TURBO-MUON: ALMOST-ORTHOGONAL PRE-CONDITIONING FOR FAST MUON UPDATES

Anonymous authors
Paper under double-blind review

ABSTRACT

Orthogonality-based optimizers, such as Muon, have recently shown strong performance across large-scale training and community-driven efficiency challenges. However, these methods rely on a costly gradient orthogonalization step. Even efficient iterative approximations such as Newton-Schulz remain expensive, typically requiring dozens of matrix multiplications to converge.

We introduce a pre-conditioning procedure that improves the initialization of the Newton–Schulz iterations while incurring negligible overhead. Furthermore, our pre-conditioning reduces the initial polar error and enables the removal of one Newton-Schulz iteration (out of the five iterations usually used in practice). The resulting implementation significantly reduces Muon's overhead.

At the end-to-end training level, we observe consistent runtime improvements across speed-run and standard benchmarks, including $\sim 3\%$ reductions in training time on multiple fast training benchmarks, while matching reference performance on both language and vision tasks. Crucially, these improvements require no hyperparameter tuning and can be adopted as a simple drop-in replacement.

Beyond empirical gains, we provide theoretical insight into the geometry of the update and its potential robustness against feature collapse. Our code is publicly available at anonymous-github.

1 INTRODUCTION

Optimization algorithms are a central component of modern deep learning, directly affecting training efficiency, stability, and final model performance. Most widely used methods build on stochastic gradient descent, either directly through momentum-based variants or through adaptive extensions such as Adam and AdamW Kingma & Ba (2015); Loshchilov & Hutter (2017). More recently, orthogonalization of weight updates has become a central ingredient in several optimizers Bernstein & Newhouse (2024; 2025). The most prominent example is the Muon optimizer Keller et al. (2024b), which has been shown to consistently surpass AdamW Kingma & Ba (2015); Loshchilov & Hutter (2017) across diverse training regimes Wen et al. (2025) and has been adopted in large foundation models such as Kimi-K2 and GLM-4.5 Kimi-AI et al. (2025); Zeng et al. (2025). Recent large-scale evaluations further report favorable scaling of Muon for LLM training Liu et al. (2025); Shah et al. (2025). In Muon and its variants, updates are projected toward the orthogonal manifold to accelerate convergence, stabilize training, and support robust hyperparameter transfer across model scales Bernstein. (2025); Bernstein & Newhouse (2025); Large et al. (2024); Pethick et al. (2025).

However, the computational cost of this projection remains the primary barrier to broad adoption at scale. Exact orthogonalization via SVD, while numerically precise, is impractical on modern accelerators due to its high cost and instability for large matrices in low precision. In practice, state-of-the-art methods therefore rely on efficient but imprecise iterative schemes such as the Newton-Schulz method (abbreviated NS and also known as Björck) Björck & Bowie (1971); Franz Louis et al. (2025).

In these methods, wall-clock overhead scales with the number of iterative steps needed to achieve a target polar error (i.e., the Frobenius distance to the closest orthogonal matrix), creating a direct trade-off between runtime and optimization quality Shulgin et al. (2026). This explains why Newton-Schulz is usually used with only five iterations. This work focuses on improving this specific, low-iteration regime, which is most relevant to deep-learning practitioners.

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.

This paper. We introduce a pre-conditioned Newton-Schulz method based on Almost Orthogonal Layer (AOL) pre-conditioning Prach & Lampert (2022). We show that this pre-conditioning yields a significantly improved initialization of the Newton-Schulz iteration toward the polar factor, enabling the removal of one Newton-Schulz iteration. The pre-conditioning incurs negligible overhead by reusing existing computations, leading to a better trade-off between runtime and orthogonalization quality (Fig. 1b). Crucially, we prove that the resulting update remains a steepest descent direction under a modified norm.

Empirically, Turbo-Muon acts as a drop-in replacement for Muon Keller et al. (2024b), consistently reducing training time by up to 4% while preserving downstream performance across diverse settings, including highly optimized speed-run benchmarks on NanoGPT and CIFAR-10 as well as the large-scale ImageNet-1K benchmark.

2 Related Work

Orthogonality-based optimizers form a recent class of training algorithms that enforce or approximate orthogonal updates to improve conditioning and stability during deep network optimization. The foundational work of Muon Keller et al. (2024b) introduced the idea of orthogonalizing parameter updates via iterative matrix normalization, such as the Newton-Schulz method, yielding isotropic update directions and smoother convergence dynamics. Follow-up works such as Deriving Muon Bernstein. (2025) and Old Optimizer, New Norm Bernstein & Newhouse (2024) offered theoretical insights linking these methods to optimization under alternative geometries and norms. Scalable implementations such as Dion Ahn et al. (2025) extended the approach to distributed settings, while Gluon Riabinin et al. (2025) and AdaMuon Si et al. (2025) incorporated adaptivity and layer-wise refinements, bridging the gap between theoretical linear minimization oracle (LMO) frameworks and practical large-model training Pethick et al. (2025). Empirical studies Keller et al. (2024a); Wen et al. (2025); Kimi-AI et al. (2025) further demonstrated that such orthogonalization can accelerate convergence, stabilize training under heavy-tailed gradients, and enhance performance across architectures from GPT-2 scale to billion-parameter models. Together, these works position orthogonality-based optimization as a promising direction for efficient and stable large-scale learning.

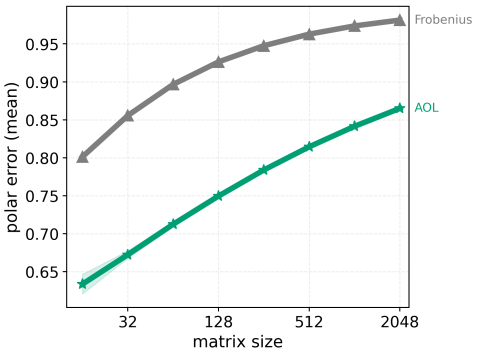
Orthogonalization methods. A variety of orthogonalization schemes have been proposed to construct weight matrices with orthogonal constraints. The Modified Gram-Schmidt QR factorization (LaPlace, 1820) finds the polar factor with an iterative process (one iteration per row). Alternatively, the Cayley transform (Cayley, 1846) establishes a bijection between skew-symmetric and orthogonal matrices through $(I - A)(I + A)^{-1}$, but requires a costly matrix inversion. The exponential map (Singla & Feizi, 2021) also leverages skew-symmetric matrices, generating $Q = \exp(A)$ while typically approximating the exponential via truncated series. Similarly, The Cholesky-based method (Hu et al., 2023) orthogonalizes a matrix with a triangular decomposition $MM^\top = LL^\top$ (where L is triangular) and then solves for $L^{-1}M$, offering efficiency when numerical stability is maintained. Finally, the iterative Newton-Schulz algorithm (Björck & Bowie, 1971; Anil et al., 2019) (also known as the Björck–Bowie algorithm) projects matrices toward the Stiefel manifold via the computation of matrix polynomials. This iteration achieves fast convergence when the input matrix is spectrally normalized. Recent works Keller et al. (2024b); Franz Louis et al. (2025) extended this algorithm with fifth-order polynomials, achieving faster convergence. While these methods provide complementary trade-offs between accuracy, stability, and computational cost, Newton-Schulz variants have gained substantial popularity due to their scalability, making these operations feasible even at large scales Kimi-AI et al. (2025).

3 Background and Motivation

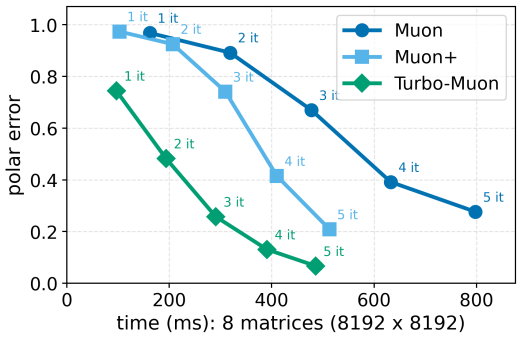
Definitions. Let $\mathbf{X} \in \mathbb{R}^{n\times n}$ denote a square matrix (all results generalize to the non-square case). The singular value decomposition (SVD) of $\mathbf{X}$ is given by

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^\top,$$

where $\mathbf{U}, \mathbf{V} \in \mathbb{R}^{n\times n}$ are orthogonal matrices and $\mathbf{\Sigma} = \operatorname{diag}(\sigma_1, \ldots, \sigma_n)$ is a diagonal matrix containing the singular values $\sigma_1 \geq \cdots \geq \sigma_n \geq 0$. This also defines the spectral norm of a matrix: $\|X\|_2 = \sigma_1$.



(a) **Comparing pre-conditioning methods:** While normalization is primarily used for numerical stability, AOL also reduces the polar error of the initial iterate of Newton-Schulz.



(b) **Practical implementations of orthogonalization face a trade-off between polar error and computation time**. Thanks to pre-conditioning, our method reduces the polar error of the Newton-Schulz algorithm. This improves the trade-off between convergence and runtime.

Figure 1: **Our approach** consists of accelerating the core computation of the Muon optimizer: the Newton-Schulz algorithm. With adequate pre-conditioning (1a), this routine can achieve lower polar error in less time, translating into a more efficient optimization, either in terms of performance (fewer steps) or runtime (lower time per step).

A matrix $\mathbf{X}$ is said to be *orthogonal* if it satisfies $\mathbf{X}^\top\mathbf{X} = \mathbf{X}\mathbf{X}^\top = \mathbf{I}$. For non-square matrices, only one of these two equalities can hold, in which case the matrix is referred to as *semi-orthogonal*. To assess deviations from orthogonality, we define the *orthogonality error* as $\varepsilon_{\text{ortho}}(\mathbf{X}) = \|\mathbf{X}^\top\mathbf{X} - \mathbf{I}\|_F$.

In the context of this paper, we seek the closest orthogonal matrix to a given matrix $\mathbf{X}$, with respect to the Frobenius norm. This matrix is the *polar factor* (PolarFactor($\mathbf{X}$)), and, for any other matrix Z, we assess the approximation quality by the *polar error* ($\varepsilon_{\text{polar}}(Z, \mathbf{X})$):

$$\text{PolarFactor}(\mathbf{X}) = \mathbf{Q} = \mathbf{U}\mathbf{V}^\top \quad \text{and} \quad \varepsilon_{\text{polar}}(Z, \mathbf{X}) = \frac{\|Z - \mathbf{Q}\|_F}{\sqrt{n}}.$$

These metrics will be used throughout the paper to evaluate the orthogonalization accuracy of the tth iteration $\text{NS}_t(\mathbf{X})$ of the Newton-Schulz algorithm.

Existing methods: strengths and limitations. The original *Muon* formulation introduced the Newton-Schulz (NS) variant of the Björck iteration, which relies on quintic polynomial expansions to approximate the polar factor efficiently Keller et al. (2024b). Building on this, Franz Louis et al. (2025) proposed iteration-dependent polynomial coefficients, allowing convergence within five or six steps–a significant improvement over the fixed-coefficient scheme. This idea was later extended and accelerated by Grishina et al. (2025) and Amsel et al. (2025), who analyzed optimal polynomial families and convergence regimes for orthogonalization.

In parallel, Ahn et al. (2025) introduced *Dion*, a distributed implementation supporting model sharding (i.e., splitting a large neural network model into smaller parts) and tensor-parallel decomposition. In addition to their optimizer, they provide an efficient implementation of Muon featuring Triton kernels Tillet et al. (2019) that efficiently exploit the matrix's symmetric structure to reduce redundant computation and memory access. While these contributions collectively enhance the scalability of orthogonality-based optimizers, the computational cost of orthogonalization remains a limiting factor. In practice, Newton-Schulz iterations require dense matrix–matrix products, which interacts nontrivially with sharded training, leading to compute and communication bottlenecks in extreme-scale settings Essential-AI (2025). Hence, each reduction in cost directly broadens the applicability of orthogonality-based optimizers to increasingly large models.

Newton-Schulz iteration To avoid the limitations of exact methods, practical applications often rely on approximate methods. Notably, Björck & Bowie (1971) showed that the following scheme:

$$X_{k+1} = (1 - \beta)X_k - \beta X_k X_k^\top X_k \tag{1}$$

Algorithm 1 Newton-Schulz Muon/Muon+

Require: Initial point: $\mathbf{X_0} \in \mathbb{R}^{n\times n}$, (a, b, c) NS factors

1: $s = 1/\|\mathbf{X_0}\|_F$ ▷ Frobenius scaling factor
2: $\mathbf{X_1} = \mathbf{X_0}s$ ▷ k=1
3: $\mathbf{A_k} = \mathbf{X_k}^\top \mathbf{X_k}$ ▷ polynomial 1
4: $\mathbf{B_k} = b\mathbf{A_k} + c\mathbf{A_k}\mathbf{A_k}$ ▷ polynomial 2
5: $\mathbf{X_{k+1}} = a\mathbf{X_k} + \mathbf{X_k}\mathbf{B_k}$ ▷ polynomial 3
... ▷ Iteration $k+1$ (no rescaling, step 3)

Algorithm 2 Newton-Schulz Turbo-Muon

Require: Initial point: $\mathbf{X_0} \in \mathbb{R}^{n\times n}$, (a, b, c) NS factors

1: $\mathbf{A_0} = \mathbf{X_0}^\top \mathbf{X_0}$ ▷ initial matmul
2: $s = (1/\sqrt{\| \, |\mathbf{A_0}_i| \, \|_1})_{i\in[0,n-1]}$ ▷ AOL vector
3: $\mathbf{X_1} = \mathbf{X_0}s$ ▷ rescaling $\mathbf{X_1}$
4: $\mathbf{A_1} = s^\top \mathbf{A_0}s$ ▷ A_1 without recomputation
5: $\mathbf{B_1} = b\mathbf{A_1} + c\mathbf{A_1}\mathbf{A_1}$ ▷ polynomial 2
6: $\mathbf{X_2} = a\mathbf{X_1} + \mathbf{X_1}\mathbf{B_1}$ ▷ polynomial 3, k= 1
... ▷ Muon iteration $k+1$ (Alg. 1 step 3)

Figure 2: Our approach changes the normalization step in the first iteration of the Newton-Schulz algorithm. This allows for a faster convergence at a negligible cost since A_0 is reused in the first iteration. This approach does not impact memory consumption, which still maintains 3 buffers at most.

converges if $\|X_0\|_2 \leq 1$ (in spectral norm) for $\beta \in [0, 0.5]$. More recently, Keller et al. (2024b) introduced a variant that uses 5th-order polynomials:

$$X_{k+1} = aX_k + bX_kX_k^\top X_k + cX_kX_k^\top X_kX_k^\top X_k \tag{2}$$

which is computed in practice in three steps described in Alg. 1 lines 3, 4, and 5. This formulation opens the door to new optimizations: Franz Louis et al. (2025) showed that (a, b, c) coefficients can be changed across iterations, using (a_k, b_k, c_k) instead of a constant (a, b, c). Amsel et al. (2025) and Grishina et al. (2025) both explored the convergence of these algorithms alongside methods to find efficient polynomial factors. Notably, they observed that the most suitable parameters rely on assumptions about the smallest singular values of X. Throughout this paper and in our experiments, we compare to two widely used variants of the Newton-Schulz algorithm[1]:

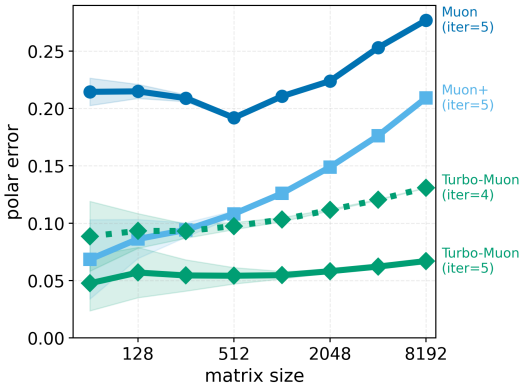
- **Muon** Keller et al. (2024b): the original Newton-Schulz implementation, written in plain PyTorch. It typically uses five iterations, with constant polynomial factors.
- **Muon+** Ahn et al. (2025): a more efficient implementation, integrating Triton kernels Tillet et al. (2019) proposed in Ahn et al. (2025) and adaptive polynomial factors from Franz Louis et al. (2025), computed for five iterations. [2]
- **Turbo-Muon** : our approach, combining a pre-conditioning method with an additional fused Triton kernel to further reduce runtime and maintain high numerical precision. Turbo-Muon uses four iterations by default, and uses polynomial factors inherited from Muon+.

This selection allows us to observe the impact of theoretical improvements (such as adaptive polynomial factors) and design choices (such as Triton kernels). Other approaches such as Grishina et al. (2025); Amsel et al. (2025) are discussed in Sec. E.

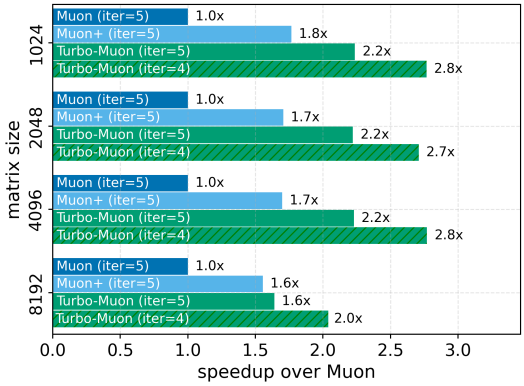
4 Methodology

Existing improvements of Newton-Schulz focus either on the polynomial steps applied Grishina et al. (2025); Amsel et al. (2025), or on efficient implementations of the algorithm Ahn et al. (2025). However, all practical implementations start with an overlooked step: normalizing the input matrix by its Frobenius norm. This ensures that the Newton-Schulz iterative scheme converges Björck & Bowie (1971). In this work, we explore an alternative normalization scheme and observe significant improvements in the convergence of the first Newton-Schulz iterations. Moreover, this better performance allows the removal of a Newton-Schulz iteration without loss of precision.

[1]Muon uses the code from github.com/KellerJordan/Muon and Muon+ is based on github.com/microsoft/dion.
[2]Dion is a distributed variant of Muon (with non-trivial modifications), we only compare here to their implementation of the Newton-Schulz algorithm to keep results comparable.



(a) **Pre-conditioning is effective for large matrices:** Applying AOL before the algorithm improves convergence speed, especially for large matrices. In this context, an iteration can be removed while still achieving improved convergence.



(b) **Breakdown of our speedup:** Building atop Muon+, which includes Triton kernels and dynamic polynomial parameters ($1.7\times$ faster), our approach adds an extra Triton kernel that unlocks moderate gains ($2.2\times$ faster). Finally, removing one of the five iterations also improves runtime ($2.8\times$ faster).

Figure 3: **Turning pre-conditioning into runtime:** Applying AOL before the algorithm improves its convergence speed (**a**). This can be used to remove an iteration while achieving a similar polar error. Removing one of the five iterations improves the runtime (**b**), making optimizers like Muon more scalable to large matrices.

4.1 Almost Orthogonal Pre-conditioning

To ensure $\|X_0\|_2 \leq 1$, existing implementations perform an initial normalization step $X_0/\|X_0\|_F$. While this guarantees convergence, in practice it often leads to $\|X_0\|_2 \ll 1$ and does not modify the condition number, since the ratio $\sigma_{\max}/\sigma_{\min}$ remains unchanged.

In contrast, the “Almost Orthogonal Layer" (AOL) parametrization introduced by Prach & Lampert (2022) provides a lightweight alternative to normalization that simultaneously enforces $\|X_0\|_2 \leq 1$ while improving the resulting matrix conditioning. In the following, we define:

$$\mathrm{AOL}(X_0) = X_0.\mathrm{diag}\left(\sum_j |X_0^\top X_0|_{(i,j)}\right)^{-1/2} \tag{3}$$

where the inverse square root of the row sum of the Gram matrix acts as a scaling vector applied column-wise to X_0. This brings the matrix closer to an orthogonal form. In Fig. 1a, we compare the normalized polar error for the two normalization methods when applied to matrices sampled from a normal distribution (sizes ranging from 16 to 2048, with 32 matrices for each matrix size and algorithm). As shown by Prach & Lampert (2022), AOL converges to the polar factor if the matrix is near column-orthogonal and is of full rank. Importantly, this phenomenon can occur naturally on random high-dimensional matrices, as a byproduct of concentration phenomena which tend to enforce near-orthogonality (see Section 3.2.4 of Vershynin (2009)). We also compare AOL with an alternative preconditioning strategy in Sec. D, further supporting its effectiveness in this setting.

Pre-conditioning for free. Standard implementations of the Newton-Schulz algorithm normalize the input matrix X_0 by its Frobenius norm up front to guarantee convergence (Alg. 1 lines 1-2). Turbo-Muon instead defers this normalization until after forming the Gram matrix $A_0 = X_0^\top X_0$ (Alg. 2 line 1). The AOL scaling vector s is then computed directly from the row sums of $|A_0|$ via a cheap element-wise inverse square root (line 2). The initialization of the Newton-Schulz states is performed by applying this scaling vector via broadcasting ($A_1 = s^\top A_0 s$ and $X_1 = X_0 s$, Alg. 2 lines 3-4). Crucially, this reuses the Gram computation and avoids any additional matrix multiplications. AOL pre-conditioning thus incurs negligible runtime overhead, and only $\mathcal{O}(\min(m,n))$ extra memory.

4.2 Benefits of AOL pre-conditioning for Newton-Schulz convergence

The observation made in Sec. 4.1 and Fig. 1a motivates the use of AOL as a pre-conditioner with the goal of removing one iteration from the Newton-Schulz scheme. Our objective is to determine whether iterative approaches to estimating the polar factor exhibit improved performance when preceded by AOL pre-conditioning in the Muon setting. In this section, we conduct empirical studies on the effects of AOL pre-conditioning on the convergence of the Turbo-Muon algorithm toward the ideal polar factor. Our study shows that under widely adopted practical constraints (i.e., using five iterations), our approach yields better approximations of the polar factor on a variety of random matrices.

Adequate pre-conditioning reduces the polar error. In Fig. 3a, for each matrix size, we sampled random matrices from a standard normal distribution[3] and compared the three implementations listed in Sec. 3 on the same samples. We estimate confidence intervals on the polar error (with 32 samples) using the singular value decomposition applied to the non-conditioned matrix (as defined in Sec. 3). It first shows how the dynamic polynomial parameters from Ahn et al. (2025) reduce the polar error from the baseline of Keller et al. (2024b). Additionally, AOL pre-conditioning (*Turbo-Muon iter=5* in Fig. 3a) further reduces the polar error significantly. Importantly, our approach lowers the polar error enough to allow the removal of one iteration (*Turbo-Muon iter=4*), while still achieving a competitive or improved polar error.

Removing one iteration and its impact on runtime. The default Newton-Schulz setting in Muon uses five iterations. Importantly, we have shown that we can leverage AOL pre-conditioning to remove a Newton-Schulz iteration without losing out on approximation quality relative to Muon and Muon+. Indeed, removing a single iteration reduces the total computational cost by $\approx 20\%$. This becomes particularly advantageous with large matrices, where each iteration follows a time complexity of $\mathcal{O}(n^3)$. Alongside this cost reduction, we leverage the Triton kernels proposed in Ahn et al. (2025); Lin (2025), which improves computation time by using the symmetric nature of the operations used in Alg. 1 lines 3 and 4: since $X^\top X$ is symmetric, only one triangular part needs to be computed, effectively saving half of the required computation. However, Alg. 1 line 5 uses a PyTorch operator that induces a duplicate load of X, while this has a small impact on the overall runtime, fixing this issue with a dedicated Triton kernel has a multiplicative effect with the removal of one iteration, leading to increased speedups.

Stress-testing on gradient-like matrices. The convergence experiments above with normal distribution matrices may not fully reflect the statistics of gradients encountered during neural network training. In particular, empirical studies by Simsekli et al. (2019) have shown that stochastic gradients often exhibit heavy-tailed behavior, with tail indices typically ranging from 1.0 to 1.8 in classification networks, and (Kunstner et al., 2024) shows similar phenomena in language transformers. In Sec. B, we conduct additional experiments on matrices sampled from a heavy-tailed Levy distribution, and show that the conclusions of our empirical convergence experiments remain valid in these settings.

4.3 Ablation study

The proposed method relies on pre-conditioning the gradient matrix, and experiments are conducted using the optimized version Muon+, which integrates Triton kernels Tillet et al. (2019) and adaptive polynomial factors from Franz Louis et al. (2025). We provide in Table 1 an ablation study that separates the contributions of each component. The results show that the proposed pre-conditioning yields both a significant reduction in polar error (0.06 compared to 0.17) and a substantial runtime speedup (up to 2.83).

5 Theoretical Analysis

Although the previous section shows that, in the practical regime of fewer than five iterations, the proposed method achieves lower polar error (relative to the true projection of the gradients) with

[3]Experiments in Sec. B show that our results hold for realistic gradient distributions.

Variant	Source	Polar error	Runtime	Cumulative speedup
Muon: baseline (5it)	Keller et al. (2024b)	0.25	130 ms	1.0
+ Dynamic coeffs	Franz Louis et al. (2025)	0.17	unchanged	1.0
Muon+: + Triton kernel	Ahn et al. (2025)	unchanged	77 ms	1.69
+ AOL	Ours	0.06	unchanged *	1.69
+ 3rd Triton kernel	Ours	unchanged	59 ms	2.20
Turbo-Muon : – 1 iter. (4it)	Ours	0.12	46 ms	2.83

Table 1: **Ablation summary**. We report polar error and runtime improvement for each component, using the same protocol as Fig. 1b and Fig. 3a with matrix size set to 4096. *AOL overhead is negligible for large matrices: matrix–vector multiplication is $4096\times$ cheaper than matrix–matrix multiplication.

reduced runtime, the introduced pre-conditioning modifies the asymptotic target of the Newton-Schulz iterations. In particular, as the number of iterations increases, the method converges to $\mathrm{PolarFactor}(AOL(G))$ rather than $\mathrm{PolarFactor}(G)$ (see Sec. J for a detailed analysis of this residual error across iterations).

We build on the work of Bernstein & Newhouse (2024), who show that several optimizers can be interpreted as first-order steepest descent methods under a particular norm. Our analysis, detailed in Sec. A.1, shows that AOL pre-conditioning leads to a problem close to a steepest descent under an induced norm determined by the AOL rescaling. However, this norm is dependent on S, and its impact on the global optimization process remains unclear. Using only the general properties of the matrix S, we will answer the following question: if pre-conditioning induces the steepest descent in a different norm, can this bias the global optimization process?

Lemma 1. *Let S be a diagonal matrix of strictly positive and bounded entries, we have that $\forall\, G \in \mathbb{R}^{m\times n}_{\setminus 0}$:*

$$\langle G, \mathrm{PolarFactor}(GS)\rangle > 0. \tag{4}$$

Meaning that the projected update recovered by Turbo-Muon is always "aligned" with the raw gradient G. Thus, it yields a strict descent direction. The proof is detailed in Sec. A.1.

Geometric properties of the Turbo-Muon rescaling: We analyze the effect of the rescaling matrix S used in Turbo-Muon pre-conditioning. This rescaling can be interpreted as inducing a norm that adapts to the structural coherence of the gradients, thereby normalizing the worst-case spectral contribution of feature clusters.

The rescaling matrix S is derived from the Gram matrix of the gradient, $H = G^\top G$. Geometrically, the entry $H_{jk} = \langle g_j, g_k\rangle$ quantifies the correlation between the gradient updates of feature column j and feature column k. High values indicate that these features are being pushed in similar directions by the loss function. To extract a scalar measure of feature stability from this matrix without expensive eigendecompositions, we use the *Gershgorin Circle Theorem* (Gershgorin, 1931). The theorem states that the eigenvalues of H are strictly bounded by its row sums. We define the Gershgorin coherence bound Λ_j for the j-th feature as:

$$\Lambda_j \triangleq \sum_{k=1}^{n} |(G^\top G)_{jk}| = \underbrace{|(G^\top G)_{jj}|}_{\text{Feature Energy}} + \underbrace{\sum_{k\neq j} |(G^\top G)_{jk}|}_{\text{Structural Redundancy}} \tag{5}$$

This quantity Λ_j serves as an upper bound on the strength and redundancy of feature j as it captures the magnitude of the feature's own gradient, and also its constructive interference with all other correlated features in the layer. By defining the scaling as $S = \mathrm{diag}(\Lambda)^{-1/2}$, the update of Turbo-Muon corresponds to steepest descent in an induced operator norm $\|\cdot\|_{\ell_2\to S}$ (Sec. A.1), that is explicitly shaped by these coherence bounds. We distinguish the following cases during optimization:

It	Turbo-Muon	Muon+	Muon
2	$3.32 \pm 1e^{-3}$	$3.35 \pm 3e^{-3}$	$3.34 \pm 2e^{-3}$
3	$3.29 \pm 1e^{-3}$	$3.31 \pm 2e^{-3}$	$3.30 \pm 2e^{-3}$
4	$3.28 \pm 1e^{-3}$	$3.29 \pm 3e^{-3}$	$3.29 \pm 5e^{-3}$
5	$3.28 \pm 1e^{-3}$	$3.28 \pm 3e^{-3}$	$3.28 \pm 1e^{-3}$

(a) **Our approach allows the removal of one Newton-Schulz iteration:** We trained a 144M GPT model on the FineWeb dataset Penedo et al. (2024) up to the performance of GPT-2 and reported validation loss. We reused the fastest existing training script and replaced the Newton-Schulz implementation without changing any other parameters.

Dataset	Variant	Perf	Time
speed-run Cifar10	Muon+	**94.04**%	1.359s
	T-Muon	94.03%	**1.348**s (-0.8 %)
speed-run nanoGPT	Muon+	**3.28**	273.8s
	T-Muon	**3.28**	**266.0**s (-2.8%)
ImageNet (1K)	Muon+	**69.56**%	422 m
	T-Muon	69.36%	**407** m (-3.6%)

(b) **End-to-end performance and runtime** for several benchmarks using Turbo-Muon with in-place replacement. Turbo-Muon (T-Muon) consistently achieves equivalent performance in less time (s: seconds, m: minutes).

Table 2: **Improving Newton-Schulz convergence has an effective impact on end-to-end training.** With all other parameters fixed, the improved orthogonality translates to a lower final loss (Table 2a). Beyond 4 to 5 iterations, the loss marginally improves; hence, it is better to remove one iteration to reduce training time while maintaining results.

- **Penalization of Coherence:** When a group of features collapses (i.e., learns highly correlated representations), the off-diagonal terms in $G^\top G$ grow large. Consequently, the Gershgorin bound Λ_j increases, and the effective step size s_j for those features decreases.
- **Amplification of Orthogonality:** Conversely, for features that are orthogonal to others, the redundancy term vanishes and the scaling is determined only by the feature's own energy, allowing for relatively larger updates.

Therefore, the steepest descent of Turbo-Muon ensures that the worst-case curvature estimate of every feature channel—bounded by the Gershgorin disk—is at most 1, resulting in a stabilized spectral update that reduces sensitivity to feature collapse.

Limitations: The previous analysis, as in Amsel et al. (2025); Bernstein & Newhouse (2024), relies on the assumption that the update direction exactly matches the $\mathrm{PolarFactor}$. In practice, however, the Newton-Schulz procedure only provides an approximation of this operator. Although we have shown that our method yields a tighter approximation, a gap remains between the theoretical analysis and the actual update used in practice Gonon et al. (2026). Additionally, as in Muon, the effective step size η depends on the gradient matrix G and is implicitly absorbed into the optimizer's learning rate. A dedicated analysis of how AOL pre-conditioning influences η could provide useful insights for principled learning rate selection.

It would therefore be valuable to extend the analysis to explicitly account for approximate updates, for instance along the lines of approaches based on Linear Minimization Oracle (LMO) Pethick et al. (2025). We leave such an investigation to future work.

6 Experiments: End-to-end training with Turbo-Muon

While orthogonalization lies at the core of optimizers such as Muon, it is not the sole factor determining end-to-end performance. In this section, we first demonstrate that the proposed method can be used as a drop-in replacement of Muon, achieving equivalent performance while reducing training time across several benchmarks, including challenging and competitive speed-run settings.

We have seen in the previous section that Turbo-Muon provides both a better approximation of the polar factor (Fig. 3a) and a large speedup (Fig. 3b) for the core routine of Muon (i.e., the Newton-Schulz algorithm). However, this projection is only a part of the full training step, which also includes forward and backward propagation. In this section, we show that an in-place replacement of Muon by Turbo-Muon achieves the same performance in less training time on several benchmarks. We choose two "speed-run" settings in NLP (Modded-NanoGPT) and vision (CIFAR-10), and a standard ImageNet-1K setting.

We use the term "speed-run" to refer to highly optimized end-to-end training benchmarks whose objective is to reach a predefined target performance in the minimum possible wall-clock time. Unlike standard evaluations, these benchmarks are explicitly engineered to eliminate avoidable overheads in the model, data pipeline, and training loop, leaving little room for further acceleration. Consequently, even modest reductions in total runtime are meaningful, as they represent measurable gains on top of already highly optimized training recipes in which the Muon optimizer step has already been minimized. In this context, speed-run benchmarks provide a conservative test of whether improvements to Muon's orthogonalization routine translate into practical end-to-end acceleration.

NanoGPT "speed-run": We evaluate Turbo-Muon on the Modded-NanoGPT benchmark Keller et al. (2024a), a highly optimized "speed-run" setup for training a 124M GPT model on FineWeb until it reaches a validation cross-entropy of 3.28, replicating the estimated performance of GPT-2 Radford et al. (2019). This benchmark is particularly challenging for our method, as it is tuned to minimize the overhead of Muon while emphasizing reproducible runtime measurements. We reuse the most recent training recipe and replace Muon with Turbo-Muon , with a configurable number of Newton-Schulz iterations. All other hyperparameters are kept unchanged to assess drop-in compatibility. Experiments are run on 4×H100, reporting average loss and runtime over 10 runs. An important point is that Turbo-Muon achieves equivalent performance with one fewer iteration (Table 2a) and reduces total runtime from 273.75 ± 0.14 s for the fastest version Muon+ to 266.00 ± 0.10 s for Turbo-Muon (Table 2b). Additional details and experiments are provided in Secs. F and G.

CIFAR-10 "speed-run": We further evaluate Turbo-Muon on the CIFAR-10 speed-run benchmark Jordan (2024), which targets 94% validation accuracy in the minimum possible time on a single H100 GPU. This setting involves training a convolutional neural network (CNN), where Muon reshapes convolutional gradient updates into 2D matrices before applying iterative orthogonalization. This provides a complementary regime to evaluate AOL preconditioning.

As shown in Table 2b, AOL pre-conditioning achieves the same performance while reducing runtime from $1.359s$ to $1.348s$ (0.8%). Despite the highly optimized nature of this benchmark, this gain highlights the drop-in efficiency of Turbo-Muon and its applicability across architectures.

ImageNet-1K: To evaluate the influence of Turbo-Muon on a larger dataset, we also evaluate a drop-in replacement on an ImageNet-1K baseline using a ViT-Tiny/16 model from `timm` Wightman (2019). Table 2b shows that a 3.6% runtime improvement is achieved with a similar final performance.

7 Conclusion and Perspectives

In this work, we propose AOL pre-conditioning to improve the efficiency of orthogonality-based optimizers such as Muon. Under practical approximation budgets (i.e., $t \leq 5$), the proposed approach yields both faster and more accurate orthogonalization: Empirically, AOL significantly reduces the initial polar error compared to Frobenius normalization on large random matrices (Fig. 1a), and enables the removal of one Newton-Schulz iteration while still achieving equal or lower error than the five-step Muon and Muon+ baselines (Fig. 3a). These gains translate into consistent runtime improvements, with AOL pre-conditioning acting as a drop-in replacement that requires no hyperparameter tuning: Across speed-run and standard benchmarks, Turbo-Muon preserves performance while reducing computation time. In particular, we achieve near-identical performance on NanoGPT and CIFAR-10 speed-runs, with up to 3% speedup, and similar trends on larger-scale settings such as ImageNet-1K. At medium scale (Table 4), Turbo-Muon can even provide step-time speedups of 8–10%.

Beyond empirical performance, we provide a geometric interpretation of AOL pre-conditioning through a steepest descent characterization. We show that Turbo-Muon updates are close to the solution of a steepest descent problem under a structured operator norm. This perspective reveals that AOL pre-conditioning implicitly controls the spectral geometry of the update. As a consequence, Turbo-Muon reduces sensitivity to feature collapse and promotes more stable and balanced optimization dynamics.

AI USE STATEMENT

In this work, we used generative AI tools for coding assistance in this project, with limited the use of this tooling to boilerplate functions and utilities. Every function or class written with AI assistance was rigorously checked and tested. The main optimizer class is nearly entirely human written except for checked boilerplate code. Additionally, AI tools were used for polishing grammatical errors in the writing of the main paper. AI tooling was used to check the soundness of the existing human written proofs that were written and checked previously by all authors. AI tooling was also used to improve the visual formatting of pre-existing human written plotting scripts, these scripts *only* changed the formatting of said plots. All concepts and novel ideas in this paper are solely the authors' work. No AI system was used for ideation or concept generation. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

Kwangjun Ahn, Byron Xu, Natalie Abreu, Ying Fan, Gagik Magakyan, Pratyusha Sharma, Zheng Zhan, and John Langford. Dion: Distributed orthonormalized updates. arXiv preprint arXiv:2504.05295, 2025.

Noah Amsel, David Persson, Christopher Musco, and Robert M Gower. The polar express: Optimal matrix sign methods and their application to the muon algorithm. arXiv preprint arXiv:2505.16932, 2025.

Cem Anil, James Lucas, and Roger Grosse. Sorting out lipschitz function approximation. In International Conference on Machine Learning, pp. 291–301. PMLR, 2019.

Jeremy Bernstein. Deriving muon. https://jeremybernste.in/writing/deriving-muon, 2025.

Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. In OPT 2024: Optimization for Machine Learning, 2024.

Jeremy Bernstein and Laker Newhouse. Modular duality in deep learning. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), Proceedings of the 42nd International Conference on Machine Learning, volume 267 of Proceedings of Machine Learning Research, pp. 3920–3930. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/bernstein25a.html.

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.

Arthur Cayley. Sur quelques Propriétés des Déterminants Gauches, pp. 332–336. Cambridge Library Collection - Mathematics. Cambridge University Press, 1846.

Essential-AI. Layer sharding for large-scale training with muon. https://www.essential.ai/blog/infra, 2025. URL https://www.essential.ai/blog/infra.

Cesista Franz Louis, Jiacheng You, and Jordan. Keller. Squeezing 1-2% efficiency gains out of muon by optimizing the newton-schulz coefficients. http://leloykun.github.io/ponder/muon-opt-coeffs/, 2025. URL http://leloykun.github.io/ponder/muon-opt-coeffs/.

Semen Aronovich Gershgorin. über die abgrenzung der eigenwerte einer matrix. Izvestiya Rossiiskoi Akademii Nauk. Seriya Matematicheskaya, 6:749–754, 1931.

Antoine Gonon, Andreea-Alexandra Muşat, and Nicolas Boumal. Insights on muon from simple quadratics. arXiv preprint arXiv:2602.11948, 2026.

Ekaterina Grishina, Matvey Smirnov, and Maxim Rakhuba. Accelerating Newton-Schulz Iteration for Orthogonalization via Chebyshev-type Polynomials, June 2025. URL http://arxiv.org/abs/2506.10935. arXiv:2506.10935 [math].

Kai Hu, Klas Leino, Zifan Wang, and Matt Fredrikson. A recipe for improved certifiable robustness. In International Conference on Learning Representations, 2023.

Keller Jordan. cifar10-airbench. https://github.com/KellerJordan/cifar10-airbench, 2024. GitHub repository.

Jordan Keller, Bernstein Jeremy, Rappazzo Brendan, Vlado Boza, Jiacheng You, Cesista Franz, and Koszarsky. Braden. modded-nanoGPT: Speedrunning the nanoGPT baseline, 2024a. URL https://github.com/KellerJordan/modded-nanogpt.

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

Kimi-AI, Bai Yifan, Bao Yiping, Chen Guanduo, Chen Jiahao, Chen Ningxin, Chen Ruijue, Chen Yanru, Chen Yuankun, and et al. Yutian, Chen. Kimi k2: Open agentic intelligence, 2025. Kimi k2: Open agentic intelligence. arXiv preprint arXiv:2507.20534, 2025.

Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun (eds.), 3rd International Conference on Learning Representations, ICLR,, 2015. URL http://arxiv.org/abs/1412.6980.

Alex Krizhevsky, Geoffrey Hinton, et al. Learning multiple layers of features from tiny images. 2009.

Frederik Kunstner, Alan Milligan, Robin Yadav, Mark Schmidt, and Alberto Bietti. Heavy-tailed class imbalance and why adam outperforms gradient descent on language models. Advances in Neural Information Processing Systems, 37:30106–30148, 2024.

Pierre Simon LaPlace. Théorie analytique des probabilités. Courcier, 1820. URL http://eudml.org/doc/203444.

Tim Large, Yang Liu, Minyoung Huh, Hyojin Bahng, Phillip Isola, and Jeremy Bernstein. Scalable optimization in the modular norm. Advances in Neural Information Processing Systems, 37:73501–73548, 2024.

Tianyang Lin. Flash-muon: An efficient implementation of muon optimizer, 2025. URL https://github.com/nil0x9/flash-muon.

Jingyuan Liu, Jianlin Su, Xingcheng Yao, Zhejun Jiang, Guokun Lai, Yulun Du, Yidao Qin, Weixin Xu, Enzhe Lu, Junjie Yan, et al. Muon is scalable for llm training. arXiv preprint arXiv:2502.16982, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In International Conference on Learning Representations, 2017.

Guilherme Penedo, Hynek Kydlíček, Anton Lozhkov, Margaret Mitchell, Colin A Raffel, Leandro Von Werra, Thomas Wolf, et al. The fineweb datasets: Decanting the web for the finest text data at scale. Advances in Neural Information Processing Systems, 37:30811–30849, 2024.

Thomas Pethick, Wanyun Xie, Kimon Antonakopoulos, Zhenyu Zhu, Antonio Silveti-Falls, and Volkan Cevher. Training deep learning models with norm-constrained LMOs. In Forty-second International Conference on Machine Learning, 2025. URL https://openreview.net/forum?id=2Oqm2IzTy9.

Bernd Prach and Christoph H Lampert. Almost-orthogonal layers for efficient general-purpose lipschitz networks. In European Conference on Computer Vision, pp. 350–365. Springer, 2022.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. OpenAI blog, 1(8):9, 2019.

Artem Riabinin, Egor Shulgin, Kaja Gruntkowska, and Peter Richtárik. Gluon: Making Muon & Scion Great Again! (Bridging Theory and Practice of LMO-based Optimizers for LLMs), May 2025. URL http://arxiv.org/abs/2505.13416. arXiv:2505.13416 [cs].

Ishaan Shah, Anthony M Polloreno, Karl Stratos, Philip Monk, Adarsh Chaluvaraju, Andrew Hojel, Andrew Ma, Anil Thomas, Ashish Tanwer, Darsh J Shah, et al. Practical efficiency of muon for pretraining. arXiv preprint arXiv:2505.02222, 2025.

Egor Shulgin, Sultan AlRashed, Francesco Orabona, and Peter Richtárik. Beyond the ideal: Analyzing the inexact muon update. In International Conference on Artificial Intelligence and Statistics, 2026.

Chongjie Si, Debing Zhang, and Wei Shen. AdaMuon: Adaptive Muon Optimizer, July 2025. URL http://arxiv.org/abs/2507.11005. arXiv:2507.11005 [cs] version: 1.

Umut Simsekli, Levent Sagun, and Mert Gurbuzbalaban. A tail-index analysis of stochastic gradient noise in deep neural networks. In International Conference on Machine Learning, pp. 5827–5837. PMLR, 2019.

Sahil Singla and Soheil Feizi. Skew orthogonal convolutions. In International Conference on Machine Learning, pp. 9756–9766. PMLR, 2021.

Philippe Tillet, Hsiang-Tsung Kung, and David Cox. Triton: an intermediate language and compiler for tiled neural network computations. In Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, pp. 10–19, 2019.

Roman Vershynin. High-dimensional probability. Cambridge University Press Cambridge, UK, 2009.

Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic Pretraining Optimizers and Where to Find Them, September 2025. URL http://arxiv.org/abs/2509.02046. arXiv:2509.02046 [cs].

Ross Wightman. Pytorch image models. https://github.com/rwightman/pytorch-image-models, 2019.

Wikipedia. Lévy distribution - Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=L%C3%A9vy%20distribution&oldid=1322680363, 2025.

Aohan Zeng, Xin Lv, Qinkai Zheng, Zhenyu Hou, Bin Chen, Chengxing Xie, Cunxiang Wang, Da Yin, Hao Zeng, Jiajie Zhang, et al. Glm-4.5: Agentic, reasoning, and coding (arc) foundation models. arXiv preprint arXiv:2508.06471, 2025.

Jack Zhang, Noah Amsel, Berlin Chen, and Tri Dao. Gram newton-schulz. https://dao-ailab.github.io/blog/2026/gram-newton-schulz/, 2026. URL https://dao-ailab.github.io/blog/2026/gram-newton-schulz/.

A On the convergence of Turbo-Muon

A.1 Link with norm constrained steepest descent

In this section, we build upon the work of Bernstein & Newhouse (2024), as they show that Muon's update (without accumulation) solves a steepest descent direction in a $\ell_2 \to \ell_2$ induced operator norm. As a starting point, we define how the norm induced by our preconditioning differs from the original one. We show in Sec. A.2 that preconditioning does not bias the overall optimization process.

Following Bernstein & Newhouse (2024), we formulate the steepest descent update using a penalized objective, which is equivalent to the constrained formulation for a specific step size η, i.e.:

$$\arg \min_{\Delta W \in \mathbb{R}^{m \times n}} \left[\langle G, \Delta W \rangle + \frac{\lambda}{2} \|\Delta W\|^2_{\alpha \to \beta} \right], \tag{6}$$

where $\langle \cdot, \cdot \rangle$ denotes the Frobenius inner product. For any matrix $M \in \mathbb{R}^{m \times n}$, and normed spaces $(\mathbb{R}^{d_{\text{in}}}, \|\cdot\|_\alpha)$ and $(\mathbb{R}^{d_{\text{out}}}, \|\cdot\|_\beta)$, the "$\alpha$ to β", we recall the induced operator norm as:

$$\|M\|_{\alpha \to \beta} = \max_{x \in \mathbb{R}^{d_{\text{in}}}} \frac{\|Mx\|_\beta}{\|x\|_\alpha}. \tag{7}$$

Importantly, as explicited in Proposition 5 of Bernstein & Newhouse (2024). When $\alpha = 2$ and $\beta = 2$, we have that the solution to Equation 6 is:

$$\Delta W_{\|\cdot\|_{\ell_2 \to \ell_2}} = -\eta \, \text{PolarFactor}(G),$$

with η the sum of the singular values of G divided by λ. In this paper, we introduce the Turbo-Muon update as $\text{PolarFactor}(GS)$. With S the diagonal matrix composed of AOL preconditioning coefficients (previously denoted as s), which are strictly positive and bounded in practice. Here we show that this update is the solution of a slightly different steepest descent direction problem formulated using a rescaled norm.

Proposition 1 (Turbo-Muon in the Steepest Descent framework)**.** *For any gradient matrix $G \in \mathbb{R}^{m \times n}_{\setminus 0}$, and any sharpness $\lambda > 0$, consider the problem:*

$$\begin{aligned} &\arg \min_{\Delta W \in \mathbb{R}^{m \times n}} \left[\langle G, \Delta W \rangle_F + \frac{\lambda}{2} \|\Delta W\|^2_{\ell_2 \to S} \right] \\ &\textit{with } \|M\|_{\ell_2 \to S} := \sup_{x \in \mathbb{R}^n_{\setminus 0}} \frac{\|MS^{-1}x\|_2}{\|x\|_2}. \end{aligned} \tag{8}$$

This problem is solved with a step size $\eta = \frac{1}{\lambda} \operatorname{tr}(\Sigma_{GS})$, with $GS = U \Sigma_{GS} V^\top$ the SVD decomposition of GS, with U and V unitary, and an update:

$$\Delta W_{\|\cdot\|_{\ell_2 \to S}} = -\eta \, \text{PolarFactor}(GS) S, \tag{9}$$

This solution is unique if and only if G is of full rank.

This proposition with S, the diagonal matrix filled with rescaling values obtained from Eq. (3) (strictly positive), generalizes a scaled variant of Turbo-Muon as an optimizer that yields steepest descent directions in an induced norm that is dependent on AOL rescalings, as per the framework of Bernstein & Newhouse (2024).

Proof. Given the problem

$$\arg \min_{\Delta W \in \mathbb{R}^{m \times n}} \left[\langle G, \Delta W \rangle_F + \frac{\lambda}{2} \|\Delta W\|^2_{\ell_2 \to S} \right] \quad \text{with } \|M\|_{\ell_2 \to S} := \sup_{x \in \mathbb{R}^n_{\setminus 0}} \frac{\|MS^{-1}x\|_2}{\|x\|_2}. \tag{10}$$

By setting $\Delta W' = \Delta W S^{-1}$, we have $\|\Delta W\|^2_{\ell_2 \to S} = \|\Delta W'\|^2_{\ell_2 \to \ell_2}$.

Besides using the cyclic property of the trace along with the symmetry of S, the inner Frobenius product on the left-hand side of this expression can be modified by:

$$\langle G, \Delta W \rangle_F = \langle G, \Delta W' S \rangle_F = \langle GS, \Delta W' \rangle_F \tag{11}$$

The problem is thus equivalent to minimizing:

$$\arg \min_{\Delta W' \in \mathbb{R}^{m \times n}} \left[\langle GS, \Delta W' \rangle + \frac{\lambda}{2} \|\Delta W'\|^2_{2 \to 2} \right] \tag{12}$$

From Proposition 5 of Bernstein & Newhouse (2024), the solution of this problem is given by:

$$\Delta W' = -\eta \, \mathrm{PolarFactor}(GS) \tag{13}$$

with a step size $\eta = \frac{1}{\lambda} \operatorname{tr}(\Sigma_{GS})$.

$$\Delta W = \Delta W' S = -\eta \, \mathrm{PolarFactor}(GS) S \tag{14}$$

The rescaled Turbo-Muon update is thus close to the solution of a steepest descent framework under the $\|.\|_{\ell_2 \to S}$ norm.

□

Discussion: One main difference between this proof and the update used in Turbo-Muon is the presence of an extra scaling factor S. Applying this scaling vector after the polar factorization would yield an optimizer with dynamics different from Muon's. The exploration of such a novel optimizer is left for future work.

A.2 AOL PRECONDITIONING CANNOT LEAD TO MUON DIVERGENCE

We start by recalling Lemma 1:

Lemma (Lemma 1)**.** *Let S be a diagonal matrix of strictly positive and bounded entries, we have that $\forall\, G \in \mathbb{R}^{m \times n}_{\setminus 0}$:*

$$\langle G, \mathrm{PolarFactor}(GS) \rangle > 0. \tag{15}$$

Meaning that the projected update recovered by Turbo-Muon is always "aligned" with the raw gradient G. Thus, it yields a strict descent direction.

Proof. Write $X := GS$. By definition of the (left) polar factor,

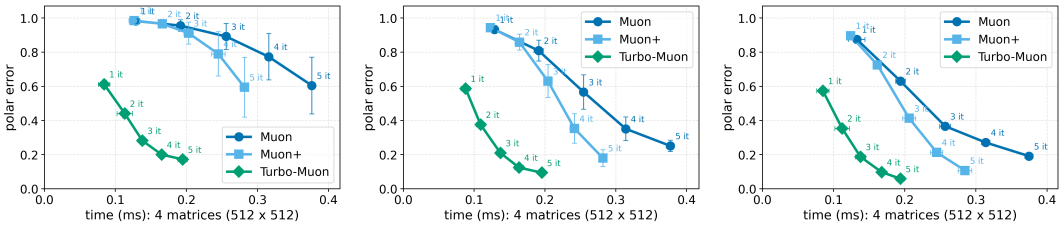
$$\mathrm{PolarFactor}(X) = X \, (X^\top X)^{-1/2},$$

where $(\cdot)^{-1/2}$ denotes the unique positive–semidefinite inverse square root on the support of $X^\top X$. Hence

$$\begin{aligned} \langle G, \mathrm{PolarFactor}(GS) \rangle &= \operatorname{tr}\big(G^\top \ \mathrm{PolarFactor}(X)\big) \\ &= \operatorname{tr}\big(S^{-1} \, X^\top \, \mathrm{PolarFactor}(X)\big), \end{aligned}$$

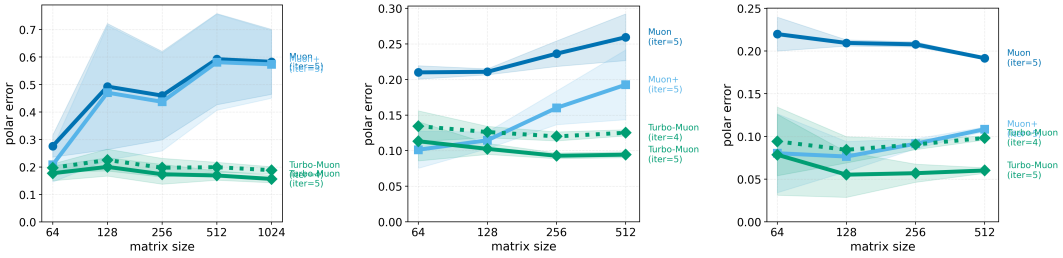
since $G^\top = S^{-1} X^\top$ and $S^{-1} \succ 0$ (as per Eq. (3)). Using $X^\top \, \mathrm{PolarFactor}(X) = X^\top X \, (X^\top X)^{-1/2} = (X^\top X)^{1/2}$, we get

$$\langle G, \mathrm{PolarFactor}(GS) \rangle = \operatorname{tr}\big(S^{-1} \, (X^\top X)^{1/2}\big).$$



(a) Levy distribution: $\alpha = 1.0$, $\beta = 0$ (b) Levy distribution: $\alpha = 1.5$, $\beta = 0$ (c) Levy distribution: $\alpha = 2.0$, $\beta = 0$

Figure 4: We reproduced the Fig. 1b using heavy-tailed distributions. Despite an overall degradation in the resulting polar error, Turbo-Muon outperforms existing approaches by a significant margin.



(a) Levy distribution: $\alpha = 1.0$, $\beta = 0$ (b) Levy distribution: $\alpha = 1.5$, $\beta = 0$ (c) Levy distribution: $\alpha = 2.0$, $\beta = 0$

Figure 5: We reproduced the Fig. 3a using heavy-tailed distributions. Despite an overall degradation in the resulting polar error, Turbo-Muon outperforms existing approaches by a significant margin.

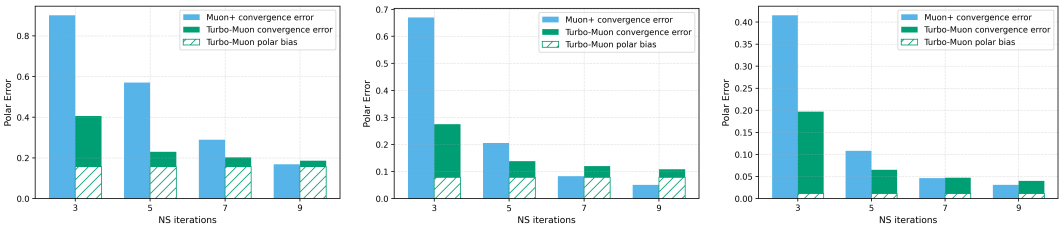
The matrix $(X^\top X)^{1/2} \succeq 0$ and is nonzero because $G \neq 0$ and S is invertible, while $S^{-1} \succ 0$. For positive semidefinite $S \neq 0$ and positive definite T, $\mathrm{tr}(TS) = \mathrm{tr}(T^{1/2} S T^{1/2}) > 0$. Therefore $\mathrm{tr}\big(S^{-1}(X^\top X)^{1/2}\big) > 0$ and the claim follows. The argument does not require G to be square or full rank. If X is rank deficient, interpret $(X^\top X)^{-1/2}$ as the Moore–Penrose inverse square root, so the identity $X^\top \mathrm{PolarFactor}(X) = (X^\top X)^{1/2}$ holds on the support of $X^\top X$, and the same positivity conclusion applies. □

B What About Gradient Distributions ?

B.1 Heavy-tailed matrices

In this section, we run the complementary experiments from Sec. 4.2 and Sec. J on matrices sampled from a Levy distribution of parameters α and β. This choice is motivated by observations made in Simsekli et al. (2019), which found that the tail index of gradients belongs to $\alpha \in [1.0, 1.8]$ in typical training of classification neural networks. Independently, Kunstner et al. (2024) have also shown similar observations on transformers trained on language modeling tasks.

We reproduce the protocol of Fig. 1b from the main paper and report results using three different tail-indices $\{1.0, 1.5, 2.0\}$ in Fig. 4, ensuring we cover realistic values. We set $\beta = 0$ since, despite the fact that the gradient expectation over timesteps is non-null, we made the assumption that its expectation over neurons is null: high values for β induce a rank collapse toward rank 0 (i.e., a constant), which is unrealistically pathological. Intuitively, when $\alpha = 2.0$, the distribution is nearly Gaussian, and for $\alpha \leq 1.0$, estimation of the mean becomes an inconsistent estimator Wikipedia (2025). For all tested values of α, Turbo-Muon outperforms Muon/Muon+ by a significant margin in realistic settings ($t \leq 5$).



(a) Levy distribution: $\alpha = 1.0$, $\beta = 0$

(b) Levy distribution: $\alpha = 1.5$, $\beta = 0$

(c) Levy distribution: $\alpha = 2.0$, $\beta = 0$

Figure 6: Reproduction of Fig. 10 with heavy tailed distributions. This extreme regime–where the singular value decomposition is at its numerical limit– shows that the tail index correlates with the bias error induced by our preconditioning.

Similarly, we reproduced the result of Fig. 3a from the main paper for Levy distributions and reported results in Fig. 5. Again, we observe a global degradation of performances, with a lower degradation for Turbo-Muon . Notably, in these experiments, we observed that the PyTorch implementation of the singular value decomposition was unstable in these regimes, especially for large matrices. Thus limiting these experiments to matrices of size 512×512.

Interestingly, in these regimes, the orthogonality error of Turbo-Muon is very low, highlighting that the remaining error is not due to a lack of convergence but to an increased bias for poorly conditioned matrices. This can be observed by reproducing the Fig. 6 experiment from the paper: for instance, when $\alpha = 1$, the polar bias can reach up to 0.15, which is significantly higher than the bias found with matrices sampled from normal distributions. Yet the drastically improved convergence still leads to a lower polar error compared to Muon+, which requires nine iterations (27 matrix multiplications) to reach a lower polar error than Turbo-Muon . The polar bias observed in Table 6 suggests that empirical gradients have a tail index $1.0 \leq \alpha \leq 1.5$, which is coherent with the observations of Simsekli et al. (2019).

C Result for various matrix sizes

In this section, we show that the results displayed in Fig. 1b of the main paper hold for various matrix sizes sampled from the Normal distribution. Interestingly, two effects are observed: on one side, our preconditioning yields improvements in the polar error (shifting Turbo-Muon on the vertical axis), especially for large matrices; however, this effect becomes smaller for small matrices. On the other side, the triton kernel of Alg. 1 line 5 improves runtime (shifting Turbo-Muon on the horizontal axis) on small matrices, as those lead to communication-bound kernels. Similarly, this effect becomes smaller for large matrices, as those lead to compute-bound kernels. This combination ensures a dominance of Turbo-Muon in all configurations.

D Comparison with other pre-conditioning strategies

AOL preconditioning is one possible way to improve the initialization of the Newton-Schulz iterations, but other normalization strategies may also be considered. As an additional baseline, we evaluate Schatten-4 normalizationGrishina et al. (2025), which rescales a matrix X by its Schatten-4 norm,

$$\|X\|_{S_4} = \left(\sum_i \sigma_i^4 \right)^{1/4}.$$

This pre-conditioning can also be computed efficiently using the Frobenius norm of A_0 (Alg. 2 line 1). Unlike the Frobenius norm, this normalization gives more weight to larger singular values and can therefore provide a stronger control of the spectrum before applying Newton-Schulz.

The results are reported in Figs. 8a and 8b. Schatten-4 normalization consistently improves over the standard Frobenius initialization used in Muon. However, AOL preconditioning remains better across matrix sizes and across Newton-Schulz iterations. In particular, AOL reaches lower polar error at

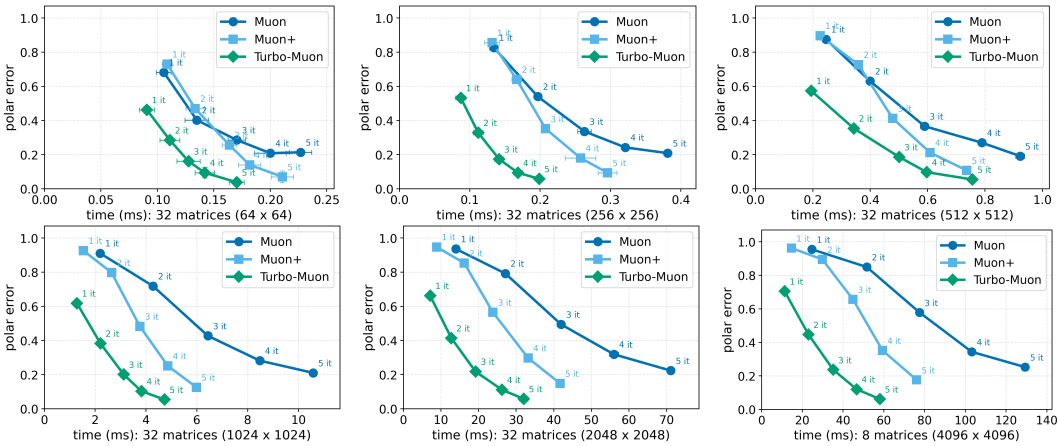
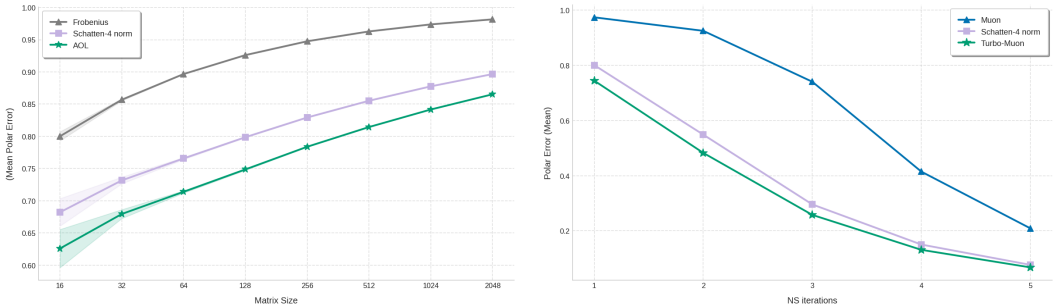


Figure 7: Reproduction of Fig. 1b for various matrix sizes. In all settings, Turbo-Muon outperforms Muon and Muon+ by a significant margin.



(a) Comparison of several pre-conditioning initializations

(b) Comparison of Newton-Schulz across iterations with several pre-conditioning initialization

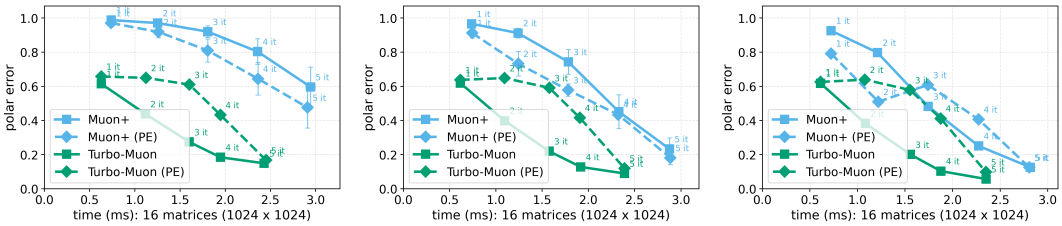
each iteration, which is the relevant regime for Muon-style optimizers, where only a small number of Newton–Schulz steps is used in practice.

E About the tuning of Newton-Schulz coefficients

In this paper, we use dynamic polynomial coefficients to run Newton-Schulz iterations, which means that each step uses different values for a, b, c. However, the implementation of Muon+ Ahn et al. (2025) uses coefficients computed explicitly for a fixed number of 5 iterations. To adapt the algorithm to a lower number of iterations, we employed a straightforward strategy as we reduced the number of iterations by retaining only the n last polynomial coefficients. The motivation behind this choice was to isolate the effect of AOL preconditioning as the only factor responsible for the reduction in polar error between Muon+ and Turbo-Muon . In this section, we demonstrate, through additional experiments, that our strategy is indeed highly effective in regimes with a low number of iterations.

E.1 Ablation

The difference between Muon and Muon+ shows the undeniable impact of using explicitly computed polynomial coefficients for a fixed number of steps. However, the improvement between the coefficients introduced by Franz Louis et al. (2025) and the coefficients introduced by Amsel et al. (2025); Grishina et al. (2025) is more nuanced, and is mainly observed on heavy-tailed distributions. Yet, the application of those polynomial coefficients to Turbo-Muon yields a negative impact. We hypothesize the root cause to be due to the assumption about the minimum singular value, required to compute the coefficients. Therefore, we leave the computation of optimal coefficients for Turbo-Muon as potential future work.



(a) Levy distribution: $\alpha = 1.0$, $\beta = 0$

(b) Levy distribution: $\alpha = 1.5$, $\beta = 0$

(c) Levy distribution: $\alpha = 2.0$, $\beta = 0$

Figure 9: Coefficients from Amsel et al. (2025) are suboptimal when combined with AOL-preconditioning. We recomputed the optimal coefficient for each number of iterations, using default parameters ($l = 10^{-3}$, cushion $= 0.024$, and safety_factor $= 2 \times 10^{-2}$). For both Muon+ and Turbo-Muon , we report the performance using Polar Express (PE) and compare it with the original. On heavy-tailed distributions, these coefficients improve the performance of Muon+ but degrade the performance of Turbo-Muon .

F NanoGPT Speed-Run Details

We adopt the Modded-NanoGPT benchmark Keller et al. (2024a), a highly optimized training setup for a 124M-parameter GPT model on the FineWeb dataset Penedo et al. (2024). The goal is to reach a validation cross-entropy of 3.28, corresponding to the estimated performance of GPT-2 Radford et al. (2019), in minimal wall-clock time. The reference performance is in less than 3 minutes on 8×NVIDIA H100, consuming roughly 0.73B tokens. This benchmark combines multiple architectural, optimization, and systems-level improvements, including rotary embeddings, QK normalization, ReLU^2 activations, sliding-window and flash attention, an FP8 language modeling head, and the Muon optimizer. These optimizations are specifically designed to minimize training time and reduce optimizer overhead, making this setting particularly challenging for methods targeting further efficiency gains. Additionally, it places a strong emphasis on reproducible runtime measurement, enabling us to precisely evaluate whether our approach can also yield performance gains in end-to-end training even in highly efficient settings.

Experimental protocol. We reuse the most recent public training recipe and modify only the optimizer by replacing Muon with Turbo-Muon . The number of Newton–Schulz iterations is treated as a configurable parameter, while all other hyperparameters (learning rate, batch size, architecture, and data pipeline) are kept unchanged. All experiments are conducted on 4× NVIDIA H100 GPUs. For each configuration, we report the average validation loss and runtime over 10 independent runs to account for runtime variability.

Results. As reported in Table 2a, Turbo-Muon achieves equivalent validation performance while requiring one fewer Newton–Schulz iteration. This reduction translates into a consistent decrease in total runtime. In particular, Table 2b shows a reduction from 274.75 ± 0.14 seconds for the fastest Muon+ configuration to 266.00 ± 0.10 seconds when using Turbo-Muon .

Additional analyses. While the experiment depicted in Table 2a compared Muon, Muon+, and Turbo-Muon , we noted that the nano-GPT speedrun script used the implementation of Muon+ with a different set of NS coefficients from Amsel et al. (2025); Grishina et al. (2025) (Depicted as Muon+ (PE) in Sec. E). Given the additional insights of the experiments of Sec. E, this choice could be seen as biased in favor of Turbo-Muon. To alleviate any doubts, we reproduced this experiment using the Polar Express factors tested in Sec. E, which would favor the baseline over Turbo-Muon .

In this setting, we compare Turbo-Muon (PE) with **four** iterations with Muon+ (PE) with **five** iterations. In order to remove one iteration, we applied the same strategy that was used between Muon+ and Turbo-Muon , which is discussed in Sec. E. In order to run this training on 4xH100 instead of 8, we used a gradient accumulation of 2 to match the original results. We performed 10 trials in a row in order to measure variance on the final validation loss and the total runtime. This was done for both the baseline and our improved version to obtain comparable results.

method	iterations	val loss	val loss std	runtime	runtime std
Muon+ (PE)	5it	3.2774	0.0014	273.75s	0.14
Turbo-Muon (PE)	4it	3.2791	0.0013	266.65s	0.10

Table 3: Direct comparison against the original training procedure. Results reported for 10 trials for each variant.

Variant	ms/step	Speedup
Muon variants		
Muon	1378.54 ± 12.35	1.000x
Muon+	1308.09 ± 8.36	1.054x
Turbo-Muon	1261.93 ± 6.89	1.092x
Dion (Muon-derived, large-scale oriented)		
dion-1	3477.93 ± 57.22	0.397x
dion-1/4	1678.71 ± 14.64	0.821x
dion-1/16	1229.01 ± 14.55	1.122x

Table 4: **Turbo-Muon unlocks step-time speedups at medium scale.** We report training throughput (ms per optimization step) when simulating the training of a 1.3B-parameter model on an A100 80GB GPU with a batch size of 32k tokens per GPU, a setting where Muon's overhead is clearly exposed, and Dion's low-rank induces a perceptible loss degradation. The speedup column indicates performance relative to `Muon`, with higher values corresponding to faster training.

Interestingly, we can observe a degradation of only 0.0017 in the final loss. These results can be observed through the lenses of observations made in Sec. E: where we observed that factors obtained from Amsel et al. (2025); Grishina et al. (2025) underperformed when combined with AOL preconditioning. However, this degradation is smaller than the degradation observed when removing one iteration as done in Table 2a.

G Runtime influence of the training regime

In practice, the optimizer step accounts for only a small fraction of total training computational cost, which is usually dominated by forward and backward propagation. Nonetheless, Muon's orthogonalization introduces a measurable overhead compared to AdamW. This overhead decreases as the batch size increases, since Muon's computational cost is independent of the batch dimension, while the model's forward cost scales linearly with it. Larger batches, therefore, improve time-to-target performance and typically move Muon past AdamW on the compute-efficiency frontier Shah et al. (2025), especially as Muon exhibits a larger critical batch sizeWen et al. (2025).

At large model scales, however, global batch size cannot grow indefinitely without sharding, which significantly affects compute efficiency. Naïve replication can increase per-step flops by up to 5×, while more advanced strategies such as sharded matrix multiplication or layer sharding can suffer from bandwidth saturation. The resulting trade-off is tightly linked to the per-device batch size, which limits the amortization of Muon's overhead Essential-AI (2025), leading to an overhead typically around 10-20%.

Within this context, our preconditioned Newton–Schulz method reduces the cost of the orthogonalization subroutine by approximately 3× at fixed polar accuracy. The resulting end-to-end gains are most pronounced at medium scales—where batch size is reduced but full sharding is unnecessary. This regime is illustrated in Table 4: On a 1.3B language model, one A100 with 80G memory can achieve a batch size of 32k tokens without sharding. In this context, Muon's cost is non-negligible, but optimizers tailored for extreme scales, such as DionAhn et al. (2025), are not yet efficient: Dion's low-rank updates–which shine at larger scales–still degrade convergence. Our drop-in preconditioned NS offers a competitive trade-off by reducing computational cost without compromising accuracy.

H CIFAR-10 SPEED-RUN DETAILS

Benchmark description. We consider the CIFAR-10 speed-running task Jordan (2024), which aims to reach 94% validation accuracy on CIFAR-10 Krizhevsky et al. (2009) in minimal wall-clock time on a single H100 GPU. Unlike NanoGPT, this benchmark involves training a convolutional neural network (CNN).

Optimizer behavior. In this setting, Muon reshapes convolutional gradient updates into matrices and applies iterative orthogonalization, providing a complementary regime to evaluate AOL preconditioning.

Results. Turbo-Muon achieves identical accuracy while reducing runtime from 1.359 s to 1.348 s, corresponding to a 0.81% speedup (Table 2b).

Discussion. Although the absolute gain is modest due to the already optimized implementation, this result confirms that AOL preconditioning consistently improves efficiency across heterogeneous architectures and training setups.

I ON THE COMPLEMENTARITY OF TURBO-MUON WITH OTHER MUON IMPROVEMENTS

Concurrent with this work, Zhang et al. (2026) proposed a Gram-based, hardware-aware variant of Newton-Schulz orthogonalization that further reduces the cost of Muon updates. This approach is complementary to our preconditioning strategy, as it modifies the implementation of the Newton-Schulz iterations whereas AOL modifies their initialization. We therefore evaluate their combination in Table 5 evaluating the runtime and residual polar error after five iterations of Newton-Schulz, on random input matrices of size 2048. Gram Newton-Schulz substantially reduces runtime compared to the Muon+ implementation, but slightly increases the polar error. Adding AOL preconditioning recovers part of this loss in approximation quality, while incurring only a modest runtime overhead.

Table 5: Runtime and polar error of Newton–Schulz variants.

Variant	Time (ms)	Polar error
Muon+	26.031	1.7419×10^{-2}
Gram NS Zhang et al. (2026)	20.621	1.7438×10^{-2}
Turbo-Muon + Gram NS	22.717	1.7431×10^{-2}

J BEYOND ITERATIVE APPROXIMATIONS: ON THE ASYMPTOTIC BEHAVIOR OF TURBO-MUON

In the previous sections, we have demonstrated that $\varepsilon_{\text{polar}}(\text{NS}_t \circ \text{AOL}, \mathbf{X}) \lessapprox \varepsilon_{\text{polar}}(\text{NS}_t, \mathbf{X})$ in the widely adopted practical settings where $t \leq 5$. This means that the polar error is of the same order as the usual error made by approximation algorithms. However, the nature of this error remains to be determined: When does the remaining polar error come from a lack of orthogonality? or when does it come from a discrepancy between $\text{PolarFactor}(\text{AOL}(\mathbf{X}))$ and $\text{PolarFactor}(\mathbf{X})$? In the latter case, can this new type of error bias the global optimization process? In this section, we quantify how AOL preconditioning can introduce a bias and prove that this eventual bias still yields a strict descent direction.

Characterizing the remaining polar error. We can quantify the bias error induced by AOL preconditioning, by computing precisely $\text{PolarFactor}(X) = Q$ and $\text{PolarFactor}(AOL(X)) =$

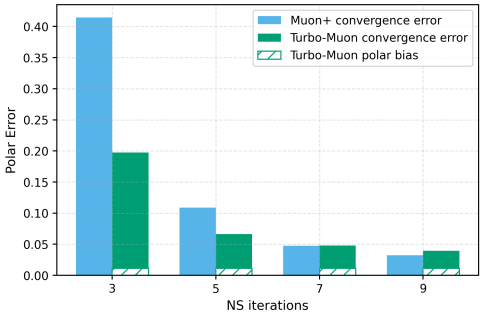


Figure 10: **Understanding the nature of the remaining polar error.** We decompose the polar error of the Turbo-Muon algorithm as an approximation error that depends on the number of NS iterations, along with a bias error that is introduced by AOL preconditioning. Results are measured on 100 random normal matrices.

Q_{aol}:

$$\varepsilon_{\text{bias}}(\text{AOL}, \mathbf{X}) = \frac{\|\mathbf{Q} - \mathbf{Q}_{aol}\|_F}{\sqrt{n}}$$

This can be interpreted as the irreducible error that cannot be reduced after full convergence of the Newton-Schulz algorithm with AOL preconditioning, since it converges toward Q_{aol}. Similarly, we can define:

$$\varepsilon_{\text{approx}}(\text{NS}_t, \text{AOL}(\mathbf{X})) = \frac{\|\mathbf{Q}_{aol} - \text{NS}_t \circ \text{AOL}(\mathbf{X})\|_F}{\sqrt{n}}$$

Which represents the approximation error between $NS_t(AOL(X))$ and Q_{aol}. These two definitions connect to the polar error with a direct application of the triangle inequality:

$$\varepsilon_{\text{polar}}(\text{NS}_t \circ \text{AOL}, \mathbf{X}) \leq \underbrace{\varepsilon_{\text{bias}}(\text{AOL}, \mathbf{X})}_{\text{irreducible bias}} + \underbrace{\varepsilon_{\text{approx}}(\text{NS}_t, \text{AOL}(\mathbf{X}))}_{\text{approximation error}}$$

This allows us to view the approximation error of the polar factor of X by Turbo-Muon as a convergence-related error $\varepsilon_{\text{approx}}$ and a bias-related error $\varepsilon_{\text{bias}}$ term, which are dependent on the original matrix $\mathbf{X}$. Importantly, we assume that $\lim_{t\to\infty} \varepsilon_{\text{approx}}(\text{NS}_t, \mathbf{X}) = 0$, as proven in Björck & Bowie (1971). To illustrate this phenomenon, in Fig. 10, we run an experiment where we orthogonalize the same batch of 100 random matrices using Newton-Schulz iterative steps. Since existing methods had coefficients computed for only five iterations, we recomputed those using the method from Amsel et al. (2025). For each number of iterations, we quantify $\varepsilon_{\text{bias}}(\text{AOL}, \mathbf{X})$ (in dashed green) alongside $\varepsilon_{\text{approx}}(\text{NS}_t, \text{AOL}(\mathbf{X}))$ (in dark green), finally, total polar error is compared with the polar error obtained without preconditioning, assuming that Newton-Schulz is perfectly unbiased (in dark blue). We observe that while Turbo-Muon is indeed biased, its bias is compensated by the superior approximation speed it confers in practically adopted low t settings. When t is high, the beneficial effects of preconditioning are absorbed while the bias remains. For reference, nine iterations of Newton-Schulz require 27 matrix multiplications in total.

Characterizing AOL preconditioning when $t \to \infty$. We have shown that the bias effect is negligible in the low iteration number of Newton-Schulz usually used in Muon. But we might want to evaluate the effect it could induce in regimes where a high number of Newton-Schulz iterative steps are used. We show that in fact the biased estimation of the polar factor introduced by AOL preconditioning still ensures that we recover a weight update that corresponds to a steepest descent direction. For reference, as explained in Bernstein & Newhouse (2024), the steepest descent update in spectral norm is (informally) defined as:

$$\arg\min_{\Delta W} \left[\langle G, \Delta W \rangle + \frac{\lambda}{2} \|\Delta W\|_2^2 \right] = \text{PolarFactor}(G). \tag{16}$$

with G the gradient and λ sharpness. Therefore, given that AOL preconditioning can be written as $\text{AOL}(\mathbf{X}) = \mathbf{X}s$ with s as vector with $s_i > 0$, using Equation 3. We have that the Turbo-Muon update

Method	t_{NS}	Accuracy (%)	$\varepsilon_{\text{approx}}$	$\varepsilon_{\text{bias}}$
Turbo-Muon	15	$94.08 \pm 0.1\%$	5.5×10^{-4}	0.1
Turbo-Muon	25	$94.07 \pm 0.1\%$	3.6×10^{-4}	0.1
Turbo-Muon	40	$94.05 \pm 0.1\%$	2.1×10^{-4}	0.1
Muon	15	$94.06 \pm 0.1\%$	1.5×10^{-2}	0.0
Muon	25	$94.06 \pm 0.1\%$	1.1×10^{-2}	0.0
Muon	40	$94.04 \pm 0.1\%$	7.4×10^{-3}	0.0

Table 6: **The measured $\varepsilon_{\text{bias}}$ does not impact final performance:** We measure the final accuracy (%) of the trained classifier on the CIFAR-10 dataset with and without AOL preconditioning after t_{NS} steps. We log the values of $\varepsilon_{\text{approx}}$ and $\varepsilon_{\text{bias}}$ once every 10 train batches and report average values across 80 training runs.

solves a steepest descent in an induced norm that is dependent of s (c.f. Definition 1 of Bernstein & Newhouse (2024)). Moreover, in practice, s forms a vector of strictly positive entries. Therefore, *the weight updates of Turbo-Muon yield a strict descent direction*. This means that even the worst case of an input that maximizes $\varepsilon_{\text{bias}}$ cannot lead to a divergence of the training process. More details are provided in Appendix A.2.

Confirming That the Estimation Bias Does Not Affect Training In order to empirically validate the theoretical result asserting that this estimation bias should not alter the training convergence, we investigate the training dynamics for an unconventionally high number of Newton-Schulz iterative steps. We train a neural network on the CIFAR-10 dataset with and without AOL preconditioning. Across each run, we vary the number of iterative NS steps and compute coefficients for the NS iterations according to the number of chosen steps (using the method from Amsel et al. (2025)). The high number of steps we choose here leads to an unrealistically long training time, but ensures that the orthogonalized weight updates are extremely close estimates of $\mathrm{PolarFactor}(\mathbf{X})$ and $\mathrm{PolarFactor}(\mathrm{AOL}(\mathbf{X}))$. Table 6 depicts the results for $15 \leq t_{\text{NS}} \leq 40$, and shows that while both Turbo-Muon and Muon converge toward their respective iterative objective, the bias induced by the AOL-preconditioning has no impact on the quality of training.

Since we have shown in Sec. 5 that Turbo-Muon yields steepest descent directions in an induced norm based on $\mathrm{PolarFactor}(GS)$, training a neural network with Turbo-Muon 's weight updates is valid in theory and in practice. Furthermore, usual practical implementations of orthogonality-based optimization rely on a relatively small amount of iterative steps to retrieve an approximation of the closest orthogonal matrix, where AOL preconditioning really shines.