CAN NORMALIZATION MATCH MUON AT SCALE?

Anonymous authors
Paper under double-blind review

ABSTRACT

Muon is increasingly replacing AdamW to train large-scale neural networks, but it requires expensive matrix orthogonalization. Recent studies report gains over Muon with simple row or column normalization alone. However, we find that normalization does not match Muon at scale. Our analysis shows that normalization leaves large correlations across rows, while Muon removes them. We also study how normalization should be combined with Muon. Applying normalization after Muon is more effective than applying it before, while using EMA statistics for normalization provides little benefit. The improvement over vanilla Muon decreases at larger scales. We establish these findings in the overtrained regime of frontier production language models. Our models are trained at 37–110$\times$ the Chinchilla optimum, whereas NanoGPT-style benchmarks commonly used in recent work are close to 1$\times$ Chinchilla. Experiments across vision tasks (*e.g.*, supervised learning, self-supervised learning, diffusion models) lead to similar conclusions.

1 INTRODUCTION

For a decade, AdamW (Kingma & Ba, 2015; Loshchilov & Hutter, 2019) has been the default optimizer to train large neural networks. Modern deep learning models (*e.g.*, GPT-3 (Brown et al., 2020), Vision Transformer (Dosovitskiy et al., 2021), AlphaFold (Jumper et al., 2021)) all rely on this. Muon (Jordan et al., 2024b) is now challenging this default. Since Liu et al. (2025) showed that Muon can scale to billion-parameter language models trained on trillions of tokens, it has been adopted in frontier production models (Kimi Team, 2026; Zeng et al., 2026; DeepSeek-AI, 2026).

Different from AdamW, Muon accumulates momentum from the gradients of each weight matrix and orthogonalizes it with Newton–Schulz iterations (Kovarik, 1970; Björck & Bowie, 1971). Because orthogonalization is a whole-matrix operation, it complicates distributed training, where weight matrices are sharded across devices. As a result, Muon introduces communication overhead (Liu et al., 2025; Khaled et al., 2026) and requires specialized techniques (Ahn et al., 2025).

This cost has motivated much simpler normalization methods. Instead of orthogonalizing the full matrix, one can rescale each row of the momentum to have the same ℓ_2 norm (Deng et al., 2026; Glentis et al., 2026; Xu et al., 2026), apply the same operation to each column (Xu et al., 2026), or alternate between row and column normalization (Scetbon et al., 2025). Normalization has also been combined directly with Muon, either before Muon (Ma et al., 2025; Chang et al., 2026; Liu et al., 2026), after it (Li et al., 2026c; Zhang et al., 2026a;b), or by alternating between the two (Dewulf et al., 2026). These methods have been shown to match or outperform Muon.

However, these results were obtained under different settings and much shorter training regimes than those in frontier production models. Therefore, it is unclear which conclusions hold at scale. In this paper, we conduct a comprehensive evaluation and study two simple questions: (1) *Can normalization replace Muon at scale?* and (2) *How should normalization be combined with Muon?*

We observe that normalization does not match Muon at scale. Row, column, and alternating normalization consistently fall behind Muon and perform similarly to one another. The gap emerges early in training and remains throughout the entire training run. Moreover, normalization converges much faster initially than AdamW in some tasks, but this advantage disappears later.

To understand this gap, we analyze the geometry of the updates induced by these methods. Normalization equalizes individual vector norms but leaves correlations across rows, whereas Muon removes these correlations. We observe that large correlations exist after normalization and persist through

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.

training. We further prove that for wide momentum matrices with full rank, row normalization is equal to Muon's orthogonalization if and only if the rows are mutually orthogonal.

Nevertheless, normalization could be beneficial when combined with Muon. We find that the ordering matters: applying normalization after Muon produces larger gains over applying it beforehand. However, the gain decreases with scale, and performance matches vanilla Muon eventually. Moreover, using an exponential moving average (EMA) for normalization shows no consistent benefit.

We test how our findings hold across scale and tasks. For language modeling, we target the overtraining regime of frontier production models. We train a model with 140M parameters on 105B tokens for many controlled experiments, and further validate our results at 950M parameters and 2.1T tokens. These training budgets (*i.e.*, 37–110$\times$ the Chinchilla optimum (Hoffmann et al., 2022)) extend well beyond small-scale NanoGPT-style benchmarks (Jordan et al., 2024a) commonly used in current optimizer research. We also evaluate methods on three vision tasks (*e.g.*, supervised learning, self-supervised learning, diffusion models) to confirm our conclusions.

In summary, our results demonstrate that Muon remains better than normalization at scale. The gains from combining normalization with Muon also decrease with scale. More broadly, our study highlights the importance of evaluating optimizers across training regimes (*i.e.*, different degrees of overtraining) and diverse tasks (*e.g.*, language, vision). Such comprehensive evaluation provides a more reliable basis to understand optimizer behavior and guide future optimizer design.

2 Optimization Methods

We denote t as the optimization step and $\boldsymbol{W} \in \mathbb{R}^{m\times n}$ as a weight matrix, where m and n denote the output and input dimensions, respectively. Let $\boldsymbol{G}_t$ be its gradient at step t. We use momentum with coefficient μ, $\boldsymbol{M}_t = \mu\boldsymbol{M}_{t-1} + \boldsymbol{G}_t$. The four methods we study differ only in the map $\mathcal{N}$ applied to the momentum. With decoupled weight decay (Loshchilov & Hutter, 2019), their updates are

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1} - \eta\alpha\mathcal{N}(\boldsymbol{M}_t) - \eta\lambda\boldsymbol{W}_{t-1}, \tag{1}$$

where η is the learning rate, λ is the weight decay strength, and α is a fixed scaling factor.

Matrix methods. Table 1 summarizes the four methods we study. RowNorm (Deng et al., 2026; Glentis et al., 2026) rescales each row (*i.e.*, along the input dimension) to the same ℓ_2 norm, while ColNorm does the same for each column (*i.e.*, along the output dimension). SinkGD (Scetbon et al., 2025) alternates RowNorm and ColNorm for multiple rounds, progressively balancing both dimensions. Muon (Jordan et al., 2024b) uses Newton–Schulz iterations (Kovarik, 1970; Björck & Bowie, 1971) to approximate matrix orthogonalization, flattening the nonzero singular values. Appendix A gives the pseudocode of each method. Section 7 discusses more on matrix optimizers.

Update scale. We define each map so that its output has unit per-element RMS $\|\mathcal{N}(\cdot)\|_F/\sqrt{mn} = 1$. Then α directly controls the RMS scale of each update. Following Liu et al. (2025), we extensively tune the learning rate for Muon and *directly transfer* its optimal value to other methods. We then empirically measure AdamW's update scale and find that it remains between 0.17 and 0.23 across weight matrices throughout training. So we set $\alpha = 0.2$ for all methods.

```
def row_norm(X):
  m, n = X.shape
  r = norm(X, dim=1)
  X = X / r[:, None]
  X *= sqrt(n)
  return X
```

(a) RowNorm*

```
def col_norm(X):
  m, n = X.shape
  c = norm(X, dim=0)
  X = X / c[None, :]
  X *= sqrt(m)
  return X
```

(b) ColNorm†

```
def sinkgd(X):
  for _ in range(K):
    X = row_norm(X)
    X = col_norm(X)
  return X
```

(c) SinkGD‡

```
def muon(X):
  m, n = X.shape
  X = newton_schulz(X)
  X *= sqrt(max(m, n))
  return X
```

(d) Muon§

Table 1: **Four methods studied in our experiments.** Each method is specified by a map $\mathcal{N}$. For an $m \times n$ matrix, (a) RowNorm normalizes each row along the input dimension, (b) ColNorm normalizes each column along the output dimension, (c) SinkGD alternates the two operations for K rounds, and (d) Muon approximately orthogonalizes the matrix using Newton–Schulz iterations. Each map is rescaled to unit per-element RMS, giving rise to the factors $\sqrt{n}$, $\sqrt{m}$, and $\sqrt{\max(m,n)}$, respectively. References: *Glentis et al. (2026), Deng et al. (2026), Yuan et al. (2026); †Xu et al. (2026); ‡Scetbon et al. (2025); and §Jordan et al. (2024b).

Parameter assignment. We apply every matrix method to the weight matrices inside Transformer blocks (*e.g.*, attention, MLP layers). Everything else is optimized with AdamW. This includes matrices outside the blocks (*e.g.*, embedding layer, output head) and one-dimensional parameters (*e.g.*, biases, learnable weights of normalization layers). Since α matches the update RMS of the matrix methods to that of AdamW, both groups share one base learning rate and one schedule. Following Kimi Team (2026), we further partition the query, key, and value projections along the head dimension and apply the map to each head separately. Appendix B provides per-task details.

3 Experiment Setup

We aim to study whether normalization can replace Muon and, if not, how the two operations are best combined. This section describes the tasks we evaluate on. More settings are in Appendix B.

3.1 Language Modeling

Recent optimizer studies (Li et al., 2026c; Deng et al., 2026) often use NanoGPT-style benchmarks or families of relatively small language models (*e.g.*, LLaMA architectures (Touvron et al., 2023)). These benchmarks use web corpora (*e.g.*, FineWeb (Penedo et al., 2024)) and scale the token number with model size, keeping the training budget near the Chinchilla optimum (Hoffmann et al., 2022).

However, frontier model families (Grattafiori et al., 2024; Qwen Team, 2025; Gemma Team, 2025) train their smaller models far beyond this compute-optimal allocation in order to improve quality at a fixed inference cost. For example, Llama 3.2 1B (Meta AI, 2024) is pre-trained on 9T tokens, several hundred times its Chinchilla-optimal budget. As Wen et al. (2026) show, this distinction matters: many optimizers that improve upon AdamW near the compute-optimal regime lose their advantage as the token-to-parameter ratio increases, with Muon among the few that retain it.

Training setup. Motivated by this, we benchmark all methods in the overtraining regime. We pre-train MobileLLM-R1 models with 140M and 950M parameters (Zhao et al., 2026) on 105B and 2.1T high-quality tokens. This corresponds to 37–110$\times$ the Chinchilla optimum. MobileLLM-R1 is a strong sub-billion model family: its released 950M base checkpoint outperforms Llama 3.2 1B.

We use the smaller 140M model to run extensive controlled studies, and the larger 950M model to test whether our findings hold at a larger scale. Our training recipe largely follows the original one and is held fixed across methods. For each experiment using 140M models, we run five seeds and report their mean, while the 950M model runs at a single seed only.

We report final validation loss on the test split of WikiText-2 (Merity et al., 2017). We also report zero-shot accuracy averaged over eight commonsense benchmarks: ARC-Easy and ARC-Challenge (Clark et al., 2018), BoolQ (Clark et al., 2019), HellaSwag (Zellers et al., 2019), OpenBookQA (Mihaylov et al., 2018), PIQA (Bisk et al., 2020), SIQA (Sap et al., 2019), and WinoGrande (Sakaguchi et al., 2020). We base our findings mainly on validation loss because it distinguishes between the studied methods much more clearly than accuracy. Accuracy is reported only for completeness.

3.2 Vision

We complement our language-modeling experiments with three representative vision tasks. As in language modeling, each task is run at two scales: a base model, where we run all controlled comparisons, and a larger model, used only for the final validation in Section 6.

Supervised learning. We train a Vision Transformer (Dosovitskiy et al., 2021) on ImageNet-1K classification (Deng et al., 2009) with a patch size of 16, using ViT-B as the base model and ViT-L as the larger model. We follow the training recipe from He et al. (2022). It provides a stronger supervised baseline than earlier ViT results (Dosovitskiy et al., 2021; Touvron et al., 2021). We report final training loss and top-1 validation accuracy. We find that the learning rate that minimizes training loss tends to overfit (*i.e.*, worse accuracy), so we tune the learning rate for lowest training loss and for highest accuracy separately and report the best of each.

Self-supervised learning. We evaluate masked autoencoders (MAE) (He et al., 2022), a widely used self-supervised method that learns transferable visual representations by reconstructing masked image patches. Following the standard self-supervised learning protocol, we pre-train on ImageNet-1K without labels and then fine-tune the full model with labels. We use ViT-B as the base model and ViT-L as the larger model, each with a patch size of 16. To isolate the effect of the optimizer during pre-training, we fine-tune every pre-trained checkpoint with AdamW. We report the training loss at the end of pre-training and top-1 validation accuracy after fine-tuning.

Diffusion models. We train a Diffusion Transformer (Peebles & Xie, 2023) on ImageNet-1K at 256×256 resolution. We use DiT-B as the base model and DiT-L as the larger model, each with a patch size of 4. Unlike the first two tasks, it is a generative one where the model learns to reverse a noise corruption process. We report the final training loss and the Fréchet Inception Distance (FID) (Heusel et al., 2017) against the standard ImageNet reference batch.

4 Can Normalization Match Muon?

We compare the three normalization methods against Muon and AdamW baselines on language modeling and three vision tasks. Table 2 shows the results.

	language		**supervised**†		**self-supervised**		**diffusion**	
	loss	change	loss	change	loss	change	loss	change
Muon	**3.0434**	–	**2.3616**	–	**0.4037**	–	**0.1548**	–
AdamW	3.0724	–	2.5041	–	0.4059	–	0.1558	–
RowNorm	3.0538	↑.0104	2.5176	↑.1560	0.4061	↑.0024	0.1556	↑.0008
ColNorm	3.0512	↑.0078	2.4999	↑.1383	0.4064	↑.0027	0.1557	↑.0009
SinkGD	3.0552	↑.0118	2.5224	↑.1608	0.4063	↑.0026	0.1554	↑.0006

(a) loss

	language		**supervised**†		**self-supervised**		**diffusion**	
	accuracy	change	accuracy	change	accuracy	change	FID	change
Muon	**45.43**	–	**82.49**	–	83.27	–	**60.08**	–
AdamW	44.90	–	82.21	–	83.41	–	69.17	–
RowNorm	44.91	↓0.52	82.14	↓0.35	**83.44**	↑0.17	67.45	↑7.37
ColNorm	45.10	↓0.33	82.07	↓0.42	83.23	↓0.04	67.74	↑7.66
SinkGD	45.08	↓0.35	82.11	↓0.38	83.37	↑0.10	65.86	↑5.78

(b) task metrics

Table 2: **Normalization methods compared with Muon and AdamW.** Muon achieves the lowest loss in every task and the best task metric in almost every task. The three normalization methods perform similarly to one another and sometimes underperform AdamW. †: loss and accuracy are each taken at their own best learning rate, so they may come from two different runs (Appendix B.2). The same protocol is used for Tables 3, 4, and 5.

Normalization does not match Muon at scale. Muon achieves the lowest loss in every task and also gives the best task-specific metric. The only exception is self-supervised learning, where RowNorm reaches 83.44 accuracy, slightly above Muon. In contrast, the normalization methods (*i.e.*, RowNorm, ColNorm, and SinkGD) generally fall behind Muon and perform very similarly to one another. Moreover, SinkGD provides little advantage over simple RowNorm or ColNorm. This suggests that progressively balancing rows and columns is not enough to bring the same level of benefit as Muon.

Figure 1 plots the loss curves of Muon, AdamW, and three normalization methods. Muon achieves the lowest final loss in every task and moves ahead of all other methods throughout training. Also, normalization methods converge much faster than AdamW early in training, but whether this advantage persists depends on the task. In language modeling, they outperform AdamW for the entire training run, and in diffusion they finish slightly ahead. In contrast, AdamW eventually overtakes these normalization methods in supervised and self-supervised learning.

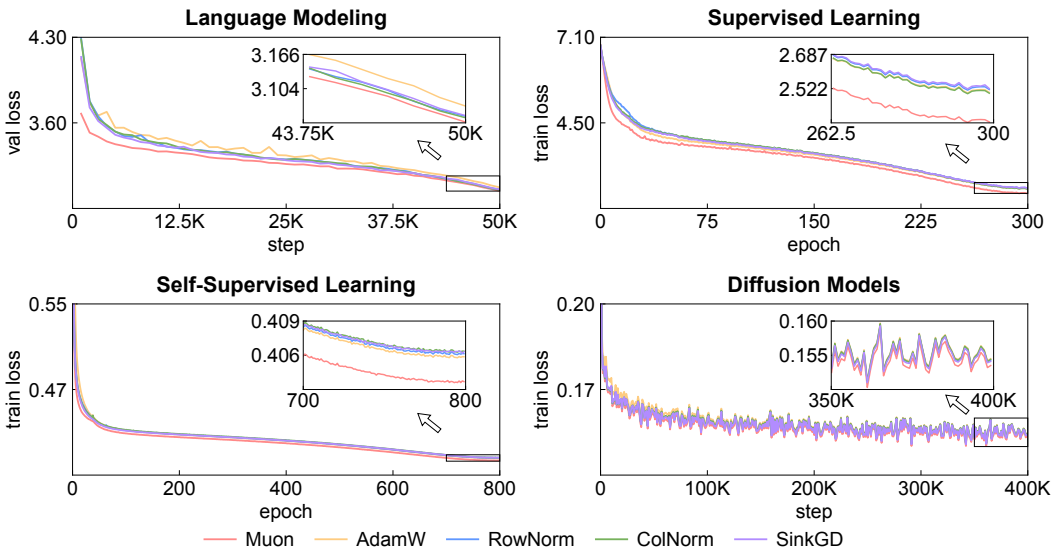

Figure 1: **Loss curves of Muon, AdamW, and three normalization methods.** Muon converges much faster than AdamW and three normalization methods. This trend is consistent throughout the entire training process.

5 Why Does Normalization Not Match Muon?

Previously, we show that normalization does not match Muon across different tasks at scale. This section provides a geometric analysis of each method's update that helps understand this gap.

Connection between Muon and normalization. Consider a momentum matrix $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ with $m \leq n$ and full row rank; when $m > n$, we apply the same analysis to $\boldsymbol{M}^\top$. Let $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$ be its compact singular value decomposition. Muon's orthogonalization, under our unit per-element RMS convention, is defined as $\mathcal{N}_{\text{Muon}}(\boldsymbol{M}) = \sqrt{n}\,\boldsymbol{U}\boldsymbol{V}^\top$. The Gram matrix of this update is

$$\frac{1}{n}\mathcal{N}_{\text{Muon}}(\boldsymbol{M})\mathcal{N}_{\text{Muon}}(\boldsymbol{M})^\top = \boldsymbol{U}\boldsymbol{V}^\top\boldsymbol{V}\boldsymbol{U}^\top = \boldsymbol{I}. \tag{2}$$

Thus, Muon produces an update whose rows have *equal norm and are mutually orthogonal*.

Let $\boldsymbol{D} = \text{diag}(\boldsymbol{M}\boldsymbol{M}^\top)$, where $\text{diag}(\cdot)$ keeps the diagonal entries and sets off-diagonal entries to 0. RowNorm can be written as $\mathcal{N}_{\text{RowNorm}}(\boldsymbol{M}) = \sqrt{n}\,\boldsymbol{D}^{-1/2}\boldsymbol{M}$. The Gram matrix of its update is

$$\frac{1}{n}\mathcal{N}_{\text{RowNorm}}(\boldsymbol{M})\mathcal{N}_{\text{RowNorm}}(\boldsymbol{M})^\top = \boldsymbol{D}^{-1/2}\boldsymbol{M}\boldsymbol{M}^\top\boldsymbol{D}^{-1/2}. \tag{3}$$

The (i, j)-th entry of Equation 3 is

$$\left[\boldsymbol{D}^{-1/2}\boldsymbol{M}\boldsymbol{M}^\top\boldsymbol{D}^{-1/2}\right]_{ij} = \frac{\langle \boldsymbol{M}_{i,:}, \boldsymbol{M}_{j,:}\rangle}{\|\boldsymbol{M}_{i,:}\|_2\|\boldsymbol{M}_{j,:}\|_2}, \tag{4}$$

where $\langle \mathbf{a}, \mathbf{b}\rangle = \mathbf{a}\mathbf{b}^\top$ is the Euclidean inner product. Thus, the Gram matrix is exactly the pairwise cosine similarity matrix of the rows of $\boldsymbol{M}$: its diagonal entries are all one, while its off-diagonal entries quantify the alignment between different rows.

This provides a connection between normalization and Muon. For wide matrices, RowNorm $\mathcal{N}_{\text{RowNorm}}$ matches the diagonal of Muon's update Gram matrix, while Muon additionally removes all cross-row correlations. For tall matrices, applying the same analysis to $\boldsymbol{M}^\top$ yields ColNorm $\mathcal{N}_{\text{ColNorm}}$. Appendix C.2 further compares this shape-aligned normalization design in training.

Visualizing Gram matrices. We want to see how close the Gram matrix of the normalized update (Equation 3) is to the identity Gram matrix from Muon (Equation 2). We compute Equation 3 using momentum matrices saved from a Muon training trajectory. Figure 2 visualizes the resulting Gram

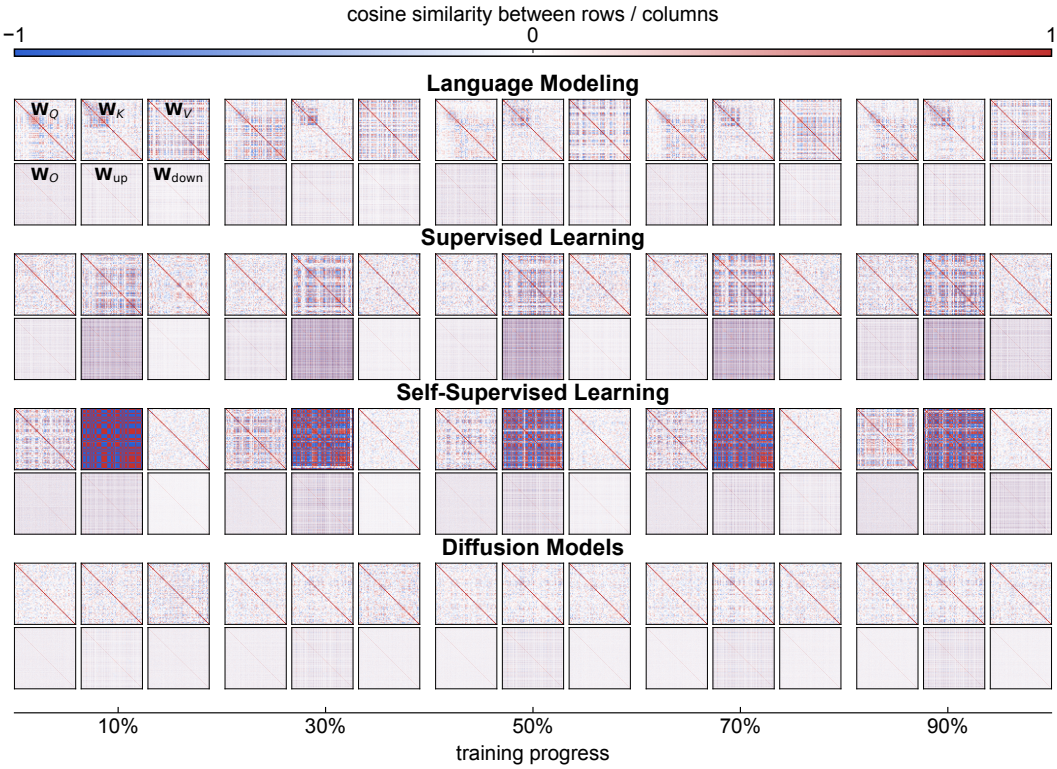

Figure 2: **Gram matrices of normalized updates.** We visualize Gram matrices (Equation 3) of updates applied by normalization. They show large values at off-diagonal entries of both signs (red for positive and blue for negative values). This contrasts with Muon, whose Gram matrix is exactly the identity (Equation 2). For each task, we visualize one attention head for the query, key, and value projections ($\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$), together with the full output and MLP matrices ($\mathbf{W}_O$, $\mathbf{W}_{\text{up}}$, and $\mathbf{W}_{\text{down}}$). The specific layer and head are randomly sampled for each matrix. Each draw is made once per task and held fixed across checkpoints.

matrices across tasks and training progress, including individual heads of the query, key, and value matrices ($\mathbf{W}_Q$, $\mathbf{W}_K$, and $\mathbf{W}_V$) and other full matrices in the Transformer ($\mathbf{W}_O$, $\mathbf{W}_{\text{up}}$, and $\mathbf{W}_{\text{down}}$).

Overall, the normalized Gram matrices are not close to the identity. Language modeling appears closest: $\mathbf{W}_O$, $\mathbf{W}_{\text{up}}$, and $\mathbf{W}_{\text{down}}$ show a clear diagonal with relatively small off-diagonal entries. However, large cross-row correlations remain for other matrices and tasks, including $\mathbf{W}_Q$ in supervised learning and $\mathbf{W}_K$ in self-supervised learning and diffusion models. Moreover, these correlations persist from early to late training, so normalization can not approach Muon as training proceeds.

Exact equality between normalization and Muon. The above analysis shows that normalization retains nonzero cross-row correlations. The following theorem establishes that normalization is equal to Muon's orthogonalization if and only if all rows are mutually orthogonal.

Theorem 1. *Let $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ with $m \leq n$ and full row rank. Then*

$$\mathcal{N}_{\text{RowNorm}}(\boldsymbol{M}) = \mathcal{N}_{\text{Muon}}(\boldsymbol{M}) \tag{5}$$

if and only if

$$\boldsymbol{M}\boldsymbol{M}^\top = \text{diag}(\boldsymbol{M}\boldsymbol{M}^\top), \tag{6}$$

i.e., the rows of $\boldsymbol{M}$ are mutually orthogonal.

We prove Theorem 1 in Appendix C.2. The theorem gives the exact condition to recover Muon from normalization. In other words, normalization matches Muon only in the special case where the momentum already has mutually orthogonal rows. Since such correlations are non-negligible across tasks and matrix types (Figure 2), this condition does not hold in practice. Appendix C.3 further compares the singular value distributions of the resulting updates between normalization and Muon.

6 How to Combine Normalization with Muon?

From our analysis above, normalization is fundamentally different from Muon's orthogonalization. In this section, we test whether combining them improves results.

Ordering normalization and Muon. We here study where to apply a normalization $\mathcal{N}_{\text{Norm}}$ relative to Muon. Normalizing first, $\mathcal{N}_{\text{Muon}}(\mathcal{N}_{\text{Norm}}(\boldsymbol{M}_t))$, yields an orthogonalized update, whereas normalizing last, $\mathcal{N}_{\text{Norm}}(\mathcal{N}_{\text{Muon}}(\boldsymbol{M}_t))$, preserves the norm. Table 3 shows the results of this comparison.

	language		supervised†		self-supervised		diffusion	
	loss	change	loss	change	loss	change	loss	change
Muon	3.0434	–	2.3616	–	0.4037	–	0.1548	–
AdamW	3.0724	–	2.5041	–	0.4059	–	0.1558	–
applying before Muon								
RowNorm	3.0517	↑.0083	2.3592	↓.0024	0.4033	↓.0004	0.1546	↓.0002
ColNorm	3.0483	↑.0049	2.3538	↓.0078	0.4036	↓.0001	0.1547	↓.0001
SinkGD	3.0472	↑.0038	2.3618	↑.0002	**0.4032**	↓.0005	**0.1545**	↓.0003
applying after Muon								
RowNorm	**3.0372**	↓.0062	2.3565	↓.0051	0.4033	↓.0004	0.1546	↓.0002
ColNorm	3.0432	↓.0002	**2.3514**	↓.0102	0.4036	↓.0001	0.1546	↓.0002
SinkGD	3.0512	↑.0078	2.3543	↓.0073	0.4033	↓.0004	**0.1545**	↓.0003

(a) loss

	language		supervised†		self-supervised		diffusion	
	accuracy	change	accuracy	change	accuracy	change	FID	change
Muon	45.43	–	82.49	–	83.27	–	60.08	–
AdamW	44.90	–	82.21	–	**83.41**	–	69.17	–
applying before Muon								
RowNorm	45.46	↑0.03	82.50	↑0.01	83.31	↑0.04	59.21	↓0.87
ColNorm	45.39	↓0.04	82.58	↑0.09	83.23	↓0.04	58.18	↓1.90
SinkGD	**45.78**	↑0.35	82.37	↓0.12	83.28	↑0.01	**58.12**	↓1.96
applying after Muon								
RowNorm	45.09	↓0.34	**82.83**	↑0.34	83.39	↑0.12	58.70	↓1.38
ColNorm	45.61	↑0.18	82.73	↑0.24	83.10	↓0.17	58.59	↓1.49
SinkGD	45.61	↑0.18	82.56	↑0.07	83.32	↑0.05	58.39	↓1.69

(b) task metrics

Table 3: **Where to apply normalization.** Applying normalization after Muon yields larger gains than applying it beforehand. With this ordering, all three normalization methods improve on seven of the eight metrics.

Applying normalization after orthogonalization is generally more effective. This trend is most clear in language modeling and supervised learning, while the two orderings perform similarly in self-supervised learning and diffusion models. Among all configurations, using RowNorm after Muon performs the best. It achieves the lowest loss in language modeling and improves over vanilla Muon on seven of the eight metrics. Using SinkGD after Muon is similarly strong.

Accumulating statistics with EMA. The normalization statistics can also be estimated using an exponential moving average (EMA). This approach has been adopted in prior work (Zhang et al., 2026a; Li et al., 2026c). Specifically, we keep a running average of the row norms of Muon's update

$$\boldsymbol{s}_t = \beta\, \boldsymbol{s}_{t-1} + (1-\beta)\, \boldsymbol{r}\big(\mathcal{N}_{\text{Muon}}(\boldsymbol{M}_t)\big), \tag{7}$$

where $\boldsymbol{r}(\boldsymbol{X})_i = \|\boldsymbol{X}_{i,:}\|_2$ and $\beta = 0.95$. We then use the update $\text{diag}(\boldsymbol{s}_t)^{-1}\mathcal{N}_{\text{Muon}}(\boldsymbol{M}_t)$. ColNorm is analogous over columns. Since using EMA statistics no longer fixes the update scale, we rescale the update to unit per-element RMS. We compare it with simple normalization under the better ordering above (*i.e.*, applying normalization after Muon). Table 4 reports the results.

EMA statistics improve over vanilla Muon but bring no consistent benefit over simple normalization. Across all metrics, EMA improves five comparisons, degrades seven, and ties the rest. As EMA also

	language		supervised†		self-supervised		diffusion	
	loss	change	loss	change	loss	change	loss	change
Muon	3.0434	–	2.3616	–	0.4037	–	0.1548	–
AdamW	3.0724	–	2.5041	–	0.4059	–	0.1558	–
simple normalization								
RowNorm	**3.0372**	↓.0062	2.3565	↓.0051	**0.4033**	↓.0004	**0.1546**	↓.0002
ColNorm	3.0432	↓.0002	**2.3514**	↓.0102	0.4036	↓.0001	**0.1546**	↓.0002
EMA statistics								
RowNorm	3.0410	↓.0024	2.3538	↓.0078	**0.4033**	↓.0004	**0.1546**	↓.0002
ColNorm	3.0392	↓.0042	2.3518	↓.0098	0.4036	↓.0001	**0.1546**	↓.0002

(a) loss

	language		supervised†		self-supervised		diffusion	
	accuracy	change	accuracy	change	accuracy	change	FID	change
Muon	45.43	–	82.49	–	83.27	–	60.08	–
AdamW	44.90	–	82.21	–	**83.41**	–	69.17	–
simple normalization								
RowNorm	45.09	↓0.34	**82.83**	↑0.34	83.39	↑0.12	58.70	↓1.38
ColNorm	45.61	↑0.18	82.73	↑0.24	83.10	↓0.17	58.59	↓1.49
EMA statistics								
RowNorm	**45.64**	↑0.21	82.58	↑0.09	83.21	↓0.06	**58.57**	↓1.51
ColNorm	45.30	↓0.13	82.48	↓0.01	83.31	↑0.04	58.61	↓1.47

(b) task metrics

Table 4: **Normalizing with simple or EMA statistics.** Accumulating normalization statistics with EMA provides little or no gain over simple normalization. All experiments here use normalization after Muon.

needs extra optimizer state, we use simple normalization for subsequent large-scale experiments. The same trend holds when normalization is applied before Muon (see Appendix C.1).

Large-scale validation. All comparisons so far use the 140M language model and the vision models with base size. Here we validate whether our findings hold at larger scales. We test the strongest combinations with Muon (*i.e.*, Muon + RowNorm and Muon + SinkGD). Table 5 shows the results.

The benefit of combining normalization with Muon becomes weaker at scale. In language modeling, applying normalization after Muon is slightly worse than vanilla Muon, and in vision it brings only small improvements. The loss curves in Figure 3 show the same trend: in language modeling, Muon and its combinations with normalization are nearly indistinguishable throughout training, while in vision the combinations end marginally below vanilla Muon in most cases.

	language		supervised†		self-supervised		diffusion	
	loss	change	loss	change	loss	change	loss	change
Muon	**2.3166**	–	2.0915	–	0.3909	–	0.1515	–
AdamW	2.3344	–	2.2565	–	0.3943	–	0.1528	–
applying after Muon								
RowNorm	2.3214	↑.0048	2.0944	↑.0029	0.3907	↓.0002	0.1514	↓.0001
SinkGD	2.3197	↑.0031	**2.0884**	↓.0031	**0.3905**	↓.0004	**0.1513**	↓.0002

(a) loss

	language		supervised†		self-supervised		diffusion	
	accuracy	change	accuracy	change	accuracy	change	FID	change
Muon	**57.64**	–	82.35	–	85.60	–	**36.25**	–
AdamW	57.04	–	80.92	–	85.56	–	44.87	–
applying after Muon								
RowNorm	56.62	↓1.02	82.37	↑0.02	85.52	↓0.08	37.09	↑0.84
SinkGD	56.48	↓1.16	**82.53**	↑0.18	**85.64**	↑0.04	36.27	↑0.02

(b) task metrics

Table 5: **Validating findings at larger scales.** At larger scales, applying normalization after Muon shows no gain in language modeling and only marginal gains on vision tasks.

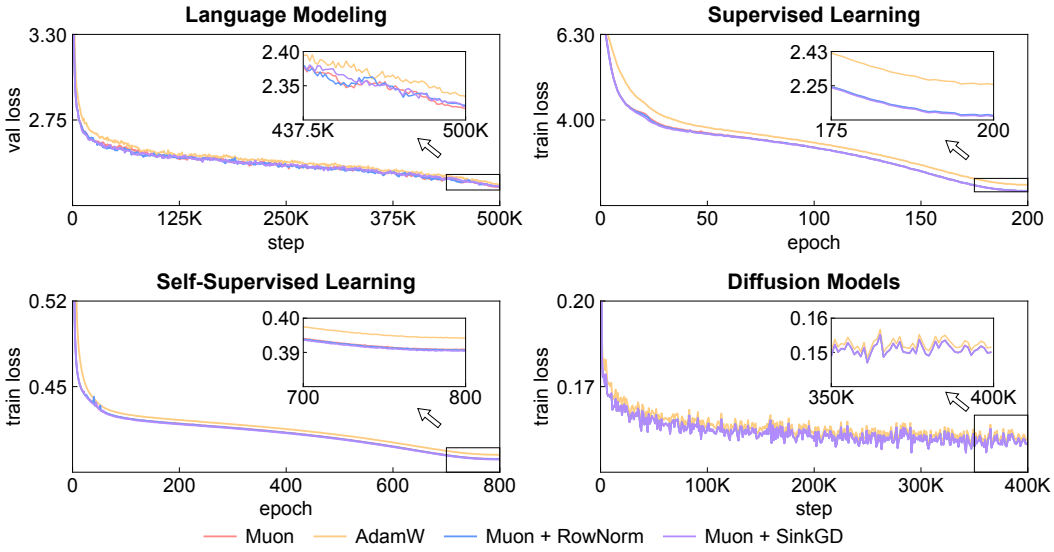

Figure 3: **Loss curves for large-scale experiments.** Applying normalization after Muon yields very similar loss to vanilla Muon throughout training across four tasks, while AdamW stays behind.

7 Related Work

Matrix-based optimizers. Prior work has exploited the matrix structure of neural network weights, including spectral descent (Carlson et al., 2015a;b; 2016), second-order methods that approximate curvature (Martens & Grosse, 2015; George et al., 2018), and preconditioners accumulated from past gradients (Gupta et al., 2018; Morwani et al., 2025; Vyas et al., 2025). More recently, Jordan et al. (2024b) introduce Muon to orthogonalize the momentum matrix using Newton–Schulz iterations (Kovarik, 1970; Björck & Bowie, 1971). Muon has since been used to train models with billions of parameters on trillions of tokens (Kimi Team, 2026; Zeng et al., 2026; DeepSeek-AI, 2026).

Normalization optimizers. There is also a cheaper family of matrix optimizers normalizing updates along one axis at a time. RowNorm (Glentis et al., 2026; Deng et al., 2026; Yuan et al., 2026) normalizes each row, ColNorm normalizes each column, and SinkGD (Scetbon et al., 2025) alternates between the two operations. Xu et al. (2026) also derive RowNorm and ColNorm from the steepest descent framework (Bernstein & Newhouse, 2024) under different choices of matrix operator norm. These methods also differ in whether normalization is applied directly to the current gradient (Scetbon et al., 2025; Glentis et al., 2026), to the momentum (Deng et al., 2026; Yuan et al., 2026; Gu & Xie, 2026), or to the updates from another optimizer (Wang et al., 2025).

Combining normalization with Muon. Normalization has also been combined with Muon's orthogonalization. It can be used before orthogonalization (Ma et al., 2025; Chang et al., 2026; Liu et al., 2026; Li et al., 2026b), after orthogonalization (Zhang et al., 2026b), or iteratively alternated with orthogonalization (Dewulf et al., 2026). Other methods combine Muon with normalization statistics accumulated across steps (Zhang et al., 2026a; Li et al., 2026c). Orthogonalization can also be applied periodically with RowNorm used on most steps (Li et al., 2026a).

8 Discussion and Conclusion

Our study shows that normalization and Muon's orthogonalization play fundamentally different roles in matrix optimization. Normalization equalizes the magnitudes of individual vectors but leaves their correlations. In contrast, Muon drives these correlations to zero. This difference matters at scale: normalization alone falls behind Muon, whereas applying it after Muon can sometimes help.

Moreover, our results suggest that optimizer designs should be evaluated beyond small-scale benchmarks. Methods that perform better in short runs can lose their advantage as model and training scale increase. Therefore, establishing whether an optimizer improvement transfers requires rigorous evaluation across training horizons, model sizes, and diverse tasks.

AI use statement. We use generative AI tools to refine the paper and optimize the visual layouts of figures and tables. We also use AI to implement optimization algorithms and help execute experiments.

Reproducibility statement. We will release the entire codebase to reproduce all experiments in the paper upon acceptance.

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.

Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. *arXiv preprint arXiv:2409.20325*, 2024.

Yonatan Bisk, Rowan Zellers, Ronan Le Bras, Jianfeng Gao, and Yejin Choi. PIQA: Reasoning about physical commonsense in natural language. In *AAAI*, 2020.

Åke Björck and Clazett Bowie. An iterative algorithm for computing the best estimate of an orthogonal matrix. *SIAM Journal on Numerical Analysis*, 1971.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *NeurIPS*, 2020.

David Carlson, Volkan Cevher, and Lawrence Carin. Stochastic spectral descent for restricted Boltzmann machines. In *AISTATS*, 2015a.

David Carlson, Edo Collins, Ya-Ping Hsieh, Lawrence Carin, and Volkan Cevher. Preconditioned spectral descent for deep learning. In *NeurIPS*, 2015b.

David Carlson, Ya-Ping Hsieh, Edo Collins, Lawrence Carin, and Volkan Cevher. Stochastic spectral descent for discrete graphical models. *IEEE Journal of Selected Topics in Signal Processing*, 2016.

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

Da Chang, Qiankun Shi, Lvgang Zhang, Yu Li, Ruijie Zhang, Yao Lu, et al. MuonEq: Balancing before orthogonalization with lightweight equilibration. *arXiv preprint arXiv:2603.28254*, 2026.

Christopher Clark, Kenton Lee, Ming-Wei Chang, Tom Kwiatkowski, Michael Collins, and Kristina Toutanova. BoolQ: Exploring the surprising difficulty of natural yes/no questions. In *NAACL*, 2019.

Peter Clark, Isaac Cowhey, Oren Etzioni, Tushar Khot, Ashish Sabharwal, Carissa Schoenick, and Oyvind Tafjord. Think you have solved question answering? try ARC, the AI2 reasoning challenge. *arXiv preprint arXiv:1803.05457*, 2018.

Ekin D Cubuk, Barret Zoph, Jonathon Shlens, and Quoc V Le. RandAugment: Practical automated data augmentation with a reduced search space. In *NeurIPS*, 2020.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009.

Shenyang Deng, Zhuoli Ouyang, Tianyu Pang, Zihang Liu, Ruochen Jin, Shuhua Yu, and Yaoqing Yang. RMNP: Row-momentum normalized preconditioning for scalable matrix-based optimization. In *ICML*, 2026.

Alec Dewulf, Dhruv Pai, Li Yang, Ashley Zhang, and Ben Keigwin. Aurora: A leverage-aware spectral optimizer. *arXiv preprint arXiv:2606.27715*, 2026.

Prafulla Dhariwal and Alexander Nichol. Diffusion models beat GANs on image synthesis. In *NeurIPS*, 2021.

Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, et al. An image is worth 16x16 words: Transformers for image recognition at scale. In *ICLR*, 2021.

Gemma Team. Gemma 3 technical report. *arXiv preprint arXiv:2503.19786*, 2025.

Thomas George, César Laurent, Xavier Bouthillier, Nicolas Ballas, and Pascal Vincent. Fast approximate natural gradient descent in a Kronecker factored eigenbasis. In *NeurIPS*, 2018.

Athanasios Glentis, Jiaxiang Li, Andi Han, and Mingyi Hong. Memory-efficient LLM pretraining via minimalist optimizer design. In *ICML*, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, Amy Yang, et al. The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Yufei Gu and Zeke Xie. Mano: Restriking manifold optimization for LLM training. *arXiv preprint arXiv:2601.23000*, 2026.

Vineet Gupta, Tomer Koren, and Yoram Singer. Shampoo: Preconditioned stochastic tensor optimization. In *ICML*, 2018.

Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, 2022.

Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. GANs trained by a two time-scale update rule converge to a local Nash equilibrium. In *NeurIPS*, 2017.

Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS Workshops*, 2021.

Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. In *NeurIPS*, 2020.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, et al. Training compute-optimal large language models. In *NeurIPS*, 2022.

Gao Huang, Yu Sun, Zhuang Liu, Daniel Sedra, and Kilian Q Weinberger. Deep networks with stochastic depth. In *ECCV*, 2016.

Keller Jordan, Jeremy Bernstein, Brendan Rappazzo, @fernbear.bsky.social, Boza Vlado, You Jiacheng, Franz Cesista, Braden Koszarsky, and @Grad62304977. modded-nanogpt: Speedrunning the NanoGPT baseline, 2024a. URL https://github.com/KellerJordan/modded-nanogpt.

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

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin Žídek, Anna Potapenko, et al. Highly accurate protein structure prediction with AlphaFold. *nature*, 2021.

Ahmed Khaled, Kaan Ozkara, Tao Yu, Mingyi Hong, and Youngsuk Park. MuonBP: Faster Muon via block-periodic orthogonalization. In *ICLR*, 2026.

Kimi Team. Kimi K3: Open frontier intelligence. *arXiv preprint arXiv:2607.24653*, 2026.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *ICLR*, 2015.

Zdislav Kovarik. Some iterative methods for improving orthonormality. *SIAM Journal on Numerical Analysis*, 1970.

Chenghao Li, Xiao Han, Xinxin Huang, Wei Liu, Boyang Li, Bing Xiao, et al. Scaling Muon for diffusion transformers. *arXiv preprint arXiv:2608.20818*, 2026a.

Jingru Li, Yibo Fan, and Huan Li. Variance-adaptive muon: Pre-orthogonalization variance modulation for efficient language model pretraining. *arXiv preprint arXiv:2601.14603*, 2026b.

Zichong Li, Liming Liu, Chen Liang, Weizhu Chen, and Tuo Zhao. NorMuon: Making Muon more efficient and scalable. In *ICML*, 2026c.

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.

Ziyue Liu, Ruijie Zhang, Zhengyang Wang, Yequan Zhao, Yupeng Su, Zi Yang, et al. Muon^2: Boosting Muon via adaptive second-moment preconditioning. *arXiv preprint arXiv:2604.09967*, 2026.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Chao Ma, Wenbo Gong, Meyer Scetbon, and Edward Meeds. SWAN: SGD with normalization and whitening enables stateless LLM training. In *ICML*, 2025.

James Martens and Roger Grosse. Optimizing neural networks with Kronecker-factored approximate curvature. In *ICML*, 2015.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. In *ICLR*, 2017.

Meta AI. Llama 3.2: Revolutionizing edge AI and vision with open, customizable models, 2024. URL https://ai.meta.com/blog/llama-3-2-connect-2024-vision-edge-mobile-devices/.

Todor Mihaylov, Peter Clark, Tushar Khot, and Ashish Sabharwal. Can a suit of armor conduct electricity? a new dataset for open book question answering. In *EMNLP*, 2018.

Depen Morwani, Itai Shapira, Nikhil Vyas, Eran Malach, Sham Kakade, and Lucas Janson. A new perspective on Shampoo's preconditioner. In *ICLR*, 2025.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The FineWeb datasets: Decanting the web for the finest text data at scale. In *NeurIPS Datasets and Benchmarks Track*, 2024.

Qwen Team. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

Keisuke Sakaguchi, Ronan Le Bras, Chandra Bhagavatula, and Yejin Choi. WinoGrande: An adversarial Winograd schema challenge at scale. In *AAAI*, 2020.

Maarten Sap, Hannah Rashkin, Derek Chen, Ronan Le Bras, and Yejin Choi. Social IQa: Commonsense reasoning about social interactions. In *EMNLP*, 2019.

Meyer Scetbon, Chao Ma, Wenbo Gong, and Ted Meeds. Gradient multi-normalization for efficient LLM training. In *NeurIPS*, 2025.

Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the Inception architecture for computer vision. In *CVPR*, 2016.

Hugo Touvron, Matthieu Cord, Matthijs Douze, Francisco Massa, Alexandre Sablayrolles, and Hervé Jégou. Training data-efficient image transformers & distillation through attention. In *ICML*, 2021.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, et al. LLaMA: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

Nikhil Vyas, Depen Morwani, Rosie Zhao, Itai Shapira, David Brandfonbrener, Lucas Janson, and Sham Kakade. SOAP: Improving and stabilizing Shampoo using Adam for language modeling. In *ICLR*, 2025.

Junjie Wang, Pan Zhou, Yiming Dong, Huan Li, Jia Li, Xun Zhou, et al. Conda: Column-normalized Adam for training large language models faster. *arXiv preprint arXiv:2509.24218*, 2025.

Kaiyue Wen, David Hall, Tengyu Ma, and Percy Liang. Fantastic pretraining optimizers and where to find them. In *ICLR*, 2026.

Ruihan Xu, Jiajin Li, and Yiping Lu. On the width scaling of neural optimizers under matrix operator norms I: Row/column normalization and hyperparameter transfer. *arXiv preprint arXiv:2603.09952*, 2026.

Jinghui Yuan, Jiaxuan Zou, Shuo Wang, Yong Liu, and Feiping Nie. Nora: Normalized orthogonal row alignment for scalable matrix optimizer. *arXiv preprint arXiv:2605.03769*, 2026.

Sangdoo Yun, Dongyoon Han, Seong Joon Oh, Sanghyuk Chun, Junsuk Choe, and Youngjoon Yoo. CutMix: Regularization strategy to train strong classifiers with localizable features. In *ICCV*, 2019.

Rowan Zellers, Ari Holtzman, Yonatan Bisk, Ali Farhadi, and Yejin Choi. HellaSwag: Can a machine really finish your sentence? In *ACL*, 2019.

Aohan Zeng, Xin Lv, Zhenyu Hou, Zhengxiao Du, Qinkai Zheng, Bin Chen, Da Yin, Chendi Ge, Chenghua Huang, Chengxing Xie, et al. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. In *ICLR*, 2018.

Minxin Zhang, Yuxuan Liu, and Hayden Schaeffer. Adam improves Muon: Adaptive moment estimation with orthogonalized momentum. *arXiv preprint arXiv:2602.17080*, 2026a.

Ruijie Zhang, Yequan Zhao, Ziyue Liu, Zhengyang Wang, Yupeng Su, Liyan Tan, et al. MUON+: Towards more effective Muon via one additional normalization step for LLM pre-training. *arXiv preprint arXiv:2602.21545*, 2026b.

Changsheng Zhao, Ernie Chang, Zechun Liu, Chia-Jung Chang, Wei Wen, Chen Lai, Sheng Cao, Yuandong Tian, et al. MobileLLM-R1: Exploring the limits of sub-billion language model reasoners with open training recipes. In *ICLR*, 2026.

Zhun Zhong, Liang Zheng, Guoliang Kang, Shaozi Li, and Yi Yang. Random erasing data augmentation. In *AAAI*, 2020.

A Optimization Methods

This appendix gives the implementation details of the four methods. It provides the pseudocode of one optimizer step for each method and also derives that each map has unit per-element RMS.

Pseudocode. Algorithms 1–4 give one full optimizer step of each method for a single matrix. The four methods share the momentum, weight decay, and update scale α, and differ only in the map applied to the momentum, each of which returns an update with unit per-element RMS. We use $\mu = 0.95$, $\alpha = 0.2$, and $K = 5$ for alternating rounds in SinkGD. For Muon, we run five Newton–Schulz steps with per-step coefficients tuned by Cesista et al. (2025). It is reported with better orthogonalization than the fixed coefficients used by Jordan et al. (2024b). In the optimizer step, weights, gradients, and momentum are kept in float32, and only Newton–Schulz runs in bfloat16.

Algorithm 1 RowNorm

```
# W: weight [m, n]; G: gradient; M: momentum buffer, same shape as W
# mu: momentum; alpha: update RMS; lr: learning rate; wd: weight decay
def rownorm_step(W, G, M, lr, wd):
    m, n = W.shape
    M = mu * M + G
    r = norm(M, dim=1)
    U = sqrt(n) * M / r[:, None] # each row to l2 norm sqrt(n)
    W = W - lr * alpha * U - lr * wd * W
    return W, M
```

Algorithm 2 ColNorm

```
# W: weight [m, n]; G: gradient; M: momentum buffer, same shape as W
# mu: momentum; alpha: update RMS; lr: learning rate; wd: weight decay
def colnorm_step(W, G, M, lr, wd):
    m, n = W.shape
    M = mu * M + G
    c = norm(M, dim=0)
    U = sqrt(m) * M / c[None, :] # each column to l2 norm sqrt(m)
    W = W - lr * alpha * U - lr * wd * W
    return W, M
```

Algorithm 3 SinkGD

```
# W: weight [m, n]; G: gradient; M: momentum buffer, same shape as W
# mu: momentum; alpha: update RMS; lr: learning rate; wd: weight decay
# K: number of alternating rounds
def sinkgd_step(W, G, M, lr, wd):
    m, n = W.shape
    M = mu * M + G
    U = M
    for _ in range(K): # alternate row and column normalization
        r = norm(U, dim=1)
        U = sqrt(n) * U / r[:, None]
        c = norm(U, dim=0)
        U = sqrt(m) * U / c[None, :]
    W = W - lr * alpha * U - lr * wd * W
    return W, M
```

Unit per-element RMS of each method. For a matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$, the per-element RMS is

$$\mathrm{RMS}(\boldsymbol{X}) = \frac{\|\boldsymbol{X}\|_F}{\sqrt{mn}} = \left(\frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} X_{ij}^2\right)^{1/2}. \tag{8}$$

We below show that every map returns an update with $\mathrm{RMS} = 1$, so that α in Equation 1 alone sets the update size and is shared across methods. Throughout, $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ denotes the momentum.

Algorithm 4 Muon

```
# W: weight [m, n]; G: gradient; M: momentum buffer, same shape as W
# mu: momentum; alpha: update RMS; lr: learning rate; wd: weight decay
# NS_COEFFS: tuned (a, b, c) of each of the five Newton-Schulz steps
NS_COEFFS = [(4.0848, -6.8946, 2.9270), (3.9505, -6.3029, 2.6377),
          (3.7418, -5.5913, 2.3037), (2.8769, -3.1427, 1.2046),
          (2.8366, -3.0525, 1.2012)]
def muon_step(W, G, M, lr, wd):
   m, n = W.shape
   M = mu * M + G
   X = M.bfloat16() # Newton-Schulz runs in bfloat16
   X = X.T if m > n else X # iterate on the wide orientation
   X = X / norm(X) # singular values <= 1
   for a, b, c in NS_COEFFS: # Newton-Schulz: push singular values to 1
      A = X @ X.T
      X = a * X + (b * A + c * A @ A) @ X
   X = X.T if m > n else X
   U = sqrt(max(m, n)) * X.float() # back to float32, rescaled to RMS 1
   W = W - lr * alpha * U - lr * wd * W
   return W, M
```

RowNorm rescales the i-th row of $\boldsymbol{M}$ to

$$[\mathcal{N}_{\text{RowNorm}}(\boldsymbol{M})]_{i,:} = \sqrt{n}\,\frac{\boldsymbol{M}_{i,:}}{\|\boldsymbol{M}_{i,:}\|_2}, \qquad \text{so} \qquad \left\|[\mathcal{N}_{\text{RowNorm}}(\boldsymbol{M})]_{i,:}\right\|_2^2 = n \tag{9}$$

for every i. Summing the squared row norms over the m rows gives

$$\|\mathcal{N}_{\text{RowNorm}}(\boldsymbol{M})\|_F^2 = \sum_{i=1}^{m} \left\|[\mathcal{N}_{\text{RowNorm}}(\boldsymbol{M})]_{i,:}\right\|_2^2 = mn, \tag{10}$$

and therefore, by Equation 8, $\text{RMS}(\mathcal{N}_{\text{RowNorm}}(\boldsymbol{M})) = \sqrt{mn}/\sqrt{mn} = 1$.

The argument is symmetric to RowNorm. ColNorm rescales the j-th column of $\boldsymbol{M}$ to

$$[\mathcal{N}_{\text{ColNorm}}(\boldsymbol{M})]_{:,j} = \sqrt{m}\,\frac{\boldsymbol{M}_{:,j}}{\|\boldsymbol{M}_{:,j}\|_2}, \qquad \text{so} \qquad \left\|[\mathcal{N}_{\text{ColNorm}}(\boldsymbol{M})]_{:,j}\right\|_2^2 = m \tag{11}$$

for every j. Summing over the n columns gives

$$\|\mathcal{N}_{\text{ColNorm}}(\boldsymbol{M})\|_F^2 = \sum_{j=1}^{n} \left\|[\mathcal{N}_{\text{ColNorm}}(\boldsymbol{M})]_{:,j}\right\|_2^2 = mn, \tag{12}$$

and again $\text{RMS}(\mathcal{N}_{\text{ColNorm}}(\boldsymbol{M})) = 1$.

SinkGD ends with a column normalization, so Equation 12 gives $\text{RMS}(\mathcal{N}_{\text{SinkGD}}(\boldsymbol{M})) = 1$ for any $K \geq 1$, while its row norms are only approximately $\sqrt{n}$.

The derivation for Muon largely follows Lemma 1 in Liu et al. (2025). Let $r = \text{rank}(\boldsymbol{M})$ and write the compact singular value decomposition $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, with $\boldsymbol{U} \in \mathbb{R}^{m\times r}$ and $\boldsymbol{V} \in \mathbb{R}^{n\times r}$ having orthonormal columns. As in Table 1, Muon maps $\boldsymbol{M}$ to

$$\mathcal{N}_{\text{Muon}}(\boldsymbol{M}) = \sqrt{\max(m,n)}\,\boldsymbol{U}\boldsymbol{V}^\top. \tag{13}$$

Its squared Frobenius norm is

$$\|\mathcal{N}_{\text{Muon}}(\boldsymbol{M})\|_F^2 = \max(m,n)\,\text{tr}\big(\boldsymbol{V}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{V}^\top\big) = \max(m,n)\,\text{tr}\big(\boldsymbol{V}^\top\boldsymbol{V}\big) = \max(m,n)\,r, \tag{14}$$

using $\boldsymbol{U}^\top\boldsymbol{U} = \boldsymbol{I}_r$, the cyclic property of the trace, and $\boldsymbol{V}^\top\boldsymbol{V} = \boldsymbol{I}_r$. When $\boldsymbol{M}$ has full rank, $r = \min(m,n)$, so $\|\mathcal{N}_{\text{Muon}}(\boldsymbol{M})\|_F^2 = mn$ and $\text{RMS}(\mathcal{N}_{\text{Muon}}(\boldsymbol{M})) = 1$.

B EXPERIMENT SETTINGS

This appendix describes the experiment settings of each task. For each task, we give the training recipe and the parameter partition between the matrix method and AdamW.

B.1 LANGUAGE MODELING

Training recipe. Table 6 summarizes the architectures and training budgets. We use a context length of 2048 at 950M, following the original pre-training recipe, and shorten it to 1024 at 140M to reduce the training budget. We use the phase-1 pre-training data of MobileLLM-R1 (Zhao et al., 2026). Other settings follow the MobileLLM-R1 recipe and are shared by all methods.

	layers	heads	KV heads	dim	hidden dim	context	params	tokens
140M	15	9	3	576	2048	1024	140.2M	105B
950M	22	24	6	1536	6144	2048	949.7M	2.1T

Table 6: **Language model architectures and training budgets.**

At 140M, we tune the learning rate for Muon and use the same value for all other methods. Table 7 shows the final validation loss. We find that 1e-2 is optimal for both AdamW and Muon.

	1e-1	3e-2	1e-2	3e-3	1e-3	3e-4
AdamW	3.2038±.0119	3.0865±.0072	**3.0724**±.0033	3.0823±.0038	3.1290±.0091	3.2236±.0025
Muon	3.1512±.0064	3.0438±.0076	**3.0434**±.0098	3.0561±.0066	3.0997±.0048	3.1587±.0060

Table 7: **Learning rate sweep for language modeling at 140M.** Each cell reports the mean and standard deviation of the final validation loss over five seeds.

Table 8 gives the training recipe. All parameters except the RMSNorm layers use a weight decay of 0.1. The warmup covers the first 0.4% of training at both scales, and its length grows in proportion to the total number of steps. We clip gradients at a norm of 1.0 and train in bfloat16 mixed precision (*i.e.*, keeping the weights and optimizer states in float32).

config	value
batch size	2048
training steps	50K (140M), 500K (950M)
warmup steps	200 (140M), 2000 (950M)
peak learning rate	1e-2 (140M), 1e-3 (950M)
learning rate schedule	linear decay
AdamW momentum	$\beta_1, \beta_2 = 0.9, 0.95$
weight decay	0.1 (0 for RMSNorm)
gradient clipping	1.0
precision	bfloat16 mixed precision

Table 8: **Language modeling training recipe.**

Parameter partition. Table 9 lists which parameters use the matrix method. For the query, key, and value matrices, the map is applied to each attention head separately. The output head shares its weight matrix with the token embedding, and this shared matrix is optimized with AdamW.

optimizer	parameter
matrix method	query, key, and value matrices
	attention output matrix
	MLP gate, up, and down matrices
AdamW	token embedding
	output head
	RMSNorm

Table 9: **Parameter partition for language modeling.**

B.2 VISION

We use the official codebases and recipes for all three vision tasks and change only the optimizer.

Supervised learning. We follow the supervised ViT training recipe of He et al. (2022), as shown in Table 10. All methods use a base learning rate of 1e-4. At this learning rate, Muon reaches the lowest training loss but overfits: its validation loss starts to rise after about epoch 250 for ViT-B and epoch 140 for ViT-L, while its validation accuracy saturates. Thus, for Muon and its combinations with normalization, we report the final training loss at a base learning rate of 1e-4 and the accuracy at 3e-4, where the validation loss keeps decreasing until late in training.

config	value
optimizer	AdamW or matrix method
base learning rate	1e-4
weight decay	0.3
AdamW momentum	$\beta_1, \beta_2 = 0.9, 0.95$
batch size	4096
learning rate schedule	cosine decay
warmup epochs	20
training epochs	300 (ViT-B), 200 (ViT-L)
augmentation	RandAug (9, 0.5) (Cubuk et al., 2020)
label smoothing (Szegedy et al., 2016)	0.1
mixup (Zhang et al., 2018)	0.8
cutmix (Yun et al., 2019)	1.0
random erasing (Zhong et al., 2020)	0.25
drop path (Huang et al., 2016)	0.1 (ViT-B), 0.2 (ViT-L)
precision	float16 mixed precision

Table 10: **Supervised training recipe.**

Table 11 lists the parameter partition. Besides the matrices in the Transformer blocks, the ViT has a patch embedding, a learnable position embedding, a class token, and a classification head, which are optimized with AdamW together with the LayerNorm parameters and all biases.

optimizer	parameter
matrix method	query, key, and value matrices
	attention output matrix
	MLP up and down matrices
AdamW	patch embedding
	position embedding
	class token
	classification head
	LayerNorm
	biases

Table 11: **Parameter partition for supervised learning.**

Self-supervised learning. Table 12 gives both the pre-training and the fine-tuning recipe from He et al. (2022). Pre-training uses a base learning rate of 1.5e-4 for all methods. Compared with the original recipe, we halve the pre-training batch size from 4096 to 2048. We find this can accelerate convergence and yield better final validation accuracy.

We fine-tune every pre-trained checkpoint with AdamW, so that runs differ only in the pre-training optimizer. The best fine-tuning learning rate may vary across checkpoints, so we sweep it around the value recommended by He et al. (2022) and report the best final accuracy of each checkpoint. The best value always lies at or next to the recommended one.

Table 13 shows the parameter partition for pre-training. In addition to the block matrices of the encoder and decoder, AdamW updates the patch embedding, the class and mask tokens, the decoder input projection, and the decoder prediction head, as well as all LayerNorm parameters and biases. Both the encoder and the decoder use fixed position embeddings.

config	value
optimizer	AdamW or matrix method
base learning rate	1.5e-4
weight decay	0.05
AdamW momentum	$\beta_1, \beta_2 = 0.9, 0.95$
batch size	2048
learning rate schedule	cosine decay
warmup epochs	40
training epochs	800
augmentation	RandomResizedCrop
precision	float16 mixed precision

(a) pre-training

config	value
optimizer	AdamW
base learning rate	1.5e-3, 5e-4, 1.5e-4 (B) 3e-3, 1e-3, 3e-4 (L)
weight decay	0.05
AdamW momentum	$\beta_1, \beta_2 = 0.9, 0.999$
layer-wise lr decay	0.65 (B), 0.75 (L)
batch size	1024
learning rate schedule	cosine decay
warmup epochs	5
training epochs	100 (B), 50 (L)
augmentation	RandAug (9, 0.5)
label smoothing	0.1
mixup / cutmix	0.8 / 1.0
random erasing	0.25
drop path	0.1 (B), 0.2 (L)
precision	float16 mixed precision

(b) fine-tuning

Table 12: **MAE recipe.**

optimizer	parameter
matrix method	query, key, and value matrices (encoder and decoder)
	attention output matrix (encoder and decoder)
	MLP up and down matrices (encoder and decoder)
AdamW	patch embedding
	class token
	mask token
	decoder input projection
	decoder prediction head
	LayerNorm
	biases

Table 13: **Parameter partition for MAE pre-training.**

Diffusion models. Table 14 lists the DiT recipe of Peebles & Xie (2023). Every method trains with the original constant learning rate of 1e-4. The model operates in the latent space of an off-the-shelf VAE (ft-EMA) from Stable Diffusion (Rombach et al., 2022). To evaluate a run, we draw 50K samples from the EMA weights of its final checkpoint. Sampling uses 250 DDPM (Ho et al., 2020) steps without classifier-free guidance (Ho & Salimans, 2021). We then compute FID against the ImageNet 256×256 reference batch using the evaluation suite of Dhariwal & Nichol (2021).

config	value
optimizer	AdamW or matrix method
learning rate	1e-4
weight decay	0
AdamW momentum	$\beta_1, \beta_2 = 0.9, 0.999$
batch size	256
learning rate schedule	constant
training steps	400K
augmentation	horizontal flip
class dropout	0.1
EMA decay	0.9999
precision	float32

Table 14: **DiT training recipe.**

Table 15 gives the parameter partition. Outside the Transformer blocks, DiT has a patch embedding, a timestep embedding, a class embedding, and a final output layer, all of which go to AdamW together with the biases. Each block also contains an adaLN modulation layer, which predicts the shift, scale,

and gate parameters from the timestep and class embedding rather than transforming hidden features. Therefore, we treat it as a conditioning layer and keep it on AdamW.

optimizer	parameter
matrix method	query, key, and value matrices
	attention output matrix
	MLP up and down matrices
AdamW	patch embedding
	timestep embedding
	class embedding
	adaLN modulation
	final output projection
	biases

Table 15: **Parameter partition for DiT.**

C Additional Results

C.1 EMA Statistics with Normalization Before Muon

Table 16 repeats the comparison of Table 4 with normalization applied before Muon. As shown before, EMA statistics bring no consistent benefit over simple normalization. Across all metrics, EMA improves six comparisons, degrades seven, and ties the rest.

	language		**supervised**†		**self-supervised**		**diffusion**	
	loss	change	loss	change	loss	change	loss	change
Muon	3.0434	–	2.3616	–	0.4037	–	0.1548	–
AdamW	3.0724	–	2.5041	–	0.4059	–	0.1558	–
simple normalization								
RowNorm	3.0517	↑.0083	2.3592	↓.0024	**0.4033**	↓.0004	**0.1546**	↓.0002
ColNorm	3.0483	↑.0049	**2.3538**	↓.0078	0.4036	↓.0001	0.1547	↓.0001
EMA statistics								
RowNorm	**3.0431**	↓.0003	2.3555	↓.0061	0.4034	↓.0003	**0.1546**	↓.0002
ColNorm	3.0453	↑.0019	2.3567	↓.0049	0.4036	↓.0001	0.1547	↓.0001

(a) loss

	language		**supervised**†		**self-supervised**		**diffusion**	
	accuracy	change	accuracy	change	accuracy	change	FID	change
Muon	45.43	–	82.49	–	83.27	–	60.08	–
AdamW	44.90	–	82.21	–	**83.41**	–	69.17	–
simple normalization								
RowNorm	**45.46**	↑0.03	82.50	↑0.01	83.31	↑0.04	59.21	↓0.87
ColNorm	45.39	↓0.04	82.58	↑0.09	83.23	↓0.04	**58.18**	↓1.90
EMA statistics								
RowNorm	45.10	↓0.33	82.40	↓0.09	83.27	0.00	59.15	↓0.93
ColNorm	45.28	↓0.15	**82.73**	↑0.24	83.24	↓0.03	58.85	↓1.23

(b) task metrics

Table 16: **Normalizing with simple or EMA statistics before Muon.** The simple normalization rows are copied from Table 3, and the same protocol applies.

C.2 Preconditioning View of Muon and Normalization

We keep the setting of Section 5: a momentum matrix $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ with $m \le n$ and full row rank, compact singular value decomposition $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, and Muon $\mathcal{N}_{\text{Muon}}(\boldsymbol{M}) = \sqrt{n}\,\boldsymbol{U}\boldsymbol{V}^\top$. We first rewrite this map directly in terms of $\boldsymbol{M}$. Substituting the singular value decomposition gives

$$\begin{aligned}(\boldsymbol{M}\boldsymbol{M}^\top)^{-1/2}\boldsymbol{M} &= \left[(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top)(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top)^\top\right]^{-1/2}(\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top)\\ &= \left(\boldsymbol{U}\boldsymbol{\Sigma}^2\boldsymbol{U}^\top\right)^{-1/2}\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top\\ &= \boldsymbol{U}\boldsymbol{\Sigma}^{-1}\boldsymbol{U}^\top\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top\\ &= \boldsymbol{U}\boldsymbol{V}^\top.\end{aligned} \tag{15}$$

Therefore, Muon's orthogonalization can equivalently be written as

$$\mathcal{N}_{\text{Muon}}(\boldsymbol{M}) = \sqrt{n}\,(\boldsymbol{M}\boldsymbol{M}^\top)^{-1/2}\boldsymbol{M}. \tag{16}$$

Equation 16 views Muon as applying the full matrix preconditioner $(\boldsymbol{M}\boldsymbol{M}^\top)^{-1/2}$ to the momentum.

We next consider a diagonal approximation to this preconditioner. Instead of using the full Gram matrix $\boldsymbol{M}\boldsymbol{M}^\top$, we retain only its diagonal:

$$\mathcal{N}_{\text{Diag}}(\boldsymbol{M}) = \sqrt{n}\ \text{diag}(\boldsymbol{M}\boldsymbol{M}^\top)^{-1/2}\boldsymbol{M}, \tag{17}$$

where $\text{diag}(\cdot)$ sets all off-diagonal entries to zero.

Equation 17 is exactly RowNorm. To see this, the i-th diagonal entry of the Gram matrix is

$$(\boldsymbol{M}\boldsymbol{M}^\top)_{ii} = \boldsymbol{M}_{i,:}\boldsymbol{M}_{i,:}^\top = \|\boldsymbol{M}_{i,:}\|_2^2. \tag{18}$$

Therefore, the i-th row of the resulting update is

$$[\mathcal{N}_{\mathrm{Diag}}(\boldsymbol{M})]_{i,:} = \sqrt{n}\,\frac{\boldsymbol{M}_{i,:}}{\|\boldsymbol{M}_{i,:}\|_2}. \tag{19}$$

Thus, $\mathcal{N}_{\mathrm{Diag}}$ rescales every row to have ℓ_2 norm $\sqrt{n}$, exactly matching RowNorm under our unit per-element RMS convention: $\mathcal{N}_{\mathrm{Diag}}(\boldsymbol{M}) = \mathcal{N}_{\mathrm{RowNorm}}(\boldsymbol{M})$. This connection was established by Deng et al. (2026), who derive RowNorm as a diagonal approximation of Muon's preconditioner $(\boldsymbol{M}\boldsymbol{M}^\top)^{-1/2}$, and we restate their argument in our notation.

When $m > n$, the same derivation applied to $\boldsymbol{M}^\top$ yields $\mathcal{N}_{\mathrm{ColNorm}}(\boldsymbol{M})$, so the shape-aligned diagonal approximation is RowNorm for wide matrices and ColNorm for tall ones.

This preconditioning view also gives a direct proof of Theorem 1. We restate it below.

Theorem 1. *Let $\boldsymbol{M} \in \mathbb{R}^{m\times n}$ with $m \le n$ and full row rank. Then*

$$\mathcal{N}_{\mathrm{RowNorm}}(\boldsymbol{M}) = \mathcal{N}_{\mathrm{Muon}}(\boldsymbol{M}) \tag{5}$$

if and only if

$$\boldsymbol{M}\boldsymbol{M}^\top = \mathrm{diag}(\boldsymbol{M}\boldsymbol{M}^\top), \tag{6}$$

i.e., the rows of $\boldsymbol{M}$ are mutually orthogonal.

Proof. Let

$$\boldsymbol{G} = \boldsymbol{M}\boldsymbol{M}^\top, \qquad \boldsymbol{D} = \mathrm{diag}(\boldsymbol{G}).$$

Equations 16 and 17 give

$$\mathcal{N}_{\mathrm{Muon}}(\boldsymbol{M}) = \sqrt{n}\,\boldsymbol{G}^{-1/2}\boldsymbol{M}, \qquad \mathcal{N}_{\mathrm{RowNorm}}(\boldsymbol{M}) = \sqrt{n}\,\boldsymbol{D}^{-1/2}\boldsymbol{M}.$$

(⇒) Suppose that RowNorm and Muon produce the same update:

$$\mathcal{N}_{\mathrm{RowNorm}}(\boldsymbol{M}) = \mathcal{N}_{\mathrm{Muon}}(\boldsymbol{M}).$$

Substituting their preconditioning forms and cancelling the common factor $\sqrt{n}$ gives

$$\boldsymbol{D}^{-1/2}\boldsymbol{M} = \boldsymbol{G}^{-1/2}\boldsymbol{M}. \tag{20}$$

Because $\boldsymbol{M}$ has full row rank, $\boldsymbol{G} = \boldsymbol{M}\boldsymbol{M}^\top$ is positive definite and invertible. We can therefore construct a right inverse of $\boldsymbol{M}$ as

$$\boldsymbol{R} = \boldsymbol{M}^\top(\boldsymbol{M}\boldsymbol{M}^\top)^{-1}, \tag{21}$$

which satisfies

$$\boldsymbol{M}\boldsymbol{R} = \boldsymbol{M}\boldsymbol{M}^\top(\boldsymbol{M}\boldsymbol{M}^\top)^{-1} = \boldsymbol{I}.$$

Multiplying Equation 20 by $\boldsymbol{R}$ on the right removes $\boldsymbol{M}$ from both sides:

$$\boldsymbol{D}^{-1/2} = \boldsymbol{G}^{-1/2}.$$

Both $\boldsymbol{D}$ and $\boldsymbol{G}$ are positive definite, so this equality implies

$$\boldsymbol{D} = \boldsymbol{G}.$$

By definition, $\boldsymbol{D} = \mathrm{diag}(\boldsymbol{G})$. Hence $\boldsymbol{G}$ has no off-diagonal entries:

$$\boldsymbol{M}\boldsymbol{M}^\top = \mathrm{diag}(\boldsymbol{M}\boldsymbol{M}^\top).$$

The off-diagonal entries of $\boldsymbol{M}\boldsymbol{M}^\top$ are the pairwise inner products between rows of $\boldsymbol{M}$, so this is equivalent to the rows of $\boldsymbol{M}$ being mutually orthogonal.

(⇐) Suppose the rows of $\boldsymbol{M}$ are mutually orthogonal. Then $\boldsymbol{G} = \boldsymbol{M}\boldsymbol{M}^\top$ is diagonal, so

$$\boldsymbol{G} = \boldsymbol{D}.$$

The two preconditioners are therefore identical:

$$\boldsymbol{G}^{-1/2} = \boldsymbol{D}^{-1/2}.$$

Substituting this equality into the two update maps immediately gives

$$\mathcal{N}_{\mathrm{RowNorm}}(\boldsymbol{M}) = \mathcal{N}_{\mathrm{Muon}}(\boldsymbol{M}).$$

□

Shape-aligned normalization. The derivation suggests RowNorm for wide matrices and ColNorm for tall ones, so each normalized vector runs along the longer dimension. We call this LongNorm; its control, ShortNorm, normalizes along the shorter dimension instead. Table 17 compares LongNorm and ShortNorm against the methods in Table 2.

	language		**supervised**†		**self-supervised**		**diffusion**	
	loss	change	loss	change	loss	change	loss	change
Muon	**3.0434**	–	**2.3616**	–	**0.4037**	–	**0.1548**	–
AdamW	3.0724	–	2.5041	–	0.4059	–	0.1558	–
RowNorm	3.0538	↑.0104	2.5176	↑.1560	0.4061	↑.0024	0.1556	↑.0008
ColNorm	3.0512	↑.0078	2.4999	↑.1383	0.4064	↑.0027	0.1557	↑.0009
SinkGD	3.0552	↑.0118	2.5224	↑.1608	0.4063	↑.0026	0.1554	↑.0006
LongNorm	3.4561	↑.4127	2.5068	↑.1452	0.4062	↑.0025	0.1557	↑.0009
ShortNorm	3.0656	↑.0222	2.5455	↑.1839	0.4068	↑.0031	0.1556	↑.0008

(a) loss

	language		**supervised**†		**self-supervised**		**diffusion**	
	accuracy	change	accuracy	change	accuracy	change	FID	change
Muon	**45.43**	–	**82.49**	–	83.27	–	**60.08**	–
AdamW	44.90	–	82.21	–	83.41	–	69.17	–
RowNorm	44.91	↓0.52	82.14	↓0.35	**83.44**	↑0.17	67.45	↑7.37
ColNorm	45.10	↓0.33	82.07	↓0.42	83.23	↓0.04	67.74	↑7.66
SinkGD	45.08	↓0.35	82.11	↓0.38	83.37	↑0.10	65.86	↑5.78
LongNorm	42.00	↓3.43	82.08	↓0.41	83.31	↑0.04	68.34	↑8.26
ShortNorm	44.86	↓0.57	82.17	↓0.32	83.23	↓0.04	67.56	↑7.48

(b) task metrics

Table 17: **Shape-aligned normalization.** LongNorm applies RowNorm to wide matrices and ColNorm to tall ones, and ShortNorm does the opposite. The other rows are copied from Table 2, and the same protocol applies. Neither choice makes the performance close to Muon, and LongNorm is much worse in language modeling.

LongNorm or ShortNorm does not improve over RowNorm or ColNorm in any task. In language modeling, it is far worse than every other method, while ShortNorm stays close to the other normalization methods. In vision, the trend is also similar. Hence, even with the side chosen by the matrix shape, normalization still falls behind Muon. This reinforces that normalization is different from Muon's orthogonalization, and it should not be viewed as an approximation.

C.3 Singular Value Distributions of the Update

We here seek to understand how different the normalized update is from Muon in the singular value distributions. We collect momentum matrices saved from a Muon training trajectory and apply both LongNorm (Appendix C.2) and Muon to them. We then compare the singular value distributions of these two updates. Since both maps follow the same unit per-element RMS convention, we normalize each singular value by $\sqrt{\max(m, n)}$. For Muon, this will make all normalized singular values equal to one, whereas normalization can produce values both below and above one.

Figure 4 plots the normalized singular value distributions. We can see that normalization and Muon produce very different singular value distributions. Muon's singular values stay close to one across tasks and training progress. In contrast, LongNorm concentrates the update on a few large singular values, while most of the others lie well below one.

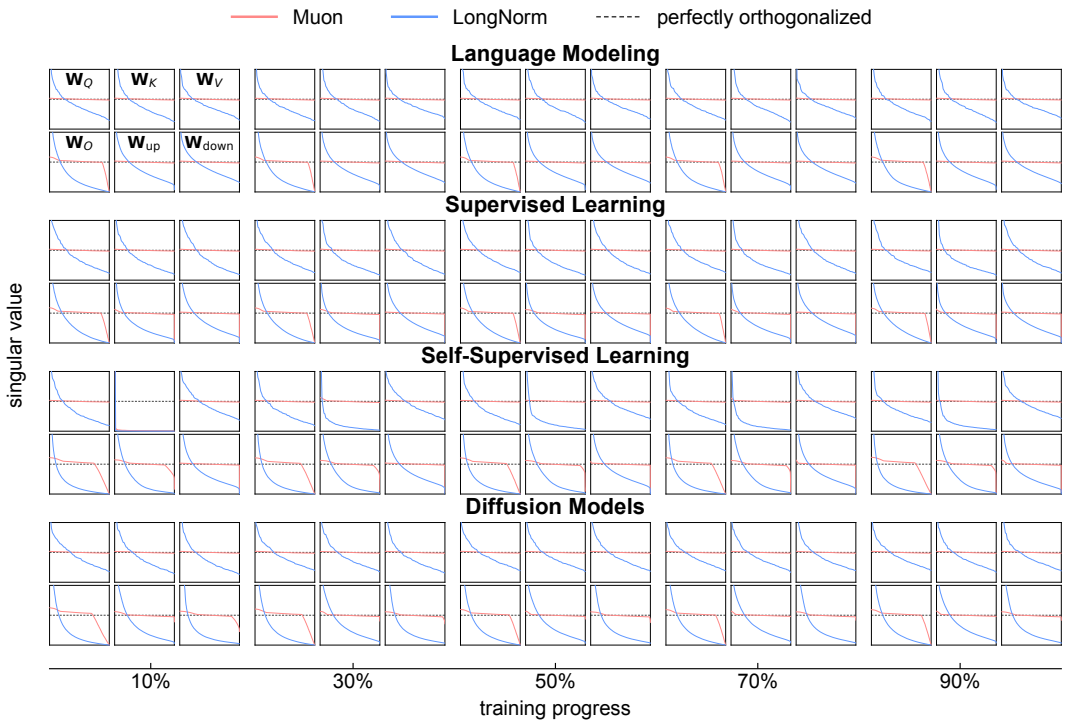

Figure 4: **Singular value distributions of normalized updates.** We plot the sorted singular values of updates from LongNorm (Appendix C.2) and Muon, computed from the same momentum and rescaled to unit per-element RMS. A perfectly orthogonalized update has all singular values equal to one (dashed line). LongNorm shows steeply decaying singular value distributions in every task and throughout training. This contrasts with Muon, whose singular values stay close to one. The sampled layers and heads follow Figure 2.