm.0}\cdot 10^{-6}$	$5.7_{\pm.0}\cdot 10^{-6}$	$6.2_{\pm.0}\cdot 10^{-6}$
CUDA POLAR	$4.3_{\pm.0}\cdot 10^{-6}$	$5.5_{\pm.0}\cdot 10^{-6}$	$7.6_{\pm.0}\cdot 10^{1}$	$7.6_{\pm.0}\cdot 10^{1}$	$7.6_{\pm.0}\cdot 10^{1}$
CUDA QR	$1.8_{\pm.1}\cdot 10^{-5}$	$2.7_{\pm.1}\cdot 10^{-5}$	$1.5_{\pm.0}\cdot 10^{-5}$	$1.6_{\pm.1}\cdot 10^{-5}$	$1.1_{\pm.0}\cdot 10^{-5}$
CUDA JACOBI	$9.7_{\pm.4}\cdot 10^{-4}$	$8.4_{\pm.4}\cdot 10^{-4}$	$6.8_{\pm.8}\cdot 10^{-4}$	$6.1_{\pm.6}\cdot 10^{-4}$	$5.9_{\pm.7}\cdot 10^{-4}$

Table 2: Comparison of the accuracy of SVD algorithms on a random square matrix $M \in \mathbb{R}^{4096\times4096}$ for different condition numbers. The reported metric is the relative orthogonality error $\max\{\|U^\top U - I\|_F/\|I\|_F, \|V^\top V - I\|_F/\|I\|_F\}$, with standard deviation. Results are averaged over 100 sampled matrices.

Method	10	10^2	10^3	10^4	10^6
CANS SVD (fp32)	$3.7_{\pm.0} \cdot 10^{-6}$	$3.6_{\pm.0} \cdot 10^{-6}$	$3.2_{\pm.9} \cdot 10^{-6}$	$3.3_{\pm.0} \cdot 10^{-6}$	$2.4_{\pm.0} \cdot 10^{-6}$
CANS SVD (tf32)	$6.6_{\pm.0} \cdot 10^{-4}$	$6.2_{\pm.0} \cdot 10^{-4}$	$2.8_{\pm.0} \cdot 10^{-6}$	$6.2_{\pm.0} \cdot 10^{-4}$	$6.2_{\pm.0} \cdot 10^{-4}$
QDWH SVD	$3.6_{\pm.0} \cdot 10^{-6}$	$3.5_{\pm.0} \cdot 10^{-6}$	$3.4_{\pm.0} \cdot 10^{-6}$	$3.2_{\pm.0} \cdot 10^{-6}$	$3.0_{\pm.0} \cdot 10^{-6}$
CUDA POLAR	$4.1_{\pm.0} \cdot 10^{-6}$	$4.0_{\pm.0} \cdot 10^{-6}$	$8.9_{\pm.0} \cdot 10^{-6}$	$8.1_{\pm.0} \cdot 10^{-6}$	$7.0_{\pm.0} \cdot 10^{-6}$
CUDA QR	$9.6_{\pm.2} \cdot 10^{-6}$	$1.4_{\pm.0} \cdot 10^{-5}$	$1.2_{\pm.0} \cdot 10^{-5}$	$1.4_{\pm.1} \cdot 10^{-5}$	$9.7_{\pm.4} \cdot 10^{-6}$
CUDA JACOBI	$2.1_{\pm.0} \cdot 10^{-3}$	$2.1_{\pm.0} \cdot 10^{-3}$	$2.2_{\pm.1} \cdot 10^{-3}$	$2.3_{\pm.1} \cdot 10^{-3}$	$2.6_{\pm.1} \cdot 10^{-3}$

Rectangular matrices. Lastly, we compare different algorithms for computing the SVD of rectangular matrices. We fix the first dimension of the matrix to 4096 and vary the second dimension. For a more comprehensive analysis, we apply the QR preprocessing trick to all algorithms: we first compute the thin QR decomposition of the matrix and then compute the SVD of the triangular factor:

$$M = QR = QU\Sigma V^\top = (QU)\Sigma V^\top.$$

Figure 2b presents the results. The differences from the square case are minor. For small matrices, CUDA JACOBI and CUDA POLAR provide the best performance-accuracy trade-offs with the optimal choice depending on the required accuracy. However, for medium and large matrices, CANS SVD consistently achieves higher performance.

5 Stable Differentiation of Matrix Decompositions

In this work, we address two challenges in differentiating matrix factorizations. First, at repeated eigenvalues or singular values, the derivatives of the EVD and SVD factors are generally undefined, and gradients may grow without bound as such points are approached. We introduce a pseudo-gradient that specifies a backward rule at repeated values and agrees with the classical backward at nondegenerate points. Second, finite-precision arithmetic introduces errors that can be strongly amplified by ill-conditioned backward formulas, particularly near rank deficiency or small spectral gaps. We regularize the potentially singular terms in these formulas to control their magnitude and damp the amplification of numerical errors.

5.1 Differentiating the Polar Decomposition

Although forward-mode differentiation (JVP) of the polar decomposition and generalized matrix functions has been studied in Gawlik & Leok (2017); Noferini (2017), our focus lies in a different direction: reverse-mode differentiation (VJP) of decompositions. To the best of our knowledge, this is the first result to derive backward formulas for the polar decomposition.

Theorem 5.1. *Let $M \in \mathbb{R}^{m\times n}$ $(m \geq n)$ have full-rank. Let $\mathcal{L}(M) = \ell(W, H)$ be a scalar loss and let $\overline{W}$ and $\overline{H}$ be the respective gradients. Set $C := (\overline{H} - W^\top \overline{W}\, H^{-1})_{\mathrm{sym}}$. Define the mask matrix*

$$T_{ij} := \begin{cases} \frac{1}{\sigma_i+\sigma_j}, & \sigma_i + \sigma_j > 0, \\ 0, & \sigma_i + \sigma_j = 0, \end{cases} \qquad 1 \leq i, j \leq n,$$

and $X := V\Big(T \odot (V^\top C V)\Big)V^\top$. Then $\overline{M} = \overline{W}\, H^{-1} + 2MX$.

Discussion. The proof is provided in Appendix D. Differentiating the polar decomposition through the SVD introduces inverse-gap terms whose analytical cancellation can be lost in finite precision. This instability is discussed for the matrix square root in Song et al. (2022). Our explicit backward formula avoids these terms: vanishing singular value gaps introduce no additional ill-conditioning, provided the smallest singular value stays bounded away from zero.

Theorem 5.1 provides an exact backward formula for a full-rank matrix M. However, this formula becomes increasingly ill-conditioned as the smallest singular value of M approaches zero. In

finite-precision arithmetic, such small singular values may be severely distorted by numerical and accumulated errors even when both M and its approximation remain full rank. Thus the central question is:

How can we reliably evaluate the backward formula when M is full rank but numerically close to rank deficiency?

In applications, we are interested in the gradient of a loss function with respect to an exact matrix M, while in practice we only have access to a perturbed matrix $\widetilde{M}$. Even when $\|M - \widetilde{M}\|$ is small, the smallest singular values of the two matrices, and consequently the corresponding gradients, may differ drastically.

We show that an appropriate regularization prevents the amplification of these errors and provides a stable approximation of the desired gradient that converges to the exact one as the perturbation vanishes.

Proposition 5.2. *Let $M, \widetilde{M} \in \mathbb{R}^{m\times n}$, $m \geq n$, be full-rank matrices such that $\|M - \widetilde{M}\|_F \leq \varepsilon$, with polar decompositions $M = WH$ and $\widetilde{M} = \widetilde{W}\widetilde{H}$, where $\widetilde{H} = \widetilde{V}\widetilde{\Sigma}\widetilde{V}^\top$, $\widetilde{\Sigma} = \operatorname{diag}(\widetilde{\sigma}_1, \ldots, \widetilde{\sigma}_n)$. For incoming gradients $\overline{W}, \overline{H}$, define*

$$\widetilde{H}_\varepsilon^{-1} = \widetilde{V}\widetilde{\Sigma}(\widetilde{\Sigma}^2 + \varepsilon^{1/2} I)^{-1}\widetilde{V}^\top, \qquad \widetilde{B}_\varepsilon = \left(\overline{H} - \widetilde{H}_\varepsilon^{-1}\widetilde{M}^\top \overline{W}\widetilde{H}_\varepsilon^{-1}\right)_{\mathrm{sym}},$$

and

$$\widetilde{X}_\varepsilon = \widetilde{V}\left[\frac{\widetilde{\sigma}_i + \widetilde{\sigma}_j}{(\widetilde{\sigma}_i + \widetilde{\sigma}_j)^2 + \varepsilon^{1/4}}(\widetilde{V}^\top \widetilde{B}_\varepsilon \widetilde{V})_{ij}\right]_{i,j=1}^{n} \widetilde{V}^\top.$$

Then the regularized gradient $\overline{\widetilde{M}}_\varepsilon = \overline{W}\widetilde{H}_\varepsilon^{-1} + 2\widetilde{M}\widetilde{X}_\varepsilon$ satisfies, for all sufficiently small $\varepsilon > 0$: $\|\overline{M} - \overline{\widetilde{M}}_\varepsilon\|_F \leq c\,\varepsilon^{1/4}$, where $c = c(M, \overline{W}, \overline{H})$ is independent of ε.

The proof is provided in Appendix E. This result suggests that the regularized gradient $\overline{\widetilde{M}}_\varepsilon$ at the perturbed point converges to the exact gradient for the unperturbed point $\overline{M}$ as $\varepsilon \to 0$. The final algorithm is given in Algorithm 2.

5.2 Differentiating the SVD

Recall the connection between the polar decomposition and the SVD equation 1. Therefore, to differentiate the SVD it only remains to differentiate the EVD of H.

Although differentiation through the EVD and SVD is classical and well studied Giles (2008) (see also Appendix F), the derivatives of the decomposition factors are generally undefined at repeated eigenvalues or singular values. Moreover, gradients can become arbitrarily large as spectral gaps vanish, causing severe numerical instability Wang et al. (2021). In this work, we introduce a pseudo-gradient that provides a backward rule at such degenerate points and agrees with the classical backward wherever the decomposition factors are differentiable.

Algorithm 2 Polar Decomposition Backward

Require: $M \in \mathbb{R}^{m\times n}$ with polar decomposition $M = WH$, gradients $\overline{W}, \overline{H}$, and $\varepsilon > 0$
Ensure: Gradient $\overline{M}$

1: EVD of $H = V\Sigma V^\top$
2: $H_\varepsilon^{-1} \leftarrow V\left(\Sigma(\Sigma^2 + \varepsilon^{1/2} I)^{-1}\right)V^\top$
3: $B \leftarrow (\overline{H} - H_\varepsilon^{-1} M^\top \overline{W} H_\varepsilon^{-1})_{\mathrm{sym}}$
4: $\widetilde{X}_{ij} \leftarrow \frac{(\sigma_i+\sigma_j)}{(\sigma_i+\sigma_j)^2+\varepsilon^{1/4}} \cdot (V^\top B V)_{ij}$
5: $X \leftarrow V\widetilde{X}V^\top$
6: $\overline{M} \leftarrow \overline{W}H_\varepsilon^{-1} + 2MX$
7: **return** $\overline{M}$

Definition 5.3 (EVD pseudo-gradient)**.** *Let $M \in \mathbb{R}^{n\times n}$ be symmetric positive definite, and fix an eigendecomposition $M = V\Sigma V^\top$. Let $\overline{V}$ and the diagonal matrix $\overline{\Sigma}$ denote the incoming gradients of a differentiable scalar loss $\mathcal{L}(M) = \ell(V, \Sigma)$.*

The EVD pseudo-gradient at this factorization, denoted by $\overline{M}_\star$, is the solution of

$$\min_{G=G^\top} \|G\|_F^2 \qquad \text{subject to} \qquad \langle G, dM\rangle_F = \langle \overline{V}, \underbrace{V\Omega}_{dV}\rangle_F + \langle \overline{\Sigma}, \underbrace{D}_{d\Sigma}\rangle_F$$

for every diagonal D and skew-symmetric Ω with $\Omega_{ij} = 0$ if $\sigma_i = \sigma_j$.

Table 3: Comparison of the gradient norm of SVD backward pass algorithms on a random square matrix $M \in \mathbb{R}^{4096\times4096}$ for different spectral gaps. The reported metric is the Frobenius norm of the gradient $\|\overline{M}\|_F$. Results are averaged over 100 sampled matrices.

Method \ Spec.gap	1	10^{-1}	10^{-2}	10^{-3}	10^{-4}	10^{-5}	10^{-6}
Analytical Townsend (2016)	$66.0_{\pm.6}$	$67.0_{\pm1.0}$	$100.0_{\pm41.0}$	$690.0_{\pm580.0}$	$5500.0_{\pm4600.0}$	NaN	NaN
Taylor Wang et al. (2021)	$49.0_{\pm.3}$	$49.0_{\pm.3}$	$49.0_{\pm.3}$	$49.0_{\pm.3}$	$49.0_{\pm.3}$	$49.0_{\pm.3}$	$49.0_{\pm.3}$
Tikhonov (ours)	$66.0_{\pm.6}$	$67.0_{\pm1.0}$	$100.0_{\pm41.0}$	$690.0_{\pm580.0}$	$5500.0_{\pm4600.0}$	$66.0_{\pm.6}$	$66.0_{\pm.6}$

The Definition 5.3 selects the symmetric matrix of minimum Frobenius norm among those reproducing the loss differential along the specified factor perturbations. The restriction on Ω excludes rotations within repeated eigenvalue eigenspaces: these rotations commute with Σ and therefore leave M unchanged, although they may change $\ell(V, \Sigma)$. Their contribution to the loss differential is disregarded by definition, rather than assumed to vanish. Existence and uniqueness are established in Appendix F.

Theorem 5.4. *Let $M \in \mathbb{R}^{n\times n}$ be symmetric positive definite, and fix an EVD: $M = V\Sigma V^\top$, $\Sigma = \operatorname{diag}(\sigma_1, \dots, \sigma_n)$. Let $\overline{V}$ and the diagonal matrix $\overline{\Sigma}$ be the incoming gradients of a differentiable scalar loss $\mathcal{L}(M) = \ell(V, \Sigma)$. Define*

$$K_{ij} = \begin{cases} \dfrac{1}{\sigma_i - \sigma_j}, & \sigma_i \neq \sigma_j, \\ 0, & \sigma_i = \sigma_j, \end{cases} \qquad S = \frac{1}{2}K^\top \odot \left(V^\top\overline{V} - \overline{V}^\top V\right).$$

Then the EVD pseudo-gradient from Definition 5.3 is given by $\overline{M}_\star = V(S + \overline{\Sigma})V^\top$.

Proof is provided in Appendix F. Similarly to the polar decomposition, in practical computations we operate in finite-precision arithmetic and therefore cannot reliably determine whether eigenvalues are exactly equal. Consequently, the notion of equality between eigenvalues must be interpreted relative to machine precision.

By analogy with Proposition 5.2, the same principle can be applied to the differentiation of the EVD. In particular, the only potentially singular object in the backward formula is the matrix K, which encodes the solution of the commutator (Lyapunov-type) equation $\Sigma S - S\Sigma = -B$, where B is defined in Algorithm 2. As in the polar case, this equation is singular whenever eigenvalues coincide, and its solution is not unique.

We introduce a smooth ε-regularization of K by setting $(K_\varepsilon)_{ij} = \frac{\sigma_i - \sigma_j}{(\sigma_i-\sigma_j)^2+\varepsilon^2}$, where $\varepsilon > 0$. This corresponds to Tikhonov regularization of the associated linear system. For each $\varepsilon > 0$, the resulting backward formula depends smoothly on the factors and incoming gradients. For a fixed eigendecomposition and fixed incoming gradients, it converges to the pseudo-gradient from Theorem 5.4 as $\varepsilon \to 0$. The full pipeline is presented in Appendix I.

5.3 Experiments

In this section, we present an empirical evaluation of the proposed algorithm for computing the backward-pass SVD.

Synthetic Experiments. First, we validate the correctness of the proposed algorithm. We compare the following methods:

- **Analytical formula Townsend (2016).** The standard closed-form SVD backward expression.
- **Taylor approximation Wang et al. (2021).** A Taylor series approximation-based method.
- **Our method**. We implement differentiation of the SVD directly from our formulas for the polar decomposition and the EVD. The arising linear equations are solved using Tikhonov regularization[3]; see Algorithm 2 (we set ε to 10^{-8}).

[3]Although we do not provide a dedicated theoretical analysis, the Tikhonov regularization in the backward pass can be replaced by a δ-regularized Moore–Penrose pseudoinverse Golub & Van Loan (2013), yielding an alternative well-defined and numerically stable gradient in degenerate or ill-conditioned cases.

We compare the gradient norms of the methods for various spectral gaps. Gradients are computed for the following loss function: $\mathcal{L}(U, \Sigma, V) = \langle A, U\rangle + \langle B, \Sigma\rangle + \langle C, V\rangle$. We sample A, B, C with i.i.d. entries from the standard normal distribution. Table 3 presents the results of differentiating the loss function. Gradients in the analytical formula explode when the spectral gap is less than 10^{-4} and produce NaNs. Both of our methods, as well as the Taylor approximation, are stable when applied to matrices with close singular values. However, the Taylor approximation is inaccurate: its norm does not match the analytical formula Townsend (2016) even for large spectral gaps. Both of our methods provide gradients with the same norm as the standard closed-form expression in the case of large spectral gaps. Additionally, we validate our implementation with finite-difference checks in the code.

LLM Compression. Here, we demonstrate how our formulas can be applied to large language model compression. We follow the Dobi-SVD pipeline Qinsi et al. (2025). Instead of compressing all weight matrices with the same ratio, as in classical approaches Parkina & Rakhuba (2025), we learn layer-specific ranks by differentiating through the SVD of activations. More precisely, let $\{W_i, X_i\}_{i=1}^n$ denote the weight matrices of linear layers and its inputs respectively. Each layer has a trainable parameter $\gamma_i \in \mathbb{R}_+$, interpreted as a continuous relaxation of its rank. At each training step, for an input batch X_i, we truncate the activations $W_i X_i$ using the SVD and minimize the loss with a regularizer that promotes the desired compression ratio.

We use the LLaMA-2-7B model Touvron et al. (2023) with 256 random training samples and 256 validation samples from the WikiText-2 dataset Merity et al. (2016). For a clearer comparison of SVD pipelines, we do not perform quantization or weight-update steps, unlike Dobi-SVD Qinsi et al. (2025). We replace the Taylor-based differentiation used in Dobi-SVD with our backward pass. To ablate the effect of regularization, we test several values of ε.

Table 4 reports the results. We compare perplexity (PPL), validation loss, and seconds per iteration. At the compression ratio of 80%, all methods remain close, but our method with $\varepsilon = 1.0$ achieves the best PPL and validation loss. At 60%, the best two variants are $\varepsilon = 0.1$ and $\varepsilon = 1.0$, both improving over Dobi-SVD, which suggests that the proposed backward pass remains useful beyond the low-compression regime. The advantage becomes most pronounced at the aggressive 40% compression ratio: our method with $\varepsilon = 1.0$ reduces PPL from 13.34 to 11.40 and validation loss from 14.72 to 12.46. This indicates that stable gradients are especially important when the rank-selection problem becomes harder. The ablation also shows that the regularization strength matters. The value $\varepsilon = 1.0$ gives the best overall trade-off, whereas stronger regularization degrades quality. Finally, our pipeline yields about a $1.35\times$ speedup.

Table 4: Metric values of various compression methods. Experiments were conducted using the *LLaMA-2-7B* model on the WikiText-2 dataset.

Ratio	Method	PPL	Val Loss	sec. / iter.
100%	LLaMA-2-7B	5.53	6.25	–
80%	Dobi-SVD	6.95	7.17	75.09
	Our ($\varepsilon = 0.1$)	6.90	7.19	55.62
	Our ($\varepsilon = 1.0$)	**6.87**	**7.11**	55.62
	Our ($\varepsilon = 10$)	6.93	7.17	55.62
60%	Dobi-SVD	7.47	7.83	75.09
	Our ($\varepsilon = 0.1$)	**7.33**	**7.62**	55.62
	Our ($\varepsilon = 1.0$)	7.34	7.63	55.62
	Our ($\varepsilon = 10$)	7.34	7.72	55.62
40%	Dobi-SVD	13.34	14.72	75.09
	Our ($\varepsilon = 0.1$)	12.54	13.56	55.62
	Our ($\varepsilon = 1.0$)	**11.40**	**12.46**	55.62
	Our ($\varepsilon = 10$)	13.25	14.70	55.62

6 Conclusion

In conclusion, we proposed an efficient GPU-friendly SVD computation via the polar decomposition, utilizing an iterative scheme for the polar factor that is dominated by matrix multiplications. This structure allows for additional speedups by performing computations in mixed precision while retaining practical accuracy.

For the backward pass, we propose to differentiate the SVD through the stable differentiation of the polar decomposition and the symmetric eigenvalue decomposition. Finally, we provide a perturbation-based perspective that makes the backward computation robust for perturbed inputs and nearly degenerate spectra (including singular values close to zero), and show how regularization allows us to control this behavior.

AI USE STATEMENT

In this work, we used generative AI tools to assist with proving mathematical claims, including checking and developing intermediate derivations and proof steps. We did not use generative AI tools for other research ideation or execution tasks, and the generation of synthetic datasets was not applicable to this work. Additionally, we used generative AI tools for literature retrieval and discovery, drafting parts of the manuscript, and improving the clarity and readability of the writing.

All AI-assisted mathematical arguments and derivations were manually reviewed and checked by the authors for correctness. References and related work identified with the assistance of AI tools were verified against the original sources, and all AI-assisted text was reviewed and edited by the authors. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Hervé Abdi and Lynne J Williams. Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459, 2010.

Noah Amsel, David Persson, Christopher Musco, and Robert M. Gower. The polar express: Optimal matrix sign methods and their application to the muon algorithm. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=yRtgZ1K8hO.

Edward Anderson, Zhaojun Bai, Christian Bischof, L Susan Blackford, James Demmel, Jack Dongarra, Jeremy Du Croz, Anne Greenbaum, Sven Hammarling, Alan McKenney, et al. *LAPACK users' guide*. SIAM, 1999.

Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(153):1–43, 2018. URL http://jmlr.org/papers/v18/17-468.html.

Rajendra Bhatia. *Matrix analysis*, volume 169. Springer Science & Business Media, 2013.

Åke Björck and Clazett Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 8(2):358–364, 1971.

James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Yash Katariya, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/jax-ml/jax.

Jie Chen and Edmond Chow. A stable scaling of newton-schulz for improving the sign function computation of a hermitian matrix. *Preprint ANL/MCS-P5059-0114 (https://www.mcs.anl.gov/papers/P5059-0114.pdf)*, 2014.

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the fourth ACM conference on Recommender systems*, pp. 39–46, 2010.

P_P M_ de Rijk. A one-sided jacobi algorithm for computing the singular value decomposition on a vector computer. *SIAM journal on scientific and statistical computing*, 10(2):359–371, 1989.

James Demmel and William Kahan. Accurate singular values of bidiagonal matrices. *SIAM Journal on Scientific and Statistical Computing*, 11(5):873–912, 1990.

James W Demmel. *Applied numerical linear algebra*. SIAM, 1997.

Alan Edelman and Steven G Johnson. Matrix calculus for machine learning and beyond. *Massachusetts Institute of Technology: MIT OpenCourseWare*, 2022.

Evan S Gawlik and Melvin Leok. Iterative computation of the fréchet derivative of the polar decomposition. *SIAM Journal on Matrix Analysis and Applications*, 38(4):1354–1379, 2017.

Mike B. Giles. Collected matrix derivative results for forward and reverse mode algorithmic differentiation. In *Advances in Automatic Differentiation*, pp. 35–44. Springer Berlin Heidelberg, 2008.

Gene H Golub and Charles F Van Loan. *Matrix computations*. JHU press, 2013.

Andreas Griewank and Andrea Walther. *Evaluating derivatives: principles and techniques of algorithmic differentiation*. SIAM, 2008.

Ekaterina Grishina, Matvey Smirnov, and Maxim Rakhuba. Accelerating newton-schulz iteration for orthogonalization via chebyshev-type polynomials. *arXiv preprint arXiv:2506.10935*, 2025.

Per Christian Hansen. *Discrete inverse problems: insight and algorithms*. SIAM, 2010.

Magnus R Hestenes. Inversion of matrices by biorthogonalization and related results. *Journal of the Society for Industrial and Applied Mathematics*, 6(1):51–90, 1958.

Nicholas J Higham and Pythagoras Papadimitriou. A parallel algorithm for computing the polar decomposition. *Parallel computing*, 20(8):1161–1173, 1994.

Nick Higham. Faster svd via polar decomposition, 2015. URL https://nhigham.com/2015/11/24/faster-svd-via-polar-decomposition/.

Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. Matrix backpropagation for deep networks with structured layers. In *Proceedings of the IEEE international conference on computer vision*, pp. 2965–2973, 2015.

Ian T. Jolliffe and Jorge Cadima. Principal component analysis: a review and recent developments. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 374(2065):20150202, 04 2016. ISSN 1364-503X. doi: 10.1098/rsta.2015.0202. URL https://doi.org/10.1098/rsta.2015.0202.

Keller Jordan, Yuchen Jin, Vlado Boza, Jiacheng You, Franz Cesista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan.github.io/posts/muon/.

Rohit Sunil Kanchi and Sicheng He. Differentiable singular value decomposition (svd). *Mechanical Systems and Signal Processing*, 237:112817, 2025.

Zdislav Kovarik. Some iterative methods for improving orthonormality. *SIAM Journal on Numerical Analysis*, 7(3):386–389, 1970.

Fanxu Meng, Zhaohui Wang, and Muhan Zhang. Pissa: Principal singular values and singular vectors adaptation of large language models. *Advances in Neural Information Processing Systems*, 37: 121038–121072, 2024.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Yuji Nakatsukasa and Nicholas J Higham. Backward stability of iterations for computing the polar decomposition. *SIAM Journal on Matrix Analysis and Applications*, 33(2):460–479, 2012.

Yuji Nakatsukasa and Nicholas J Higham. Stable and efficient spectral divide and conquer algorithms for the symmetric eigenvalue decomposition and the svd. *SIAM Journal on Scientific Computing*, 35(3):A1325–A1349, 2013.

Yuji Nakatsukasa, Zhaojun Bai, and François Gygi. Optimizing halley's iteration for computing the matrix polar decomposition. *SIAM Journal on Matrix Analysis and Applications*, 31(5):2700–2720, 2010.

Vanni Noferini. A formula for the fréchet derivative of a generalized matrix function. *SIAM Journal on Matrix Analysis and Applications*, 38(2):434–457, 2017. doi: 10.1137/16M1072851. URL https://doi.org/10.1137/16M1072851.

NVIDIA. *cuSOLVER Library Reference Guide (PDF)*. NVIDIA, 2025. URL https://docs.nvidia.com/cuda/pdf/CUSOLVER_Library.pdf.

Ivan V Oseledets. Tensor-train decomposition. *SIAM Journal on Scientific Computing*, 33(5): 2295–2317, 2011.

Uliana Parkina and Maxim Rakhuba. Coala: Numerically stable and efficient framework for context-aware low-rank approximation. *Advances in Neural Information Processing Systems*, 39, 2025.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32, 2019.

Wang Qinsi, Jinghan Ke, Masayoshi Tomizuka, Kurt Keutzer, and Chenfeng Xu. Dobi-SVD: Differentiable SVD for LLM compression and some new perspectives. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=kws76i5XB8.

Keyvan Vahidy Rodpysh, Seyed Javad Mirabedini, and Touraj Banirostam. Employing singular value decomposition and similarity criteria for alleviating cold start and sparse data in context-aware recommender systems. *Electronic Commerce Research*, 23(2):681–707, 2023.

Zhenmei Shi, Yifei Ming, Ying Fan, Frederic Sala, and Yingyu Liang. Domain generalization via nuclear norm regularization. In *Conference on Parsimony and Learning*, pp. 179–201. PMLR, 2024.

Yue Song, Nicu Sebe, and Wei Wang. Fast differentiable matrix square root. In *International Conference on Learning Representations*, 2022.

Łukasz Struski, Paweł Morkisz, Przemysław Spurek, Samuel Rodriguez Bernabeu, and Tomasz Trzciński. Efficient gpu implementation of randomized svd and its applications. *Expert Systems with Applications*, 248:123462, 2024.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

James Townsend. Differentiating the singular value decomposition. Technical report, Technical Report 2016, https://j-towns.github.io/papers/svd-derivative.pdf, 2016.

Eugene E. Tyrtyshnikov. *Matrichnyi analiz i osnovy algebry [Matrix Analysis and Fundamentals of Algebra]*. MCNMO, 2025. [in Russian].

Nick Vannieuwenhoven, Raf Vandebril, and Karl Meerbergen. A new truncation strategy for the higher-order singular value decomposition. *SIAM Journal on Scientific Computing*, 34(2):A1027–A1052, 2012. doi: 10.1137/110836067. URL https://doi.org/10.1137/110836067.

Hongyi Wang, Scott Sievert, Shengchao Liu, Zachary Charles, Dimitris Papailiopoulos, and Stephen Wright. Atomo: Communication-efficient learning via atomic sparsification. *Advances in neural information processing systems*, 31, 2018.

Qilong Wang, Peihua Li, and Lei Zhang. G2denet: Global gaussian distribution embedding network and its application to visual recognition. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 6507–6516, 2017. doi: 10.1109/CVPR.2017.689.

Qilong Wang, Jiangtao Xie, Wangmeng Zuo, Lei Zhang, and Peihua Li. Deep cnns meet global covariance pooling: Better representation and generalization. *IEEE transactions on pattern analysis and machine intelligence*, 43(8):2582–2597, 2020.

Wei Wang, Zheng Dang, Yinlin Hu, Pascal Fua, and Mathieu Salzmann. Robust differentiable svd. *IEEE transactions on pattern analysis and machine intelligence*, 44(9):5472–5487, 2021.

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

Xiaofeng Yuan, Lixin Han, Subin Qian, Guoxia Xu, and Hong Yan. Singular value decomposition based recommendation using imputed data. *Knowledge-Based Systems*, 163:485–494, 2019.

Yinghao Zhang and Yue Hu. Differentiable svd based on moore-penrose pseudoinverse for inverse imaging problems. *arXiv preprint arXiv:2411.14141*, 2024.

A Implementation Details

A.1 Newton-Schulz Initialization Normalization

An important detail in implementing the Newton–Schulz iteration is that it converges only if $\|X_0\|_2 < \sqrt{3}$. Therefore, the initial matrix should be normalized before the iteration begins. Ideally, one would divide the matrix by its spectral norm so that $\|X_0\|_2 = 1$, but computing the spectral norm is expensive. The standard approach is to normalize the matrix using the Frobenius norm; here we use a normalization method based on the 1-norm and ∞-norm, as in the QDWH implementation in JAX[4]:

$$X_0 = M/\sqrt{\|M\|_1\|M\|_\infty}, \quad \text{since} \quad \|M\|_2 \le \sqrt{\|M\|_1\|M\|_\infty}.$$

The 1-norm and ∞-norm are straightforward to compute, as they correspond to the maximum ℓ_1 norm of the columns and rows of the matrix, respectively.

Algorithm 3 Normalization for Newton-Schulz based iterations

Require: Matrix $M \in \mathbb{R}^{m\times n}$ $(m \ge n)$
Ensure: Normalized matrix $\hat{M}$
1: $a \leftarrow \|M\|_1\|M\|_\infty$
2: $\hat{M} \leftarrow M/\sqrt{a}$
3: **return** $\hat{M}$

A.2 Handling Rank-Deficient Cases

Importantly, CANS and QDWH iterations do not converge to an orthogonal polar factor when applied to a rank-deficient matrix M with rank $r < n$. Instead, both iterations converge to a factor

$$W_r = U \begin{pmatrix} I_r & 0 \\ 0 & 0 \end{pmatrix} V^\top = (U_r \quad 0)\, V^\top,$$

where U and V are the left and right singular factors of M respectively. Therefore, the matrix W_rV does not have orthonormal columns, as required in the definition of the SVD. This issue can be resolved by orthogonalizing the matrix W_rV using a QR factorization (Nakatsukasa & Higham, 2013, Section 5.5). Importantly, the QR routine used in this procedure should return an R factor with non-negative diagonal entries. In practice, the rank of a matrix cannot be computed stably, so rank-deficient cases can be detected by checking the column norms of WV. See the experiments for rank-deficient matrices in Appendix B.2.

A.3 Matrix Generation Scheme

We generate a random matrix $M \in \mathbb{R}^{n\times m}$ with $\kappa_2(M) = c$ and $\mathrm{rank}(M) = r$ using the following scheme. First, we generate two matrices $G_1 \in \mathbb{R}^{n\times m}$ and $G_2 \in \mathbb{R}^{m\times m}$ with entries sampled independently from the standard normal distribution. Then, we perform thin QR factorizations:

$$G_1 = Q_1R_1, \quad G_2 = Q_2R_2.$$

[4] https://github.com/jax-ml/jax/blob/23db456a8acd01a04ed5a9f87f8265cc21703926/jax/_src/tpu/linalg/qdwh.py#L132

Finally, we obtain the matrix

$$M = Q_1 \Sigma Q_2,$$

where $\Sigma \in \mathbb{R}^{m \times m}$ is a diagonal matrix with diagonal entries $\Sigma_{ii} = c^{(r-i)/(r-1)}$ for $i \leq r$, and $\Sigma_{ii} = 0$ otherwise.

We generate a random matrix $M \in \mathbb{R}^{n \times n}$ with spectral gap Δ using the same scheme. First, we generate two matrices $G_1 \in \mathbb{R}^{n \times n}$ and $G_2 \in \mathbb{R}^{n \times n}$ with entries sampled independently from the standard normal distribution and perform QR factorizations. Finally, the matrix is obtained via:

$$M = Q_1 \Sigma Q_2,$$

where $\Sigma \in \mathbb{R}^{n \times n}$ is a diagonal matrix with diagonal entries $\Sigma_{ii} = i$ for $i < n$, and $\Sigma_{nn} = n - 1 + \Delta$.

A.4 Reproducibility Details

All experiments were conducted on an NVIDIA B200 GPU with 192 GB of VRAM, running Debian Linux 12. Each individual experiment took at most one hour, and the total compute budget did not exceed one GPU day.

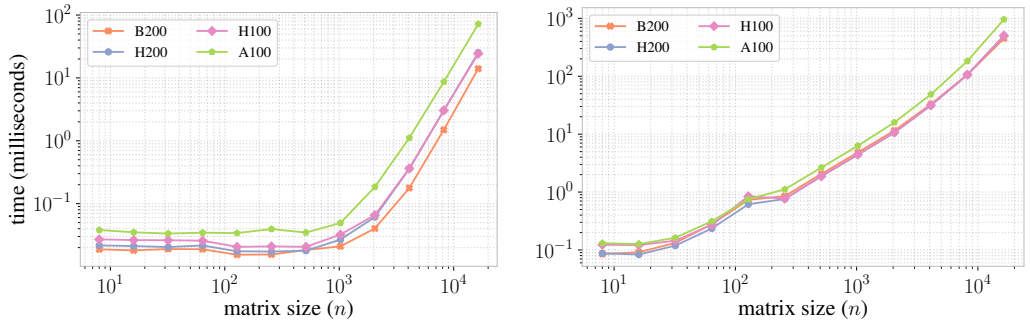
For the CANS iteration, we use a polynomial of degree 3 with a limit of 50 iterations. For preprocessing, we use 2 iterations, and the parameter δ is set to 0.99, as in the original implementation Grishina et al. (2025). As a stopping criterion, we use a tolerance of 10^{-5} for fp32 experiments and 10^{-3} for tf32 experiments.

For CANS SVD, we use $\varepsilon_{\text{QR}} = 10^{-5}$ for fp32 experiments and 10^{-3} for tf32 experiments to detect rank-deficient cases.

B Additional Experiments

B.1 Scaling QR Factorization and Matrix Multiplications on Modern GPUs

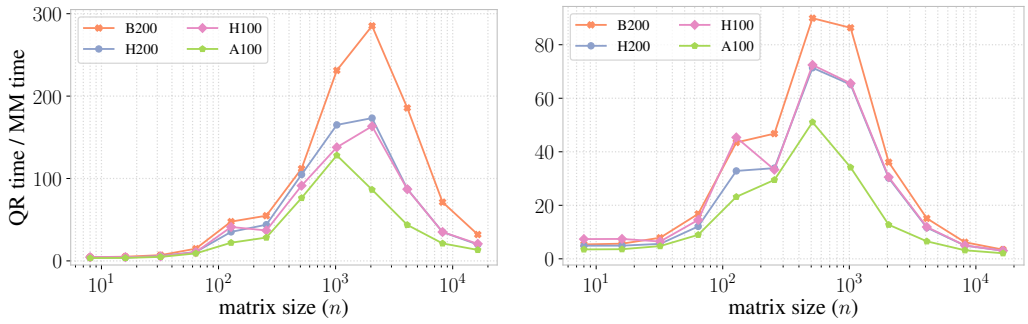
Recent advances in GPU engineering, driven by the growth of deep learning popularity, have made matrix multiplication operations very fast on GPUs. We empirically compare how the performance of matrix multiplication scales relative to QR decomposition. We provide a comparison across several NVIDIA GPU architectures: A100, H100, H200, and B200. As shown in Figure 3, QR factorization does not substantially speed up beyond the A100 architecture, whereas matrix multiplications in tf32 precision continue to speed up with newer generations of GPUs. Moreover, Figure 4 shows that matrix multiplications can be much faster than QR factorization across various matrix sizes. This performance gap becomes larger with newer GPU generations.



(a) Matrix multiplication computation in tf32 precision

(b) QR decomposition computation

Figure 3: Comparison of computation time for matrix multiplications in tf32 precision (left plot) and QR decompositions (right plot) across different GPUs on random square matrices of various sizes. The total wall-clock time is averaged over 100 trials.



(a) Matrix multiplications performed in tf32

(b) Matrix multiplications performed in fp32

Figure 4: Comparison illustrating how many matrix multiplications can be executed within the runtime of a single QR decomposition for square matrices of different sizes across different GPUs. In the left plot, matrix multiplications are performed in tf32 precision. In the right plot, matrix multiplications are performed in fp32 precision. The total wall-clock time is averaged over 100 trials.

B.2 Low Rank Matrices

We provide additional experiments comparing the performance and accuracy of different algorithms on low-rank square random matrices of varying sizes. Tables 5 and 6, as well as Figure 5, present the results. The CUDA POLAR algorithm failed to compute the SVD even for matrices with rank one less than full rank. The other methods demonstrate stability across various rank values. Notably, CANS SVD in fp32 precision runs slower than QDWH SVD on large matrices compared to the full-rank case, where they run in approximately the same time.

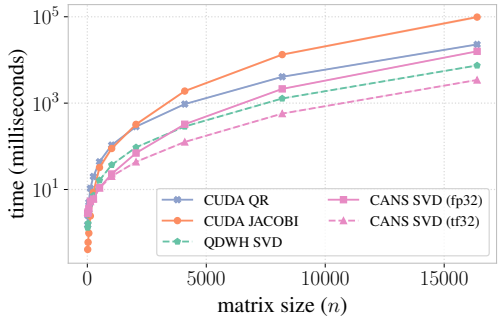


Figure 5: Comparison of SVD algorithms on a random low rank matrix $M \in \mathbb{R}^{n\times n}$ with $\mathrm{rank}(M) = \lfloor n/10 \rfloor$. Total wall-clock time was averaged over 100 trials and reported with the standard deviation. **Note:** we do not report CUDA POLAR for this setting, since it has substantially large reconstruction error. And algorithms have different accuracy, see Tables 5 and 6 for a more comprehensive analysis. The experiments are conducted on an NVIDIA B200 GPU.

B.3 Comparison with Additional Baselines

For a more comprehensive ablation study, we compare CANS SVD with additional baselines in terms of accuracy and performance. The additional baselines are listed below:

- **EVD of Gram Matrix.** A naive way of computing SVD via EVD of the Gram matrix $M^\top M$.
- **EVD of Extended Matrix.** A way of computing SVD via EVD of the extended matrix $\begin{pmatrix} 0 & M \\ M^\top & 0 \end{pmatrix}$.

Table 5: Comparison of SVD algorithms accuracy on a random square matrix $M \in \mathbb{R}^{4096\times4096}$ for different ranks. Reported relative reconstruction error $\|M - U\Sigma V^\top\|_F/\|M\|_F$, with standard deviation. Metrics were averaged over 100 sampled matrices. Red indicates cases with significantly large error.

Method	1	64	1024	4095	4096
CANS SVD (fp32)	$1.9_{\pm.0} \cdot 10^{-6}$	$2.7_{\pm.1} \cdot 10^{-6}$	$4.3_{\pm.0} \cdot 10^{-6}$	$8.4_{\pm.4} \cdot 10^{-6}$	$4.9_{\pm.0} \cdot 10^{-6}$
CANS SVD (tf32)	$5.3_{\pm.2} \cdot 10^{-4}$	$7.5_{\pm.2} \cdot 10^{-4}$	$1.1_{\pm.0} \cdot 10^{-3}$	$9.8_{\pm.0} \cdot 10^{-4}$	$9.7_{\pm.0} \cdot 10^{-4}$
QDWH SVD	$1.7_{\pm.1} \cdot 10^{-6}$	$1.9_{\pm.0} \cdot 10^{-6}$	$2.7_{\pm.0} \cdot 10^{-6}$	$3.5_{\pm.0} \cdot 10^{-6}$	$3.4_{\pm.0} \cdot 10^{-6}$
CUDA POLAR	$8.3_{\pm22.7} \cdot 10^{0}$	$7.6_{\pm.0} \cdot 10^{1}$	$7.6_{\pm.0} \cdot 10^{1}$	$7.6_{\pm.0} \cdot 10^{1}$	$4.3_{\pm.0} \cdot 10^{-6}$
CUDA QR	$7.4_{\pm.5} \cdot 10^{-7}$	$1.3_{\pm.2} \cdot 10^{-6}$	$6.6_{\pm.4} \cdot 10^{-6}$	$1.8_{\pm.1} \cdot 10^{-5}$	$1.8_{\pm.1} \cdot 10^{-5}$
CUDA JACOBI	$3.5_{\pm.6} \cdot 10^{-5}$	$9.5_{\pm.2} \cdot 10^{-5}$	$5.1_{\pm.4} \cdot 10^{-4}$	$1.0_{\pm.2} \cdot 10^{-4}$	$9.7_{\pm.4} \cdot 10^{-4}$

Table 6: Comparison of SVD algorithms accuracy on a random square matrix $M \in \mathbb{R}^{4096\times4096}$ for different ranks. Reported relative orthogonality $\max\{\|U^\top U - I\|_F/\|I\|_F, \|V^\top V - I\|_F/\|I\|_F\}$, with standard deviation. Metrics were averaged over 100 sampled matrices.

Method	1	64	1024	4095	4096
CANS SVD (fp32)	$1.6_{\pm.0} \cdot 10^{-6}$	$1.5_{\pm.0} \cdot 10^{-6}$	$1.8_{\pm.0} \cdot 10^{-6}$	$3.1_{\pm1.2} \cdot 10^{-6}$	$3.7_{\pm.0} \cdot 10^{-6}$
CANS SVD (tf32)	$1.9_{\pm.2} \cdot 10^{-6}$	$1.9_{\pm.0} \cdot 10^{-6}$	$2.2_{\pm.0} \cdot 10^{-6}$	$3.0_{\pm.0} \cdot 10^{-6}$	$6.6_{\pm.0} \cdot 10^{-4}$
QDWH SVD	$1.6_{\pm.0} \cdot 10^{-6}$	$1.5_{\pm.0} \cdot 10^{-6}$	$1.8_{\pm.0} \cdot 10^{-6}$	$3.6_{\pm.0} \cdot 10^{-6}$	$3.6_{\pm.0} \cdot 10^{-6}$
CUDA POLAR	$7.5_{\pm1.4} \cdot 10^{-6}$	$3.2_{\pm.0} \cdot 10^{-6}$	$6.9_{\pm.0} \cdot 10^{-6}$	$1.2_{\pm.0} \cdot 10^{-5}$	$4.1_{\pm.0} \cdot 10^{-6}$
CUDA QR	$7.9_{\pm.2} \cdot 10^{-6}$	$7.7_{\pm.2} \cdot 10^{-6}$	$6.4_{\pm.2} \cdot 10^{-6}$	$9.6_{\pm.3} \cdot 10^{-6}$	$9.6_{\pm.2} \cdot 10^{-6}$
CUDA JACOBI	$2.5_{\pm.2} \cdot 10^{-6}$	$3.8_{\pm.1} \cdot 10^{-4}$	$1.7_{\pm.1} \cdot 10^{-3}$	$2.1_{\pm.0} \cdot 10^{-3}$	$2.1_{\pm.0} \cdot 10^{-3}$

- **SciPy SVD.** Computing SVD on CPU using the SciPy routine to double-check the correctness of the proposed method.

We expect the Gram-based algorithm to lose accuracy due to its numerical instability (see Remark 1), while the extended-matrix-based algorithm should provide better accuracy but be much slower. The SciPy routine should be accurate, but it is computed on CPU and therefore is much slower, so we do not report its time.

Results are presented in Table 7 and Figure 6. As anticipated, the SciPy routine, Extended EVD, and CANS SVD in single precision have approximately the same accuracy, while the Gram-based algorithm starts losing accuracy even when $\kappa_2(M) = 100$ due to numerical instability and has a large reconstruction error for $\kappa_2(M) = 10^3$. Extended EVD is slower than the Gram-based method and CANS SVD.

Table 7: Comparison of the accuracy of SVD algorithms on a random square matrix $M \in \mathbb{R}^{4096\times4096}$ for different condition numbers. The reported metric is the relative reconstruction error $\|M - U\Sigma V^\top\|_F/\|M\|_F$, with standard deviation. Results are averaged over 100 sampled matrices.

Method	10	10^2	10^3	10^4	10^6
CANS SVD (fp32)	$4.9_{\pm.0} \cdot 10^{-6}$	$4.2_{\pm.0} \cdot 10^{-6}$	$4.3_{\pm.1} \cdot 10^{-6}$	$4.4_{\pm.0} \cdot 10^{-6}$	$5.3_{\pm.1} \cdot 10^{-6}$
Extended	$5.8_{\pm.0} \cdot 10^{-6}$	$5.6_{\pm.1} \cdot 10^{-6}$	$5.4_{\pm.1} \cdot 10^{-6}$	$1.1_{\pm.0} \cdot 10^{-5}$	$9.0_{\pm.0} \cdot 10^{-6}$
SciPy	$5.2_{\pm.0} \cdot 10^{-6}$	$5.1_{\pm.0} \cdot 10^{-6}$	$5.2_{\pm.0} \cdot 10^{-6}$	$5.2_{\pm.0} \cdot 10^{-6}$	$5.1_{\pm.0} \cdot 10^{-6}$
Gram	$4.4_{\pm.0} \cdot 10^{-6}$	$2.3_{\pm.0} \cdot 10^{-5}$	$1.1_{\pm.0} \cdot 10^{-4}$	$3.9_{\pm.0} \cdot 10^{-4}$	$6.9_{\pm.0} \cdot 10^{-4}$

B.4 Ablation on Iterations for Computing Polar Factor

In this section, we provide an ablation study of different methods for computing the polar factor that rely heavily on matrix multiplications. For a more difficult setting, we test all methods on a random square matrix $M \in \mathbb{R}^{1024\times1024}$ with a large condition number $\kappa_2(M) = 10^6$.

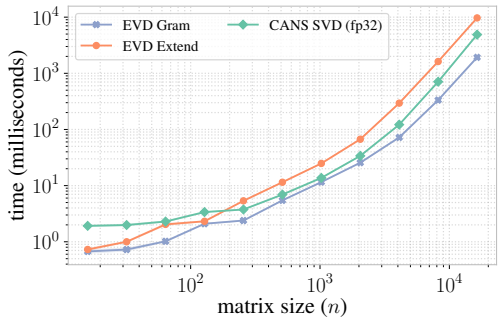


Figure 6: Comparison of SVD algorithms on a square random matrices $M \in \mathbb{R}^{n\times n}$ with $\kappa_2(M) = 10$. Total wall-clock time was averaged over 100 trials and reported with the standard deviation. The experiments are conducted on an NVIDIA B200 GPU.

Results are shown in Figure 7. For the Muon Newton–Schulz iteration Jordan et al. (2024) and Polar-Express Amsel et al. (2026), we use the default parameters provided in the authors implementations. As expected, the Muon iteration does not converge to an orthogonal matrix and yields a high relative error. The original Newton–Schulz iteration converges slowly, whereas Polar-Express and CANS converge substantially faster and exhibit similar convergence behavior. However, Polar-Express reaches a plateau at a higher error level than CANS and only drops to a comparable accuracy at the final iteration. This may be caused by the fact that the input to the polynomial is divided by 1.01 at all iterations except the final one. This behavior makes it difficult to use a stopping criterion for this method. CANS does not have this issue, and we therefore choose it as our main iteration.

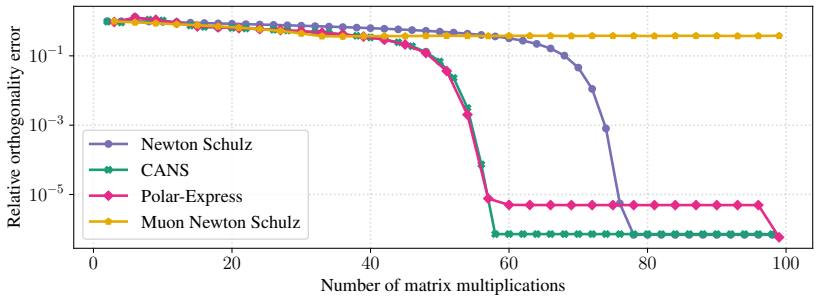


Figure 7: Comparison of different methods for computing polar decomposition on a square random matrices $M \in \mathbb{R}^{n\times n}$ with $\kappa_2(M) = 10^6$. The reported metric is relative orthogonality error $\|W^\top W - I\|_F / \|I\|_F$.

B.5 CUDA JACOBI HYPERPARAMETERS

According to the cuSOLVER documentation[5], CUDA JACOBI provides two hyperparameters controlling accuracy: `tolerance` and `max_sweeps`. We observe that the default value of the `max_sweeps` parameter is sufficient to reach the required precision, as stated in the documentation. Moreover, the number of sweeps does not change substantially across different settings.

We test different values of `tolerance` in Figure 8. Decreasing the `tolerance` parameter makes the algorithm more accurate but slower. For our experiments, we choose the default parameter value, which reaches approximately the same accuracy as CANS SVD in mixed precision. Making this algorithm more accurate would make it slower and allow it to outperform our method in fewer cases, making the comparison unfair.

[5] https://docs.nvidia.com/cuda/cusolver/index.html

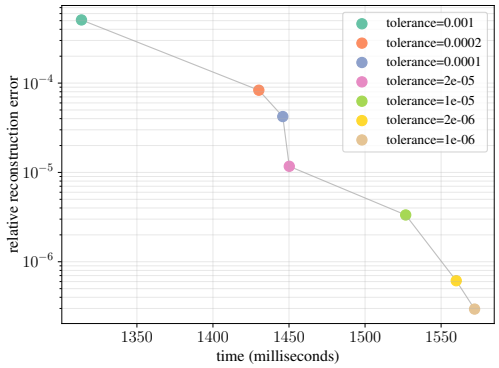


Figure 8: Pareto front for different CUDA JACOBI tolerance settings on square random matrices $M \in \mathbb{R}^{4096\times4096}$ with $\kappa_2(M) = 10^2$.

B.6 Choosing δ Hyperparameter

Here, we provide an ablation study on the hyperparameter δ in the CANS algorithm Grishina et al. (2025).

Let us briefly introduce how δ appears in CANS. In a nutshell, CANS algorithm divided into two steps: preprocessing step, and main CANS iteration. In the preprocessing stage, we apply a polynomial that finds the largest interval $[t, 1]$ whose values can be mapped into the interval $[1 - \delta, 1 + \delta]$. Then, CANS iteration is applied with parameters $a = 1 - \delta$ and $b = 1 + \delta$. For singular values inside the interval $[a, b]$, the iteration has accelerated convergence (see Corollary 3.5 of Grishina et al. (2025)). For singular values smaller than a, one can also show convergence, moreover, it is faster than the classical NS iteration.

In Tables 8 and 9, the reported relative reconstruction error depends on different δ values and condition numbers. The hyperparameter δ provides a tradeoff between accuracy and speed: increasing δ enlarges the interval of accelerated convergence, but also increases the stability constant, which results in reduced numerical accuracy. In our paper, we chose $\delta = 0.99$ based on empirical results across different arithmetic precisions. We could have made our method even more accurate by choosing a smaller value of δ. However, we argue that selected setting is superior as it allows us to achieve accuracy comparable to the methods used by default libraries, while running faster.

Table 8: Ablation on different δ for preprocessing on a random matrix $M \in \mathbb{R}^{4096\times4096}$ for different condition numbers. The reported metric is the relative reconstruction error of CANS SVD (fp32).

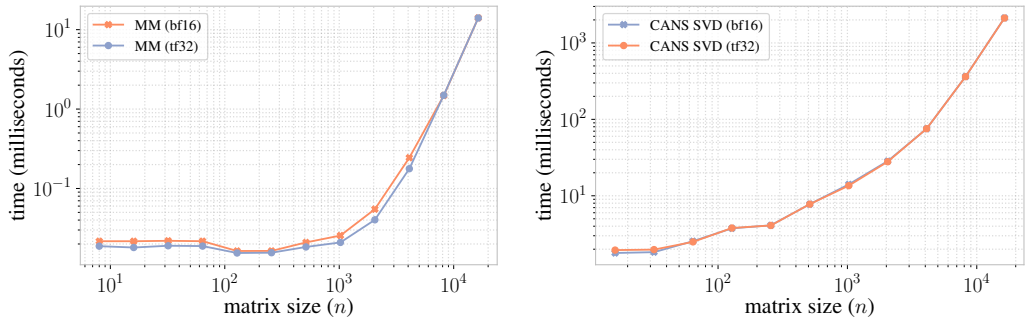
$\kappa_2(M) \downarrow$ $\delta \rightarrow$	0.9	0.99	0.999	0.9999
10^2	$3.30 \cdot 10^{-6}$	$4.26 \cdot 10^{-6}$	$7.25 \cdot 10^{-6}$	$1.84 \cdot 10^{-5}$
10^4	$3.77 \cdot 10^{-6}$	$4.41 \cdot 10^{-6}$	$1.05 \cdot 10^{-5}$	$2.51 \cdot 10^{-5}$
10^6	$4.00 \cdot 10^{-6}$	$5.38 \cdot 10^{-6}$	$1.69 \cdot 10^{-5}$	$4.88 \cdot 10^{-5}$

Table 9: Ablation on different δ for preprocessing on a random matrix $M \in \mathbb{R}^{4096\times4096}$ for different condition numbers. The reported metric is the relative reconstruction error of CANS SVD (tf32).

$\kappa_2(M) \downarrow$ $\delta \rightarrow$	0.9	0.99	0.999	0.9999
10	$7.20 \cdot 10^{-4}$	$9.73 \cdot 10^{-4}$	$2.98 \cdot 10^{-3}$	$9.15 \cdot 10^{-3}$
10^2	$6.57 \cdot 10^{-4}$	$8.24 \cdot 10^{-4}$	$1.84 \cdot 10^{-3}$	$4.33 \cdot 10^{-3}$
10^3	$7.04 \cdot 10^{-4}$	$8.52 \cdot 10^{-4}$	$1.91 \cdot 10^{-3}$	$3.63 \cdot 10^{-3}$

B.7 Difference Between TF32 and BF16 Matrix Multiplications

We compare matrix multiplication in bf16 and tf32 precision on an NVIDIA B200 GPU. Figure 9 presents the results of this comparison. In Figure 9a, the wall-clock time of the matrix multiplications themselves is compared; we observe a negligible speedup when using bf16 precision compared to tf32. In Figure 9b, the wall-clock time of the SVD computation with different precisions is shown, and the effect of using bf16 precision is almost invisible.



(a) Matrix multiplications time

(b) SVD computation time

Figure 9: Performance comparison of different precisions (tf32 and bf16) for matrix multiplication. The left plot shows the runtime of the matrix multiplications themselves. The right plot shows how the SVD runtime changes when the multiplication precision is varied. The total wall-clock time is averaged over 100 trials. Experiments are conducted on an NVIDIA B200 GPU.

B.8 Performance Comparison of Methods for Solving Symmetric Lyapunov Equation

Here, we conduct an ablation study of different methods for solving the symmetric Lyapunov equation

$$HX + XH = M,$$

where M is a symmetric matrix and H is symmetric positive semidefinite. We compare the iterative method proposed in Song et al. (2022) with a closed-form solution based on symmetric EVD (EIGH). Figure 10 summarizes the results. For small ($n \leq 64$) and large ($n > 1024$) matrices, the EIGH-based approach is faster than the iterative method. Moreover, as the matrix size n increases, the EIGH-based approach becomes increasingly more efficient. Additionally, we observe that the convergence of the iterative method depends on the condition number of the matrix, whereas the EIGH-based approach does not.

C SVD-inv Zhang & Hu (2024) inconsistency

The SVD-inv work Zhang & Hu (2024) contains a bug in Equation 25 in Theorem 3.1: the pseudoinverse of the matrix $\begin{pmatrix} \sigma & -\sigma \\ -\sigma & \sigma \end{pmatrix}$ is not equal to $\begin{pmatrix} 1/2\sigma & -1/2\sigma \\ 0 & 0 \end{pmatrix}$.

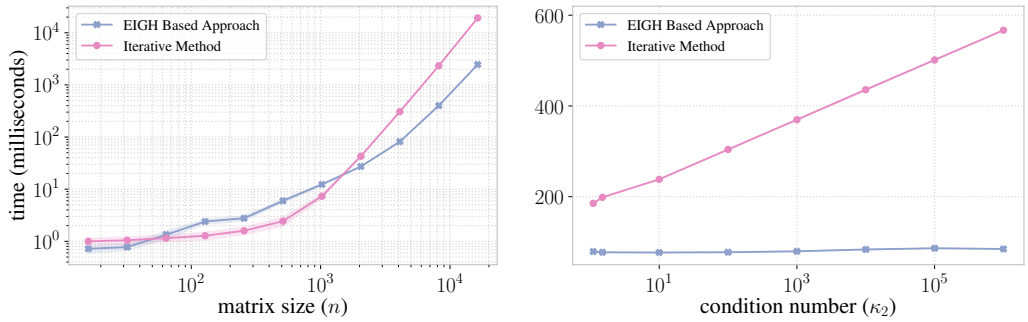
Example C.1 (SVD-inv Zhang & Hu (2024) inconsistency)**.** *Consider the SVD of the identity matrix, $I = U\Sigma V^\top$. At this degenerate point (all singular values are equal), the SVD-inv backward expression of Zhang & Hu (2024) exhibits inconsistency. Despite the fact that $U = V$ at this point, distinct gradients are returned during the backward pass even in symmetric cases. If $\overline{U} = \overline{\Sigma} = 0$, then the formula gives*

$$\overline{I} = 0.$$

If $\overline{V} = \overline{\Sigma} = 0$, then the same formula gives

$$\overline{I} = U(U^\top \overline{U})_{\textit{off-diag}} V^\top.$$

This inconsistency indicates that, at degenerate points, the SVD-inv backward pass may return gradients that do not correspond to the expected gradient of the underlying matrix-valued function.



(a) Comparison performed on random square matrices with $\kappa_2(M) = \kappa_2(H) = 100$ and varying size n.

(b) Comparison performed on random square matrices $M, H \in \mathbb{R}^{4096\times 4096}$ with varying condition number $\kappa_2(M) = \kappa_2(H)$.

Figure 10: Comparison of different methods for solving the symmetric Lyapunov equation under different scenarios. The total wall-clock time was averaged over 100 trials and reported with the standard deviation. The experiments were conducted on an NVIDIA B200 GPU.

Explanation for Example C.1. For the first case, setting $\overline{U} = \overline{\Sigma} = 0$ and using $VV^\top = I$ in the final formulas of (Zhang & Hu, 2024, Theorem 3.2) gives:

$$\begin{aligned}
\overline{I} &= U\,[\cancel{(F \odot [U^\top\overline{U} - \overline{U}^\top U])\Sigma} + \cancel{T \odot (U^\top\overline{U})}]V^\top \\
&\quad + \cancel{(I - UU^\top)\overline{U}\Sigma^{-1}V^\top} \\
&\quad + \cancel{U(I \odot \overline{\Sigma})V^\top} \\
&\quad + U\Sigma(F \odot [V^\top\overline{V} - \overline{V}^\top V])V^\top \\
&\quad + U\Sigma^{-1}\overline{V}^\top\cancel{(I - VV^\top)} \\
&= U\Sigma(F \odot [V^\top\overline{V} - \overline{V}^\top V])V^\top = 0,
\end{aligned}$$

since all singular values of the identity matrix are equal and thus $F = 0$.

For the second case, setting $\overline{V} = \overline{\Sigma} = 0$, using $UU^\top = I$, and setting $F = 0$ gives:

$$\begin{aligned}
\overline{I} &= U\,[\cancel{(F \odot [U^\top\overline{U} - \overline{U}^\top U])\Sigma} + T \odot (U^\top\overline{U})]V^\top \\
&\quad + \cancel{(I - UU^\top)\overline{U}\Sigma^{-1}V^\top} \\
&\quad + \cancel{U(I \odot \overline{\Sigma})V^\top} \\
&\quad + \cancel{U\Sigma(F \odot [V^\top\overline{V} - \overline{V}^\top V])V^\top} \\
&\quad + \cancel{U\Sigma^{-1}\overline{V}^\top(I - VV^\top)} \\
&= U(T \odot (U^\top\overline{U}))V^\top = U(U^\top\overline{U})_{\text{off-diag}}V^\top,
\end{aligned}$$

since all singular values of I are equal and therefore all elements in T are set to 1 except the diagonal elements, which are set to 0. □

D Proofs for Polar Derivatives

In this section, we present formulas and their proofs for differentiating the polar decomposition, which is used in our method.

Before proving the main formulas, we state the following useful lemma.

Lemma D.1. *Let $H \in \mathbb{R}^{n\times n}$ be a symmetric positive semi-definite matrix. Let Y be a matrix satisfying the Lyapunov equation*

$$HY + YH = A, \tag{2}$$

where $A \in \mathbb{R}^{n\times n}$ is an arbitrary matrix. Let X be a solution of

$$HX + XH = B, \tag{3}$$

where $B \in \mathbb{R}^{n\times n}$ is an arbitrary matrix. Then the following identity holds:

$$\langle A, X\rangle = \langle Y, B\rangle. \tag{4}$$

Proof. We compute

$$\langle A, X\rangle = \langle HY + YH, X\rangle = \langle HX, Y\rangle + \langle XH, Y\rangle = \langle HX + XH, Y\rangle = \langle B, Y\rangle,$$

which proves the claim. □

Proposition D.2. *Let $M \in \mathbb{R}^{m\times n}$ $(m \geq n)$ be a full-rank matrix with the polar decomposition*

$$M = WH.$$

Let $\mathcal{L}(M) = \ell(W, H)$ be a scalar loss function, and assume that the partial derivatives

$$\overline{W} := \frac{\partial \ell}{\partial W}, \qquad \overline{H} := \frac{\partial \ell}{\partial H}$$

are given. Then the gradient of $\mathcal{L}$ with respect to M is

$$\frac{\partial \mathcal{L}}{\partial M} = \overline{W}H^{-1} + 2MX,$$

where X is the symmetric solution of the Lyapunov equation

$$HX + XH = (\overline{H} - W^\top \overline{W} H^{-1})_{\mathrm{sym}}.$$

Proof. Since M is full rank, the polar factor H is invertible. Moreover, because $H = H^\top$, its differential satisfies

$$\mathrm{d}H = \mathrm{d}H^\top,$$

i.e., $\mathrm{d}H$ is symmetric.

The differential of the polar factorization $M = WH$ is given by

$$\mathrm{d}M = \mathrm{d}W\, H + W\, \mathrm{d}H. \tag{5}$$

Since H is nonsingular, this relation can be rearranged as

$$\mathrm{d}W = (\mathrm{d}M - W\, \mathrm{d}H)H^{-1}. \tag{6}$$

The total differential of the loss function is

$$\mathrm{d}\mathcal{L} = \langle \overline{W}, \mathrm{d}W\rangle + \langle \overline{H}, \mathrm{d}H\rangle. \tag{7}$$

Substituting equation 6 into equation 7 yields

$$\mathrm{d}\mathcal{L} = \left\langle \overline{W},\, (\mathrm{d}M - W\, \mathrm{d}H)H^{-1} \right\rangle + \left\langle \overline{H},\, \mathrm{d}H \right\rangle,$$

which can be rewritten as

$$\mathrm{d}\mathcal{L} = \left\langle \overline{W}H^{-1},\, \mathrm{d}M \right\rangle - \left\langle W^\top \overline{W} H^{-1},\, \mathrm{d}H \right\rangle + \left\langle \overline{H},\, \mathrm{d}H \right\rangle.$$

Therefore,

$$\mathrm{d}\mathcal{L} = \left\langle \overline{W}H^{-1}, \mathrm{d}M \right\rangle + \left\langle \overline{H} - W^\top \overline{W} H^{-1}, \mathrm{d}H \right\rangle. \tag{8}$$

Since $H^2 = M^\top M$, differentiation gives the Lyapunov equation

$$H\, \mathrm{d}H + \mathrm{d}H\, H = \mathrm{d}M^\top M + M^\top \mathrm{d}M. \tag{9}$$

Let X denote the symmetric solution of

$$HX + XH = (\overline{H} - W^\top \overline{W} H^{-1})_{\mathrm{sym}}. \tag{10}$$

Applying Lemma D.1 to equation 9 and equation 10, we obtain

$$\left\langle \overline{H} - W^\top \overline{W} H^{-1}, \mathrm{d}H \right\rangle = \left\langle (\overline{H} - W^\top \overline{W} H^{-1})_{\mathrm{sym}}, \mathrm{d}H \right\rangle = \langle X, \mathrm{d}M^\top M + M^\top \mathrm{d}M\rangle,$$

which simplifies to

$$\left\langle \overline{H} - W^\top \overline{W} H^{-1}, \mathrm{d}H \right\rangle = 2\langle MX, \mathrm{d}M\rangle. \tag{11}$$

Combining equation 8 and equation 11, we finally obtain

$$\mathrm{d}\mathcal{L} = \left\langle \overline{W}H^{-1} + 2MX, \mathrm{d}M \right\rangle,$$

which proves the stated gradient formula. □

So, the task of computing the derivative of the polar decomposition reduces to solving a Lyapunov equation. The following proposition describes how this equation can be solved.

Proposition D.3 (Solving the Symmetric Lyapunov Equation)**.** *Let $H \in \mathbb{R}^{n\times n}$ be a symmetric positive definite matrix with eigenvalue decomposition*

$$H = Q\Lambda Q^\top,$$

where Q is orthogonal and $\Lambda = \operatorname{diag}(\lambda_1, \dots, \lambda_n)$ with $\lambda_i > 0$. For a given matrix C, the unique solution X of the Lyapunov equation

$$HX + XH = C$$

is given by

$$X = Q\widetilde{X}Q^\top, \qquad \widetilde{X}_{ij} = \frac{(Q^\top CQ)_{ij}}{\lambda_i + \lambda_j}.$$

Proof. In book Golub & Van Loan (2013), p. 226. □

Remark 3. *If the Eigenvalue Decomposition of the polar factor H is available, solving the Lyapunov equation in the backward pass reduces to simple elementwise operations in the eigenvectors of H. As a result, computing the gradient is almost free compared to the forward pass and does not require additional matrix factorizations. Moreover, this computation does not depend on spectral gaps, since all terms involve only sums $\lambda_i + \lambda_j$.*

Nevertheless, this equation can also be solved using alternative approaches; for example, the work Song et al. (2022) proposes an iterative method. However, in our setting we explicitly seek the SVD via polar decomposition, and therefore the corresponding EVD is already available, which allows us to solve fast this equation.

Proof for Theorem 5.1. Since M has full column rank, H is symmetric positive definite. By Proposition D.2,

$$\overline{M} = \overline{W}H^{-1} + 2MX,$$

where X is the unique symmetric solution of

$$HX + XH = C.$$

Applying Proposition D.3 to $H = V\Sigma V^\top$ gives

$$X = V\Big(T \odot (V^\top CV)\Big)V^\top, \qquad T_{ij} = \frac{1}{\sigma_i + \sigma_j}.$$

Substituting this expression for X into the formula above proves the result. □

E Regularized Gradient For Polar Decomposition

Proposition E.1 (Tikhonov Regularization for Linear Equations, (Tyrtyshnikov, 2025, Sec. 18.16), (Hansen, 2010, Sec. 4.5))**.** *Let $A \in \mathbb{R}^{m\times n}$, $b \in \mathbb{R}^m$, and suppose that*

$$\|A - A_\varepsilon\| \le \varepsilon, \qquad \|b - b_\varepsilon\| \le \varepsilon.$$

Let $x_0 = A^+ b$ be the minimum-norm least-squares solution of $Ax = b$, and, for $\alpha > 0$, let

$$x_{\varepsilon,\alpha} = (A_\varepsilon^\top A_\varepsilon + \alpha I)^{-1} A_\varepsilon^\top b_\varepsilon.$$

Then, for $0 < \varepsilon \le 1$ and $\alpha = \sqrt{\varepsilon}$,

$$\|x_{\varepsilon,\alpha} - x_0\| \le c\sqrt{\varepsilon},$$

where $c = c(A, b) > 0$ is independent of ε.

Lemma E.2. *For any symmetric $A, B \in \mathbb{R}^{N\times N}$,*

$$\big\||A| - |B|\big\|_F \;\le\; \|A - B\|_F, \qquad |A| := (A^\top A)^{1/2}.$$

Proof. In book (Bhatia, 2013, Eq. (X.24)). □

Proof of Proposition 5.2. Since M and $\widetilde{M}$ have full column rank,

$$M = WH, \qquad \widetilde{M} = \widetilde{W}\widetilde{H}, \qquad H, \widetilde{H} \succ 0.$$

Throughout the proof, $c > 0$ may depend on $M, \overline{W}, \overline{H}$, but not on ε. By smoothness of the polar decomposition in the full-rank regime and Lemma E.2,

$$\|H - \widetilde{H}\|_F \le c\varepsilon. \tag{12}$$

The matrix H^{-1} is the solution of $HZ = I$, whereas $\widetilde{H}_\varepsilon^{-1}$ is the Tikhonov solution of the perturbed system:

$$(\widetilde{H}^2 + \varepsilon^{1/2} I)\widetilde{H}_\varepsilon^{-1} = \widetilde{H}.$$

Applying Proposition E.1 columnwise gives

$$\|\widetilde{H}_\varepsilon^{-1} - H^{-1}\|_F \le c\varepsilon^{1/2}. \tag{13}$$

In particular, $\|\widetilde{H}_\varepsilon^{-1}\|_2$ is bounded for all sufficiently small ε.

Define

$$B = \left(\overline{H} - H^{-1}M^\top \overline{W} H^{-1}\right)_{\mathrm{sym}}.$$

Using the definition of $\widetilde{B}_\varepsilon$, we add and subtract intermediate products to obtain

$$\begin{aligned}
&\widetilde{H}_\varepsilon^{-1}\widetilde{M}^\top \overline{W}\widetilde{H}_\varepsilon^{-1} - H^{-1}M^\top \overline{W} H^{-1} \\
&= (\widetilde{H}_\varepsilon^{-1} - H^{-1})\widetilde{M}^\top \overline{W}\widetilde{H}_\varepsilon^{-1} \\
&\quad + H^{-1}(\widetilde{M}^\top - M^\top)\overline{W}\widetilde{H}_\varepsilon^{-1} \\
&\quad + H^{-1}M^\top \overline{W}(\widetilde{H}_\varepsilon^{-1} - H^{-1}).
\end{aligned}$$

Since symmetrization does not increase the Frobenius norm, equation 13 and $\|\widetilde{M} - M\|_F \le \varepsilon$ imply

$$\|\widetilde{B}_\varepsilon - B\|_F \le c\left(\|\widetilde{H}_\varepsilon^{-1} - H^{-1}\|_F + \|\widetilde{M} - M\|_F\right) \le c\varepsilon^{1/2}. \tag{14}$$

Let X be the solution of

$$HX + XH = B.$$

After vectorization, this equation becomes

$$\mathcal{A}\,\mathrm{vec}(X) = \mathrm{vec}(B), \qquad \mathcal{A} := I \otimes H + H \otimes I.$$

Similarly, let

$$\widetilde{\mathcal{A}} := I \otimes \widetilde{H} + \widetilde{H} \otimes I.$$

By equation 12 and equation 14,

$$\|\mathcal{A} - \widetilde{\mathcal{A}}\| = \mathcal{O}(\varepsilon), \qquad \|\mathrm{vec}(B) - \mathrm{vec}(\widetilde{B}_\varepsilon)\| = \mathcal{O}(\varepsilon^{1/2}).$$

Thus, the effective perturbation level of the Lyapunov system is $\mathcal{O}(\varepsilon^{1/2})$. The matrix $\widetilde{X}_\varepsilon$ defined in the statement is precisely the Tikhonov solution

$$\left(\widetilde{\mathcal{A}}^\top \widetilde{\mathcal{A}} + \varepsilon^{1/4} I\right)\mathrm{vec}(\widetilde{X}_\varepsilon) = \widetilde{\mathcal{A}}^\top \mathrm{vec}(\widetilde{B}_\varepsilon).$$

Therefore, Proposition E.1 yields

$$\|\widetilde{X}_\varepsilon - X\|_F \le c\varepsilon^{1/4}. \tag{15}$$

By Theorem 5.1,

$$\overline{M} = \overline{W}H^{-1} + 2MX,$$

whereas

$$\overline{\widetilde{M}}_\varepsilon = \overline{W}\widetilde{H}_\varepsilon^{-1} + 2\widetilde{M}\widetilde{X}_\varepsilon.$$

Consequently,

$$\begin{aligned}
\|\overline{M} - \overline{\widetilde{M}}_\varepsilon\|_F \le{}& \|\overline{W}(H^{-1} - \widetilde{H}_\varepsilon^{-1})\|_F + 2\|M(X - \widetilde{X}_\varepsilon)\|_F \\
&+ 2\|(M - \widetilde{M})\widetilde{X}_\varepsilon\|_F.
\end{aligned}$$

By equation 13 and equation 15, the matrices $\widetilde{X}_\varepsilon$ remain bounded for sufficiently small ε. Hence,

$$\|\overline{M} - \overline{\widetilde{M}}_\varepsilon\|_F \le c\left(\varepsilon^{1/2} + \varepsilon^{1/4} + \varepsilon\right) \le c\varepsilon^{1/4},$$

which proves the claim. □

F EIGENVALUE DECOMPOSITION DERIVATIVES

Proposition F.1 (Existing gradient of EVD, from Giles (2008)). *Let $M \in \mathbb{R}^{n\times n}$ be a symmetric matrix with EVD*

$$M = V\Sigma V^\top,$$

where V is orthogonal and $\Sigma = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$. Given a scalar loss function $L = L(V, \Sigma)$, the gradient of L with respect to M is given by

$$\frac{\partial L}{\partial M} = V\left(K^\top \odot \left(V^\top \frac{\partial L}{\partial V}\right) + \frac{\partial L}{\partial \Sigma}\right) V^\top.$$

The matrix $K \in \mathbb{R}^{n\times n}$ is defined elementwise as

$$K_{ij} = \begin{cases} \dfrac{1}{\sigma_i - \sigma_j}, & i \neq j, \\ 0, & i = j. \end{cases} \tag{16}$$

Proof of Theorem 5.4. We solve the minimization problem in Definition 5.3. For the factor perturbations $dV = V\Omega$ and $d\Sigma = D$,

$$dM = dV\,\Sigma V^\top + V\,d\Sigma\,V^\top + V\Sigma\,dV^\top = V(\Omega\Sigma - \Sigma\Omega + D)V^\top.$$

For a symmetric candidate G, set

$$H := V^\top G V, \qquad B := \frac{1}{2}\big(V^\top \overline{V} - \overline{V}^\top V\big).$$

Then $H^\top = H$ and $\|G\|_F = \|H\|_F$. Moreover,

$$\langle G, dM\rangle_F = \langle H\Sigma - \Sigma H, \Omega\rangle_F + \langle H, D\rangle_F,$$
$$\langle \overline{V}, V\Omega\rangle_F = \langle B, \Omega\rangle_F,$$

where the second identity follows from $\Omega^\top = -\Omega$. Thus, the constraints in the definition are equivalent to

$$\langle H\Sigma - \Sigma H - B, \Omega\rangle_F + \langle H - \overline{\Sigma}, D\rangle_F = 0$$

for every diagonal D and every skew-symmetric Ω with $\Omega_{ij} = 0$ whenever $\sigma_i = \sigma_j$.

Taking $\Omega = 0$ and varying D gives

$$H_{ii} = \overline{\Sigma}_{ii}, \qquad 1 \le i \le n.$$

Taking $D = 0$ and varying the permitted entries of Ω gives

$$(\sigma_j - \sigma_i)H_{ij} = B_{ij} \qquad \text{whenever } \sigma_i \neq \sigma_j.$$

Consequently, the constraints are equivalent to

$$H_{ii} = \overline{\Sigma}_{ii}, \qquad H_{ij} = \frac{B_{ij}}{\sigma_j - \sigma_i} \quad \text{if } \sigma_i \neq \sigma_j,$$

with no restrictions on H_{ij} when $i \neq j$ and $\sigma_i = \sigma_j$. The prescribed entries are consistent with symmetry because $B^\top = -B$.

Since

$$\|G\|_F^2 = \|H\|_F^2 = \sum_i H_{ii}^2 + 2\sum_{i<j} H_{ij}^2,$$

the unique minimum-norm solution is obtained by setting all unrestricted entries to zero. Hence

$$H_\star = K^\top \odot B + \overline{\Sigma} = S + \overline{\Sigma},$$

and therefore

$$\overline{M}_\star = V H_\star V^\top = V(S + \overline{\Sigma})V^\top,$$

as claimed. □

G Existing formulas for SVD Derivatives

Theorem G.1 (Backward differentiation of low-rank SVD)**.** *Let $A \in \mathbb{R}^{m\times n}$ admit a rank-k singular value decomposition*

$$A = USV^\top, \qquad U \in \mathbb{R}^{m\times k},\ S = \operatorname{diag}(s_1,\dots,s_k),\ V \in \mathbb{R}^{n\times k},$$

with $U^\top U = V^\top V = I_k$ and $s_i > 0$ pairwise distinct.

Let $\overline{U}, \overline{S}, \overline{V}$ denote gradients corresponding to U, S, V, respectively. Define the matrix $F \in \mathbb{R}^{k\times k}$ as

$$F_{ij} = \begin{cases} \dfrac{1}{s_j^2 - s_i^2}, & i \neq j, \\ 0, & i = j. \end{cases} \tag{17}$$

Then the gradient of a scalar loss $\mathcal{L}$ with respect to A is given by

$$\begin{aligned} \overline{A} = &\Big[U\big(F \odot (U^\top\overline{U} - \overline{U}^\top U)\big)S + (I_m - UU^\top)\,\overline{U}\,S^{-1}\Big]V^\top \\ &+ U\big(I_k \odot \overline{S}\big)V^\top \\ &+ U\Big[S\big(F \odot (V^\top\overline{V} - \overline{V}^\top V)\big)V^\top + S^{-1}\,\overline{V}^\top(I_n - VV^\top)\Big]. \end{aligned}$$

Proof. In work Townsend (2016). □

Corollary G.2 (Full-rank SVD backward)**.** *Let $m \geq n$. If $k = n$, then the orthogonal complements vanish,*

$$I_n - VV^\top = 0,$$

and the backward formula simplifies to

$$\overline{A} = U\big(F \odot (U^\top\overline{U} - \overline{U}^\top U)\big)SV^\top - (I_m - UU^\top)\overline{U}S^{-1}V^\top \tag{18}$$

$$+ U\big(I_k \odot \overline{S}\big)V^\top + US\big(F \odot (V^\top\overline{V} - \overline{V}^\top V)\big)V^\top. \tag{19}$$

H Backward differentiation of SVD in degenerate case

Theorem H.1 (SVD backward formula in the degenerate case)**.** *Let*

$$M = U\Sigma V^\top, \qquad \Sigma = \operatorname{diag}(\sigma_1,\dots,\sigma_n),$$

where equal singular values may have multiplicities. Define

$$A := U^\top\overline{U}, \qquad B := V^\top\overline{V}.$$

Introduce the matrices $F, G \in \mathbb{R}^{n\times n}$ by

$$F_{ij} = \begin{cases} \dfrac{1}{\sigma_j^2 - \sigma_i^2}, & \sigma_i \neq \sigma_j, \\ 0, & \sigma_i = \sigma_j, \end{cases}$$

and

$$G_{ij} = \begin{cases} \dfrac{1}{2\sigma_i}, & \sigma_i = \sigma_j > 0, \\ 0, & \sigma_i \neq \sigma_j \text{ or } \sigma_i = \sigma_j = 0. \end{cases}$$

Then

$$\overline{M} = U\left[F \odot \Big((A - A^\top)\Sigma + \Sigma(B - B^\top)\Big) + \overline{\Sigma} + G \odot \operatorname{skew}(A - B)\right]V^\top,$$

Lemma H.2. *Let $M \in \mathbb{R}^{n\times n}$ have a full singular value decomposition*

$$M = U\Sigma V^\top,$$

where $U, V \in \mathbb{R}^{n\times n}$ are orthogonal. Assume that the singular values are grouped into distinct blocks:

$$U = [U_1 \ \cdots \ U_k], \qquad V = [V_1 \ \cdots \ V_k],$$

and

$$\Sigma = \operatorname{diag}(\sigma_1 I_{r_1}, \ldots, \sigma_k I_{r_k}), \qquad \sigma_i \neq \sigma_j \quad (i \neq j),$$

Then the adjoint with respect to M has the form

$$\overline{M} = U R V^\top,$$

where R is the block matrix with blocks $R_{ij} \in \mathbb{R}^{r_i \times r_j}$ defined as follows. For $i \neq j$,

$$R_{ij} = \frac{\sigma_j \left(U_i^\top \overline{U}_j - \overline{U}_i^\top U_j\right) + \sigma_i \left(V_i^\top \overline{V}_j - \overline{V}_i^\top V_j\right)}{\sigma_j^2 - \sigma_i^2}, \qquad i \neq j.$$

For the diagonal blocks with $\sigma_i > 0$,

$$R_{ii} = \overline{\Sigma}_{ii} + \frac{1}{2\sigma_i} \operatorname{skew}\left(U_i^\top \overline{U}_i - V_i^\top \overline{V}_i\right).$$

For a block with $\sigma_i = 0$, we take

$$R_{ii} = \overline{\Sigma}_{ii}.$$

Proof. Let

$$C := U^\top \mathrm{d}M V, \qquad \Omega_U := U^\top \mathrm{d}U, \qquad \Omega_V := V^\top \mathrm{d}V.$$

Since U and V are orthogonal, we have

$$\Omega_U^\top = -\Omega_U, \qquad \Omega_V^\top = -\Omega_V.$$

Differentiating

$$M = U\Sigma V^\top$$

gives

$$\mathrm{d}M = \mathrm{d}U\, \Sigma V^\top + U\, \mathrm{d}\Sigma\, V^\top + U\Sigma\, \mathrm{d}V^\top.$$

Multiplying by $U^\top$ from the left and by V from the right, we obtain

$$C = \Omega_U \Sigma + \mathrm{d}\Sigma - \Sigma \Omega_V.$$

In block form this reads

$$C_{ij} = \sigma_j (\Omega_U)_{ij} - \sigma_i (\Omega_V)_{ij} + (\mathrm{d}\Sigma)_{ij}.$$

For $i \neq j$, the block $(\mathrm{d}\Sigma)_{ij}$ vanishes. Therefore,

$$C_{ij} = \sigma_j (\Omega_U)_{ij} - \sigma_i (\Omega_V)_{ij}.$$

Using the skew-symmetry of Ω_U and Ω_V, the transposed block equation is

$$C_{ji}^\top = -\sigma_i (\Omega_U)_{ij} + \sigma_j (\Omega_V)_{ij}.$$

Thus, for each off-diagonal pair $i \neq j$,

$$\begin{pmatrix} C_{ij} \\ C_{ji}^\top \end{pmatrix} = \begin{pmatrix} \sigma_j I & -\sigma_i I \\ -\sigma_i I & \sigma_j I \end{pmatrix} \begin{pmatrix} (\Omega_U)_{ij} \\ (\Omega_V)_{ij} \end{pmatrix}.$$

Since $\sigma_i \neq \sigma_j$, this system is nonsingular, and hence

$$(\Omega_U)_{ij} = \frac{\sigma_j C_{ij} + \sigma_i C_{ji}^\top}{\sigma_j^2 - \sigma_i^2},$$

$$(\Omega_V)_{ij} = \frac{\sigma_i C_{ij} + \sigma_j C_{ji}^\top}{\sigma_j^2 - \sigma_i^2}.$$

Now compute the differential of the loss:

$$\mathrm{d}\mathcal{L} = \langle \overline{U}, \mathrm{d}U \rangle + \langle \overline{\Sigma}, \mathrm{d}\Sigma \rangle + \langle \overline{V}, \mathrm{d}V \rangle.$$

Define

$$A := U^\top \overline{U}, \qquad B := V^\top \overline{V}.$$

The contributions from the rotations inside the U- and V-coordinates are

$$\langle \overline{U}, \mathrm{d}U \rangle = \langle U^\top \overline{U}, \Omega_U \rangle = \langle A, \Omega_U \rangle,$$

and

$$\langle \overline{V}, \mathrm{d}V \rangle = \langle V^\top \overline{V}, \Omega_V \rangle = \langle B, \Omega_V \rangle.$$

Fix $i \neq j$. Since Ω_U is block skew-symmetric,

$$(\Omega_U)_{ji} = -(\Omega_U)_{ij}^\top.$$

Therefore the two blocks (i, j) and (j, i) contribute

$$\langle A_{ij}, (\Omega_U)_{ij} \rangle + \langle A_{ji}, (\Omega_U)_{ji} \rangle = \left\langle A_{ij} - A_{ji}^\top, (\Omega_U)_{ij} \right\rangle.$$

Similarly,

$$\langle B_{ij}, (\Omega_V)_{ij} \rangle + \langle B_{ji}, (\Omega_V)_{ji} \rangle = \left\langle B_{ij} - B_{ji}^\top, (\Omega_V)_{ij} \right\rangle.$$

Substituting the formulas for $(\Omega_U)_{ij}$ and $(\Omega_V)_{ij}$, we get

$$\begin{aligned} &\left\langle A_{ij} - A_{ji}^\top, (\Omega_U)_{ij} \right\rangle + \left\langle B_{ij} - B_{ji}^\top, (\Omega_V)_{ij} \right\rangle \\ &= \left\langle A_{ij} - A_{ji}^\top, \frac{\sigma_j C_{ij} + \sigma_i C_{ji}^\top}{\sigma_j^2 - \sigma_i^2} \right\rangle + \left\langle B_{ij} - B_{ji}^\top, \frac{\sigma_i C_{ij} + \sigma_j C_{ji}^\top}{\sigma_j^2 - \sigma_i^2} \right\rangle. \end{aligned}$$

Collecting the coefficient of C_{ij}, we obtain

$$R_{ij} = \frac{\sigma_j (A_{ij} - A_{ji}^\top) + \sigma_i (B_{ij} - B_{ji}^\top)}{\sigma_j^2 - \sigma_i^2}, \qquad i \neq j.$$

It remains to identify the diagonal block. For the i-th diagonal block,

$$C_{ii} = \sigma_i \left((\Omega_U)_{ii} - (\Omega_V)_{ii} \right) + \mathrm{d}\sigma_i I_{r_i}.$$

Set

$$Z_i := (\Omega_U)_{ii} - (\Omega_V)_{ii}.$$

Since both $(\Omega_U)_{ii}$ and $(\Omega_V)_{ii}$ are skew-symmetric, Z_i is skew-symmetric. Hence

$$C_{ii} = (\mathrm{d}\Sigma)_{ii} + \sigma_i Z_i, \qquad Z_i^\top = -Z_i.$$

Here $(\mathrm{d}\Sigma)_{ii}$ is diagonal, because it contains the differentials of the singular values in the i-th block. Therefore,

$$(\mathrm{d}\Sigma)_{ii} = \mathrm{diag}(\mathrm{diag}(C_{ii})),$$

and, for $\sigma_i > 0$,

$$Z_i = \frac{1}{\sigma_i} \mathrm{skew}(C_{ii}).$$

The diagonal contribution to the differential is

$$\langle A_{ii}, (\Omega_U)_{ii} \rangle + \langle B_{ii}, (\Omega_V)_{ii} \rangle + \langle \overline{\Sigma}_{ii}, (\mathrm{d}\Sigma)_{ii} \rangle.$$

Write

$$T_i := \frac{1}{2} \left((\Omega_U)_{ii} + (\Omega_V)_{ii} \right).$$

Then

$$(\Omega_U)_{ii} = T_i + \frac{1}{2}Z_i, \qquad (\Omega_V)_{ii} = T_i - \frac{1}{2}Z_i.$$

Thus

$$\begin{aligned}&\langle A_{ii}, (\Omega_U)_{ii}\rangle + \langle B_{ii}, (\Omega_V)_{ii}\rangle \\ &= \langle A_{ii} + B_{ii}, T_i\rangle + \frac{1}{2}\langle A_{ii} - B_{ii}, Z_i\rangle.\end{aligned}$$

The common rotation T_i is a gauge direction inside the degenerate singular subspace. Since this direction does not change M, the loss must be invariant with respect to it, and hence its contribution vanishes:

$$\langle A_{ii} + B_{ii}, T_i\rangle = 0.$$

Therefore the observable diagonal contribution is

$$\frac{1}{2}\langle A_{ii} - B_{ii}, Z_i\rangle + \langle \overline{\Sigma}_{ii}, (\mathrm{d}\Sigma)_{ii}\rangle.$$

Substituting

$$Z_i = \frac{1}{\sigma_i}\operatorname{skew}(C_{ii}), \qquad (\mathrm{d}\Sigma)_{ii} = \operatorname{diag}(\operatorname{diag}(C_{ii})),$$

we obtain

$$\begin{aligned}&\frac{1}{2}\langle A_{ii} - B_{ii}, Z_i\rangle + \langle \overline{\Sigma}_{ii}, (\mathrm{d}\Sigma)_{ii}\rangle \\ &= \left\langle \frac{1}{2\sigma_i}\operatorname{skew}(A_{ii} - B_{ii}), C_{ii} \right\rangle + \langle \overline{\Sigma}_{ii}, C_{ii}\rangle.\end{aligned}$$

Hence, for $\sigma_i > 0$,

$$R_{ii} = \overline{\Sigma}_{ii} + \frac{1}{2\sigma_i}\operatorname{skew}(A_{ii} - B_{ii}).$$

If $\sigma_i = 0$, then the relative rotation Z_i is not observable from M, because it is multiplied by σ_i. Thus only the singular-value part contributes, and we take

$$R_{ii} = \overline{\Sigma}_{ii}.$$

Therefore,

$$\mathrm{d}\mathcal{L} = \langle U R V^\top, \mathrm{d}M\rangle,$$

which proves

$$\overline{M} = U R V^\top.$$

□

I Complete SVD Forward and Backward Pipeline

Algorithm 4 combines the CANS SVD forward pass with the regularized EVD and polar backward formulas. The backward pass is a custom rule evaluated from the returned factors, with $W = UV^\top$, and reuses the eigenpairs computed in the forward pass. At rank-deficient inputs, this specifies a regularized backward convention. We write $A_{\mathrm{sym}} = (A + A^\top)/2$ and take $\overline{\Sigma}$ to be diagonal. Since the forward pass returns $V^\top$, its incoming gradient is transposed to obtain $\overline{V}$.

Algorithm 4 CANS SVD with a regularized backward pass

Require: $M \in \mathbb{R}^{m\times n}$ $(m \geq n)$, tolerance $\varepsilon_{\text{QR}} > 0$, regularization parameter $\varepsilon > 0$
Ensure: SVD factors $U, \Sigma, V^\top$ and a regularized backward rule

Forward pass
Compute $U, \Sigma, V^\top$ using Algorithm 1 with tolerance ε_{QR}
Cache M, U, Σ, V and return $U, \Sigma, V^\top$
Backward pass
Receive incoming gradients $\overline{U}, \overline{\Sigma}, \overline{V}$
$W \leftarrow UV^\top$
$\overline{W} \leftarrow \overline{U}V^\top$
$\overline{V}_{\text{tot}} \leftarrow \overline{V} + W^\top \overline{U}$
$B \leftarrow \frac{1}{2}\big(V^\top \overline{V}_{\text{tot}} - \overline{V}_{\text{tot}}^\top V\big)$
$(K_\varepsilon)_{ij} \leftarrow \dfrac{\sigma_i - \sigma_j}{(\sigma_i - \sigma_j)^2 + \varepsilon^{1/2}}, 1 \leq i, j \leq n$
$\overline{H} \leftarrow V\big(\overline{\Sigma} + K_\varepsilon^\top \odot B\big)V^\top$
$R_\varepsilon \leftarrow V\Sigma\big(\Sigma^2 + \varepsilon^{1/2} I\big)^{-1} V^\top$
$C_\varepsilon \leftarrow \big(\overline{H} - R_\varepsilon M^\top \overline{W} R_\varepsilon\big)_{\text{sym}}$
$(T_\varepsilon)_{ij} \leftarrow \dfrac{\sigma_i + \sigma_j}{(\sigma_i + \sigma_j)^2 + \varepsilon^{1/4}}, 1 \leq i, j \leq n$
$X_\varepsilon \leftarrow V\big(T_\varepsilon \odot (V^\top C_\varepsilon V)\big)V^\top$
$\overline{M}_\varepsilon \leftarrow \overline{W} R_\varepsilon + 2MX_\varepsilon$
return $\overline{M}_\varepsilon$