MUON IS SCALABLE FOR LLM TRAINING

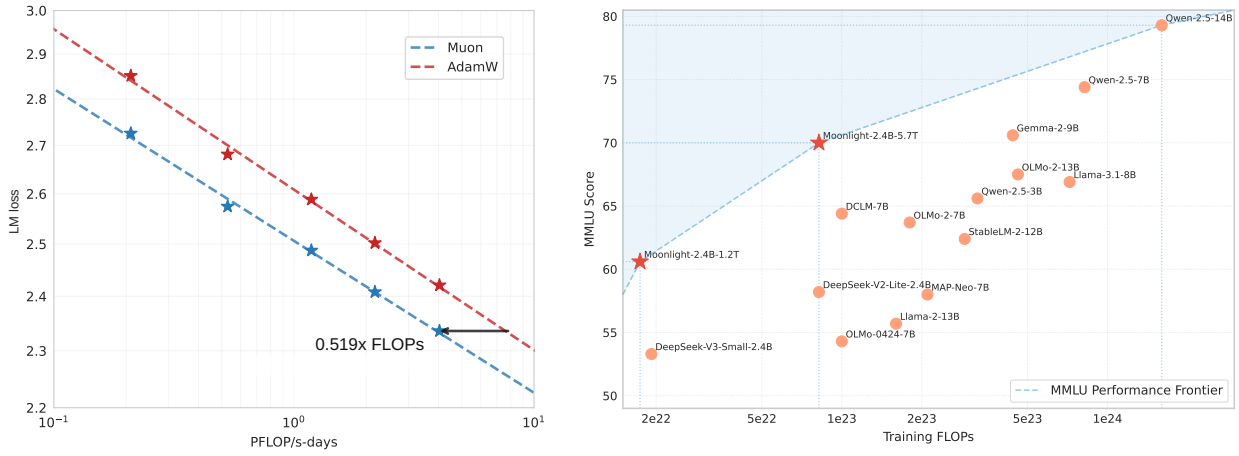
TECHNICAL REPORT

Jingyuan Liu[1] **Jianlin Su**[1] **Xingcheng Yao**[2] **Zhejun Jiang**[1] **Guokun Lai**[1] **Yulun Du**[1] **Yidao Qin**[1] **Weixin Xu**[1] **Enzhe Lu**[1] **Junjie Yan**[1] **Yanru Chen**[1] **Huabin Zheng**[1] **Yibo Liu**[1] **Shaowei Liu**[1] **Bohong Yin**[1] **Weiran He**[1] **Han Zhu**[1] **Yuzhi Wang**[1] **Jianzhou Wang**[1] **Mengnan Dong**[1] **Zheng Zhang**[1] **Yongsheng Kang**[1] **Hao Zhang**[1] **Xinran Xu**[1] **Yutao Zhang**[1] **Yuxin Wu**[1] **Xinyu Zhou**[1] * **Zhilin Yang**[1]

[1] Moonshot AI [2] UCLA

ABSTRACT

Recently, the Muon optimizer (K. Jordan et al. 2024) based on matrix orthogonalization has demonstrated strong results in training small-scale language models, but the scalability to larger models has not been proven. We identify two crucial techniques for scaling up Muon: (1) adding weight decay and (2) carefully adjusting the per-parameter update scale. These techniques allow Muon to work out-of-the-box on large-scale training without the need of hyper-parameter tuning. Scaling law experiments indicate that Muon achieves $\sim 2\times$ computational efficiency compared to AdamW with compute optimal training. Based on these improvements, we introduce Moonlight, a 3B/16B-parameter Mixture-of-Expert (MoE) model trained with 5.7T tokens using Muon. Our model improves the current Pareto frontier, achieving better performance with much fewer training FLOPs compared to prior models. We open-source our distributed Muon implementation that is memory optimal and communication efficient. We also release the pretrained, instruction-tuned, and intermediate checkpoints to support future research.



(a) (b)

Figure 1: Scaling up with Muon. **(a)** Scaling law experiments comparing Muon and Adam. Muon is $\sim 2\times$ more computational efficient than Adam with compute optimal training. **(b)** The MMLU performance of our Moonlight model optimized with Muon and other comparable models. Moonlight advances the Pareto frontier of performance vs training FLOPs.

*Corresponding author: zhouxinyu@moonshot.cn

1 Introduction

The rapid advancement of large language models (LLMs) (OpenAI et al. 2024; DeepSeek-AI et al. 2024; Grattafiori et al. 2024; Gemini Team et al. 2024; Pethick et al. 2025) has significantly pushed forward the progress in artificial general intelligence. However, training capable LLMs remains a computationally intensive and resource-demanding process due to scaling laws (Kaplan et al. 2020; Hoffmann et al. 2022). Optimizers play a crucial role in efficiently and effectively training of LLMs, with Adam (Kingma et al. 2015) and its variant AdamW (Loshchilov et al. 2019) being the standard choice for most large-scale training.

Recent developments in optimization algorithms have shown potential to improve training efficiency beyond AdamW (Liu et al. 2024; K. Jordan et al. 2024; Yuan et al. 2024; Vyas et al. 2025; X.-L. Li 2018a; X.-L. Li 2018b; Pooladzandi et al. 2024; X. Li 2022; X.-L. Li 2024). Among these, K. Jordan et al. 2024 proposed Muon, which updates matrix parameters with orthogonalized gradient momentum using Newton-Schulz iteration. Initial experiments with Muon have demonstrated promising results in small-scale language model training. However, as discussed in this blog (K. Jordan et al. 2024), several critical challenges remain unaddressed: (1) how to effectively scale optimizers based on matrix orthogonalization to larger models with billions of parameters trained with trillions of tokens, (2) how to compute approximate orthogonalization in a distributed setting, and (3) whether such optimizers can generalize across different training stages including pre-training and supervised finetuning (SFT).

In this technical report, we present a comprehensive study addressing these challenges. Our work builds upon Muon while systematically identifying and resolving its limitations in large-scale training scenarios. Our technical contributions include:

- **Analysis for Effective Scaling of Muon**: Through extensive analysis, we identify that weight decay plays a crucial role in Muon's scalability. Besides, we propose scale adjustments to Muon's parameter-wise update rule. Such adjustments allow Muon to work out-of-the-box without hyper-parameter tuning, and also significantly improve training stability.
- **Efficient Distributed Implementation**: We develop a distributed version of Muon with ZeRO-1 (Rajbhandari et al. 2020) style optimization, achieving optimal memory efficiency and reduced communication overhead while preserving the mathematical properties of the algorithm.
- **Scaling Law Validation**: We performed scaling law research that compares Muon with strong AdamW baselines, and showed the superior performance of Muon (1a). Based on the scaling law results, Muon achieves comparable performance to AdamW trained counterparts while requiring only approximately 52% of the training FLOPs.

Our comprehensive experiments demonstrate that Muon can effectively replace AdamW as the de facto optimizer for large-scale LLM training, offering significant improvements in both training efficiency and model performance. As a result of this work, we release Moonlight, a 16B-parameter MoE model trained using Muon, along with our implementation and intermediate training checkpoints to facilitate further research in scalable optimization techniques for LLMs.

2 Methods

2.1 Background

The Muon Optimizer Muon (K. Jordan et al. 2024) has recently been proposed to optimize neural network weights representable as matrices. At iteration t, given current weight $\mathbf{W}_{t-1}$, momentum μ, learning rate η_t and objective $\mathcal{L}_t$, the update rule of the Muon optimizer can be stated as follows:

$$\begin{aligned} \mathbf{M}_t &= \mu \mathbf{M}_{t-1} + \nabla \mathcal{L}_t(\mathbf{W}_{t-1}) \\ \mathbf{O}_t &= \text{Newton-Schulz}(\mathbf{M}_t)^1 \\ \mathbf{W}_t &= \mathbf{W}_{t-1} - \eta_t \mathbf{O}_t \end{aligned} \tag{1}$$

Here, $\mathbf{M}_t$ is the momentum of gradient at iteration t, set as a zero matrix when $t = 0$. In Equation 1, a Newton-Schulz iteration process (Bernstein et al. 2024) is adopted to approximately solve $(\mathbf{M}_t\mathbf{M}_t^{\mathrm{T}})^{-1/2}\mathbf{M}_t$. Let $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^{\mathrm{T}} = \mathbf{M}_t$ be the singular value decomposition (SVD) of $\mathbf{M}_t$, we will have $(\mathbf{M}_t\mathbf{M}_t^{\mathrm{T}})^{-1/2}\mathbf{M}_t = \mathbf{U}\mathbf{V}^{\mathbf{T}}$, which orthogonalizes $\mathbf{M}_t$. Intuitively, orthogonalization can ensure that the update matrices are isomorphic, preventing the weight from learning along a few dominant directions (K. Jordan et al. 2024).

[1] In practice, we follow (K. Jordan et al. 2024) to use a Nesterov-style momentum by putting $\mu\mathbf{M}_t + \nabla\mathcal{L}_t(\mathbf{W}_{t-1})$ to the Newton-Schulz iteration instead of $\mathbf{M}_t$.

Newton-Schulz Iterations for Matrix Orthogonalization Equation 1 is calculated in an iterative process. At the beginning, we set $\mathbf{X}_0 = \mathbf{M}_t / \|\mathbf{M}_t\|_{\mathrm{F}}$. Then, at each iteration k, we update $\mathbf{X}_k$ from $\mathbf{X}_{k-1}$ as follows:

$$\mathbf{X}_k = a\mathbf{X}_{k-1} + b(\mathbf{X}_{k-1}\mathbf{X}_{k-1}^{\mathrm{T}})\mathbf{X}_{k-1} + c(\mathbf{X}_{k-1}\mathbf{X}_{k-1}^{\mathrm{T}})^2\mathbf{X}_{k-1} \tag{2}$$

where $\mathbf{X}_N$ is the result of such process after N iteration steps. Here a, b, c are coefficients. In order to ensure the correct convergence of Equation 2, we need to tune the coefficients so that the polynomial $f(x) = ax + bx^3 + cx^5$ has a fixed point near 1. In the original design of K. Jordan et al. 2024, the coefficients are set to $a = 3.4445$, $b = -4.7750$, $c = 2.0315$ in order to make the iterative process converge faster for small initial singular values. In this work, we follow the same setting of coefficients.

Steepest Descent Under Norm Constraints Bernstein et al. 2024 proposed to view the optimization process in deep learning as steepest descent under norm constraints. From this perspective, we can view the difference between Muon and Adam (Kingma et al. 2015; Loshchilov et al. 2019) as the difference in norm constraints. Whereas Adam is a steepest descent under the a norm constraint dynamically adjusted from a Max-of-Max norm, Muon offers a norm constraint that lies in a static range of Schatten-p norm for some large p (Franz 2024). When equation 1 is accurately computed, the norm constraint offered by Muon will be the spectral norm. Weights of neural networks are used as operators on the input space or the hidden space, which are usually (locally) Euclidean (Cesista 2024), so the norm constraint on weights should be an induced operator norm (or spectral norm for weight matrices). In this sense, the norm constraint offered by Muon is more reasonable than that offered by Adam.

2.2 Scaling Up Muon

Weight Decay While Muon performs significantly better than AdamW on a small scale as shown by K. Jordan et al. 2024, we found the performance gains diminish when we scale up to train a larger model with more tokens. We observed that both the weight and the layer output's RMS keep growing to a large scale, exceeding the high-precision range of bf16, which might hurt the model's performance. To resolve this issue, we introduced the standard AdamW (Loshchilov et al. 2019) weight decay mechanism into Muon[2].

$$\mathbf{W}_t = \mathbf{W}_{t-1} - \eta_t(\mathbf{O}_t + \lambda\mathbf{W}_{t-1}) \tag{3}$$

We experimented on Muon both with and without weight decay to understand its impact on the training dynamics of LLMs. Based on our scaling law research in Sec 3.2, we trained an 800M parameters model with 100B tokens ($\sim 5\times$ optimal training tokens). Figure 2 shows validation loss curves of the model trained with AdamW, vanilla Muon (without weight decay), and Muon with weight decay. While vanilla Muon initially converges faster, we observed that some model weights grew too large over time, potentially limiting the model's long-term performances. Adding weight decay addressed this issue - the results demonstrate that Muon with weight decay outperforms both vanilla Muon and AdamW, achieving lower validation loss in the over-train regime. Therefore, we adjusted our update rule to equation 3, where λ is the weight decay ratio.

Consistent update RMS An important property of Adam and AdamW (Kingma et al. 2015, Loshchilov et al. 2019) is that they maintain a theoretical update RMS around 1[3]. However, we show that Muon's update RMS varies depending on the shape of the parameters, according to the following lemma:

Lemma 1. *For a full-rank matrix parameter of shape* $[A, B]$*, its theoretical Muon update RMS is* $\sqrt{1/\max(A, B)}$.

The proof can be found in the Appendix A. We monitored Muon's update RMS during training and found it typically close to the theoretical value given above. We note that such inconsistency can be problematic when scaling up the model size:

- When $\max(A, B)$ is too large, e.g. the dense MLP matrix, the updates become too small, thus limiting the model's representational capacity and leading to suboptimal performances;
- When $\max(A, B)$ is too small, e.g. treating each KV head in GQA (Shazeer 2019) or MLA (DeepSeek-AI et al. 2024) as a separate parameter, the updates become too large, thus causing training instabilities and leading to suboptimal performances as well.

[2]The original implementation of Muon omits weight decay. A recent concurrent work in Muon incorporates weight decay and demonstrates improved performance. See this commit and this discussion.

[3]Due to Adam's $\beta_1 < \beta_2$ and $\epsilon > 0$, the actual update RMS is usually less than 1.

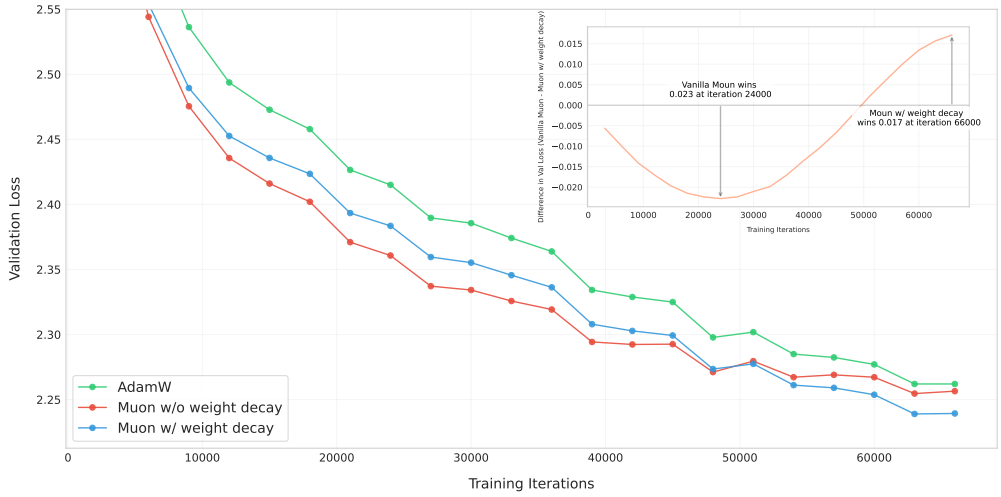


Figure 2: Validation loss curves for AdamW (green), Muon without weight decay (red), and Muon with weight decay (blue).

In order to maintain consistent update RMS among matrices of different shapes, we propose to scale the Muon update for each matrix by its $\sqrt{\max(A, B)}$ to cancel the effect of Lemma 1 [4] . Experiments in Sec 3.1 show that this strategy is beneficial for optimization.

Matching update RMS of AdamW Muon is designed to update matrix-based parameters. In practice, AdamW is used in couple with Muon to handle non-matrix based parameters, like RMSNorm, LM head, and embedding parameters. We would like the optimizer hyper-parameters (learning rate η, weight decay λ) to be shared among matrix and non-matrix parameters.

We propose to match Muon's update RMS to be similar to that of AdamW. From empirical observations, AdamW's update RMS is usually around 0.2 to 0.4. Therefore, we scale Muon's update RMS to this range by the following adjustment:

$$\mathbf{W}_t = \mathbf{W}_{t-1} - \eta_t(0.2 \cdot \mathbf{O}_t \cdot \sqrt{\max(A, B)} + \lambda \mathbf{W}_{t-1}) \tag{4}$$

We validated this choice with empirical results (see Appendix A for details). Moreover, we highlighted that with this adjustment, Muon can directly **reuse** the learning rate and weight decay tuned for AdamW.

Other Hyper-parameters Muon contains two other tunnable hyper-parameters: Newton-Schulz iteration steps and momentum μ. We empirically observe that when setting N to 10, the iterative process will yield a more accurate orthogonalization result than $N = 5$, but it won't lead to better performances. Hence we set $N = 5$ in this work for the sake of efficiency. We do not see a consistent performance gain in tuning momentum, so we chose 0.95, same as K. Jordan et al. 2024.

2.3 Distributed Muon

ZeRO-1 and Megatron-LM Rajbhandari et al. 2020 introduced the ZeRO-1 technique that partitions the expensive optimizer states (e.g. master weights, momentum) all over the cluster. Megatron-LM (Shoeybi et al. 2020) integrated ZeRO-1 into its native parallel designs. Based on Megatron-LM's sophisticated parallel strategies, e.g. Tensor-Parallel (TP), Pipeline Parallel (PP), Expert Parallel (EP) and Data Parallel (DP), the communication workload of ZeRO-1 can be reduced from gathering all over the distributed world to only gathering over the data parallel group.

Method ZeRO-1 is efficient for AdamW because it calculates updates in an element-wise fashion. However, Muon requires the full gradient matrix to calculate the updates. Therefore, vanilla ZeRO-1 is not directly applicable to Muon. We propose a new distributed solution based on ZeRO-1 for Muon, referred to as Distributed Muon. Distributed Muon

[4] K. Jordan et al. 2024's original implementation scales the updates by $\sqrt{\max(1, A/B)}$, which is equivalent to our proposal (up to a global scale) if all matrices have the same second dimension; You 2025 discusses a similar issue.

Algorithm 1 Distributed Muon

Require: Full Gradients $\mathbf{G}$, DP partitioned Momentum $\mathbf{m}$, DP partitioned parameters $\mathbf{p}$, momentum μ.

// Reduce-scatter G on DP for correct gradients
$\mathbf{g} =$ reduce_scatter($\mathbf{G}$, dp_group)
// Apply momentum to $\mathbf{g}$ using local partitioned momentum $\mathbf{m}$
$\mathbf{g}' =$ update_with_momentum($\mathbf{g}, \mathbf{m}, \mu$)
// DP Gather: gathering $\mathbf{g}'$ across DP into a full matrix $\mathbf{G}$
$\mathbf{G} =$ gather($\mathbf{g}'$, dp_group)
// Calculate Muon update
$\mathbf{U} =$ Newton-Schulz($\mathbf{G}$)
// Discard the rest of $\mathbf{U}$ and only keep the local partition $\mathbf{u}$, then apply the update rule
$\mathbf{p}' =$ apply_update($\mathbf{p}, \mathbf{u}$)
// All-gather updated $\mathbf{p}'$ into $\mathbf{P}$
$\mathbf{P} =$ all_gather($\mathbf{p}'$, dp_group)
// Return the update RMS for logging
return $\sqrt{\mathbf{u}^2.\texttt{mean}()}$

follows ZeRO-1 to partition the optimizer states on DP, and introduces two additional operations compared to a vanilla Zero-1 AdamW optimizer:

1. `DP Gather.` For a local DP partitioned master weight ($1/DP$ the size of the model weight), this operation is to gather the corresponding partitioned gradients into a full gradient matrix.
2. `Calculate Full Update.` After the above gathering, perform Newton-Schulz iteration steps on the full gradient matrix as described in Sec 2.1. Note that we will then discard part of the full update matrix, as we only need the partition corresponding to the local parameters to perform update.

The implementation of Distributed Muon is described in Algorithm 1. The additional operations introduced by Distributed Muon are colored in blue.

Analysis We compared Distributed Muon to a classic ZeRO-1 based distributed AdamW (referred as Distributed AdamW for simplicity) in several aspects:

- `Memory Usage.` Muon uses only one momentum buffer, while AdamW uses two momentum buffers. Therefore, the additional memory used by the Muon optimizer is half of Distributed AdamW.
- `Communication Overhead.` For each device, the additional DP gathering is only required by the local DP partitioned parameters $\mathbf{p}$. Therefore, the communication cost is less than the reduce-scatter of $\mathbf{G}$ or the all-gather of $\mathbf{P}$. Besides, Muon only requires the Newton-Schulz iteration steps in bf16, thus further reducing the communication overhead to 50% comparing to fp32. Overall, the communication workload of Distributed Muon is $(1, 1.25]$ of that of Distributed AdamW. The upper-bound is calculated as that the communication of Distributed Muon is 4 (fp32 $\mathbf{G}$ reduce-scatter) + 2 (bf16 Muon gather) + 4 (fp32 $\mathbf{P}$ all-gather), while Distributed AdamW is 4 + 4. In practice, as we usually train with multiple DP, the empirical additional cost usually is closer to the lower-bound 1.[5].
- `Latency.` Distributed Muon has larger end-to-end latencies than Distributed AdamW because it introduces additional communication and requires running Newton-Schulz iteration steps. However, this is not a significant issue because (a) only about 5 Newton-Schultz iteration steps are needed for a good result (discussed in Sec 2.2), and (b) the end-to-end latency caused by the optimizer is negligible compared to the model's forward-backward pass time (e.g. usually 1% to 3%). Moreover, several engineering techniques, such as overlapping gather and computation, and overlapping optimizer reduce-scatter with parameter gather, can further reduce latency.

When training large-scale models in our distributed cluster, Distributed Muon has no noticeable latency overhead compared to its AdamW counterparts. We will soon release a pull request that implements Distributed Muon for the open-source Megatron-LM (Shoeybi et al. 2020) project.

[5]If TP is enabled, Distributed Muon needs an extra bf16 TP gather on TP group.

Table 1: Controlling Muon's Update RMS Across Different Model Params

Methods	Training loss	Validation loss	query weight RMS	MLP weight RMS
Baseline	2.734	2.812	3.586e-2	2.52e-2
Update Norm	**2.72**	**2.789**	4.918e-2	5.01e-2
Adjusted LR	2.721	**2.789**	3.496e-2	4.89e-2

3 Experiments

3.1 Consistent Update RMS

As discussed in Sec 2.2, we aim to match the update RMS across all matrix parameters and also match it with that of AdamW. We experimented with two methods to control the Muon update RMS among parameters and compared them to a baseline that only maintains a consistent RMS with AdamW:

1. `Baseline.` We multiplied the update matrix by $0.2 \cdot \sqrt{H}$ (H is the model hidden size) to maintain a consistent update RMS with AdamW. Note that $\max(A, B)$ equals to H for most matrices.
$$\mathbf{W}_t = \mathbf{W}_{t-1} - \eta_t(0.2 \cdot \mathbf{O}_t \cdot \sqrt{H} + \lambda \mathbf{W}_{t-1}) \tag{5}$$
2. `Update Norm.` We can directly normalize the updates calculated via Newton-Schulz iterations so its RMS strictly becomes 0.2;
$$\mathbf{W}_t = \mathbf{W}_{t-1} - \eta_t(0.2 \cdot \mathbf{O}_t / \operatorname{RMS}(\mathbf{O}_t) + \lambda \mathbf{W}_{t-1}) \tag{6}$$
3. `Adjusted LR.` For each update matrix, we can scale its learning rate by a factor of $0.2 \cdot \sqrt{\max(A, B)}$ based on its shape.
$$\mathbf{W}_t = \mathbf{W}_{t-1} - \eta_t(0.2 \cdot \mathbf{O}_t \cdot \sqrt{\max(A, B)} + \lambda \mathbf{W}_{t-1}) \tag{7}$$

Analysis We designed experiments to illustrate the impact of Muon update RMS at an early training stage, because we observed that unexpected behaviors happened very quickly when training models at larger scale. We experimented with small scale 800M models as described in 3.2. The problem of inconsistent update RMS is more pronounced when the disparity between matrix dimensions increases. To highlight the problem for further study, we slightly modify the model architecture by replacing the Swiglu MLP with a standard 2-layer MLP, changing the shape of its matrix parameters from $[H, 2.6H]$ to $[H, 4H]$. We evaluated the model's loss and monitored a few of its parameters' RMS, specifically, attention query (shape $[H, H]$) and MLP (shape $[H, 4H]$). We evaluated the model after training for 4B tokens out of a 20B-token schedule. From Table 1, we observed several interesting findings:

1. Both `Update Norm` and `Adjusted LR` achieved better performances than `Baseline`;
2. For the MLP weight matrix of shape $[H, 4H]$, both `Update Norm` and `Adjusted LR` obtain a weight RMS that is roughly doubled comparing to `Baseline`. This is reasonable as $\sqrt{\max(H, 4H)}/\sqrt{H} = 2$, so the update RMS of `Update Norm` and `Adjusted LR` is roughly two times of `Baseline`;
3. For the attention query weight matrix of shape $[H, H]$, `Update Norm` still norms the update, while `Adjusted LR` does not because $\sqrt{\max(H, H)}/\sqrt{H} = 1$. As a result, `Adjusted LR` results in a similar weight RMS as `Baseline`, but `Update Norm` has a larger weight rms similar to its MLP.

Based on these findings, we choose the `Adjusted LR` method for future experiments because it has lower cost.

3.2 Scaling Law of Muon

For a fair comparison with AdamW, we performed scaling law experiments on a series of dense models in Llama (Grattafiori et al. 2024) architecture. Building a strong baseline is of crucial importance in optimizer research. Hence, we perform a grid search for hyper-parameters of AdamW, following the compute-optimal training setup (Kaplan et al. 2020) (the grid search experiments can be found in Appendix B). Details of the model architecture and hyper-parameters can be found in Table 2. For Muon, as discussed in Sec 2.2, since we matched Muon's update RMS to AdamW, we directly reused the hyper-parameters that are optimal for the AdamW baseline.

The fitted scaling law curve can be found in figure 3, and the fitted equations are detailed in table 3. As shown in Figure 1a, Muon only requires about 52% training FLOPs to match the performance of AdamW under compute-optimal setting.

Table 2: Scaling Law Models and Hyper-Parameters

# Params. w/o Embedding	Head	Layer	Hidden	Tokens	LR	Batch Size*
399M	12	12	1536	8.92B	9.503e-4	96
545M	14	14	1792	14.04B	9.143e-4	128
822M	16	16	2048	20.76B	8.825e-4	160
1.1B	18	18	2304	28.54B	8.561e-4	192
1.5B	20	20	2560	38.91B	8.305e-4	256

*In terms of number of examples in 8K context length.

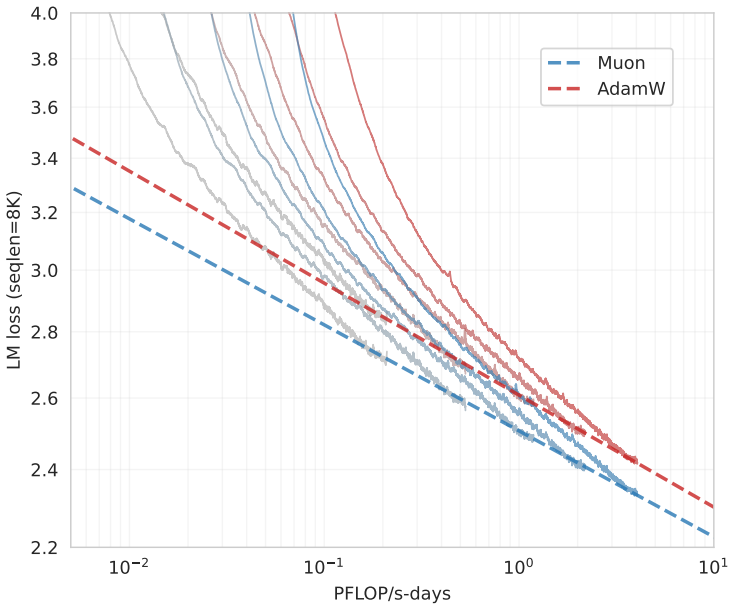


Figure 3: Fitted scaling law curves for Muon and AdamW optimizers.

3.3 Pretraining with Muon

Model Architecture To evaluate Muon against contemporary model architectures, we pretrained from scratch using the deepseek-v3-small architecture (DeepSeek-AI et al. 2024) as it demonstrates strong performance and the original results serve as a reference for comparison. Our pretrained model has 2.24B activated and 15.29B total parameters (3B activated and 16B total when including embedding). Minor modifications to the architecture are detailed in Appendix C.

Pretraining Data Our pretraining data details can be found in K. Team 2025. The maximum context length during pretraining is 8K.

Pretraining The model is trained in several stages. We use a 1e-3 auxfree bias update rate in stage 1 and 2, and 0.0 auxfree bias update rate in stage 3. The weight decay is set to 0.1 for all stages. More details and discussions of model training can be found in the Appendix D.

1. `0 to 33B tokens:` In this stage, the learning rate linearly increases to 4.2e-4 in 2k steps. The batch size is kept at 2048 examples;

Table 3: Fitted parameters of the scaling law curves

	Muon	AdamW
LM loss (seqlen=8K)	$2.506 \times C^{-0.052}$	$2.608 \times C^{-0.054}$

2. `33B to 5.2T tokens:` In this stage, the learning rate decays from 4.2e-4 to 4.2e-5 in a cosine style. We keep the batch size at 2048 until 200B tokens, and then doubled to 4096 for the remaining;
3. `5.2T to 5.7T tokens:` In this stage (also referred as the cooldown stage), the learning rate increases to 1e-4 in in 100 steps, and then linearly decays to 0 in 500B tokens, and we keep a constant 4096 batch size. In this stage, we use the highest quality data, focusing on math, code, and reasoning.

Evaluation Benchmarks Our evaluation encompasses four primary categories of benchmarks, each designed to assess distinct capabilities of the model:

- **English Language Understanding and Reasoning**: MMLU(5-shot)(Hendrycks, Burns, Basart, et al. 2021), MMLU-pro(5-shot) (Wang et al. 2024), BBH(3-shot) (Suzgun et al. 2022), TriviaQA(5-shot) (Joshi et al. 2017)
- **Code Generation**: HumanEval(pass@1) (M. Chen et al. 2021), MBPP(pass@1)(Austin et al. 2021)
- **Mathematical Reasoning**: GSM8K(4-shot) (Cobbe et al. 2021) MATH (Hendrycks, Burns, Kadavath, et al. 2021), CMATH (Wei et al. 2023)
- **Chinese Language Understanding and Reasoning**: C-Eval(5-shot) (Y. Huang et al. 2023), CMMLU(5-shot)(H. Li et al. 2024)

Performance We named our model trained with Muon "Moonlight". We compared Moonlight with different public models on a similar scale. We first evaluated Moonlight at 1.2T tokens and compared it with the following models that have the same architecture and trained with comparable number of tokens:

- `Deepseek-v3-Small` (DeepSeek-AI et al. 2024) is a 2.4B/16B-parameter MoE model trained with 1.33T tokens;
- `Moonlight-A` follows the same training settings as Moonlight, except that it uses the AdamW optimizer.

For Moonlight and Moonlight-A, we used the intermediate 1.2T token checkpoint of the total 5.7T pretraining, where the learning rate is not decayed to minimal and the model has not gone through the cooldown stage yet.

Table 4: Comparison of different models at around 1.2T tokens.

	Benchmark (Metric)	DSV3-Small	Moonlight-A@1.2T	Moonlight@1.2T
	Activated Params†	2.24B	2.24B	2.24B
	Total Params†	15.29B	15.29B	15.29B
	Training Tokens	1.33T	1.2T	1.2T
	Optimizer	AdamW	AdamW	Muon
English	MMLU	53.3	60.2	**60.4**
	MMLU-pro	-	26.8	**28.1**
	BBH	41.4	**45.3**	43.2
	TriviaQA	-	57.4	**58.1**
Code	HumanEval	26.8	29.3	**37.2**
	MBPP	36.8	49.2	**52.9**
Math	GSM8K	31.4	43.8	**45.0**
	MATH	10.7	16.1	**19.8**
	CMath	-	57.8	**60.2**
Chinese	C-Eval	-	57.2	**59.9**
	CMMLU	-	58.2	**58.8**

† The reported parameter counts exclude the embedding parameters.

As shown in Table 4, Moonlight-A, our AdamW-trained baseline model, demonstrates strong performance compared to similar public models. Moonlight performs significantly better than Moonlight-A, proving the scaling effectiveness of Muon. We observed that Muon especially excels on Math and Code related tasks, and we encourage the research community to further investigate this phenomena. After Moonlight is fully trained to 5.7T tokens, we compared it with public models at similar scale and showed the results in Table 5:

- `LLAMA3-3B` from Grattafiori et al. 2024 is a 3B-parameter dense model trained with 9T tokens.
- `Qwen2.5-3B` from Yang et al. 2024 is a 3B-parameter dense model trained with 18T tokens.

Table 5: Comparison of different models on various benchmarks.

	Benchmark (Metric)	Llama3.2-3B	Qwen2.5-3B	DSV2-Lite	Moonlight
	Activated Param†	2.81B	2.77B	2.24B	2.24B
	Total Params†	2.81B	2.77B	15.29B	15.29B
	Training Tokens	9T	18T	5.7T	5.7T
	Optimizer	AdamW	Unknown	AdamW	Muon
English	MMLU	54.7	65.6	58.3	**70.0**
	MMLU-pro	25.0	34.6	25.5	**42.4**
	BBH	46.8	56.3	44.1	**65.2**
	TriviaQA‡	59.6	51.1	65.1	**66.3**
Code	HumanEval	28.0	42.1	29.9	**48.1**
	MBPP	48.7	57.1	43.2	**63.8**
Math	GSM8K	34.0	**79.1**	41.1	77.4
	MATH	8.5	42.6	17.1	**45.3**
	CMath	-	80.0	58.4	**81.1**
Chinese	C-Eval	-	75.0	60.3	**77.2**
	CMMLU	-	75.0	64.3	**78.2**

† The reported parameter counts exclude the embedding parameters.‡ We tested all listed models with the full set of TriviaQA.

- `Deepseek-v2-Lite` from DeepSeek-AI 2024 is a 2.4B/16B-parameter MOE model trained with 5.7T tokens.

As shown in Table 5, Moonlight outperforms models with similar architectures trained with an equivalent number of tokens. Even when compared to dense models trained on substantially larger datasets, Moonlight maintains competitive performance. Detailed comparisons can be found in Appendix E. The performance of Moonlight is further compared with other well-known language models on MMLU and GSM8k, as illustrated in Figure 1b and Appendix E Figure 8.[6]. Notably, Moonlight lies on the Pareto frontier of model performance versus training budget, outperforming many other models across various sizes.

3.4 Dynamics of Singular Spectrum

In order to validate the intuition that Muon can optimize the weight matrices in more diverse directions, we conducted a spectral analysis of the weight matrices trained with Muon and AdamW. For a weight matrix with singular values $\sigma = (\sigma_1, \sigma_2, \cdots, \sigma_n)$, we calculate the SVD entropy (Alter et al. 2000; Roy et al. 2007) of this matrix as follows:

$$H(\sigma) = -\frac{1}{\log n} \sum_{i=1}^{n} \frac{\sigma_i^2}{\sum_{j=1}^{n} \sigma_j^2} \log \frac{\sigma_i^2}{\sum_{j=1}^{n} \sigma_j^2}$$

As shown in Figure 4, we visualized the average SVD entropy of the weight matrices across different training checkpoints during pretraining with 1.2T tokens. We can see that across all training checkpoints and all groups of weight matrices, the SVD entropy of Muon is higher than that of AdamW, which verifies the intuition that Muon can provide a more diverse spectrum of updates for the weight matrices. This discrepancy is more significant in the router weights for expert selection, which indicates that mixture-of-expert models can benefit more from Muon.

Moreover, we visualized the singular value distributions of each weight matrix at the checkpoint trained with 1.2T tokens as demonstrated in Appendix F. We find that, for over 90% of the weight matrices, the SVD entropy when optimized by Muon is higher than that of AdamW, providing strong empirical evidence for Muon's superior capability in exploring diverse optimization directions.

3.5 Supervised Finetuning (SFT) with Muon

In this section, we present ablation studies on the Muon optimizer within the standard SFT stage of LLM training. Our findings demonstrate that the benefits introduced by Muon persist during the SFT stage. Specifically, a model that is both Muon-pretrained and Muon-finetuned outperforms others in the ablation studies. However, we also observe that when the SFT optimizer differs from the pretraining optimizer, SFT with Muon does not show a significant advantage over AdamW. This suggests that there is still considerable room for further exploration, which we leave for future work.

[6]Performance metrics and computational requirements (FLOPs) for baseline models are sourced from (OLMo et al. 2024)

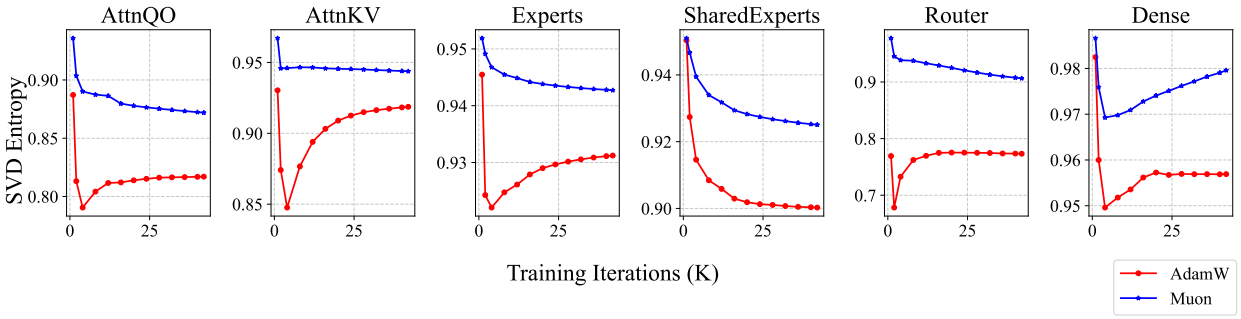


Figure 4: SVD entropy of weight matrices across different training iterations. We categorize the weight matrices into 6 different groups: 1) AttnQO denotes the weight matrices related to the query and output projection in the attention layer; 2) AttnKV denotes the weight matrices related to the key and value projection in the attention layer; 3) Experts denotes the weight matrices in expert models; 4) SharedExperts denotes the weight matrices in shared expert models; 5) Router denotes the weight matrices in the router; 6) Dense denotes the weight matrices in the first dense layer. The SVD entropy is calculated as the macro-average of the weight matrices in each group across all layers. For weights in expert models, we only calculate 3 out of 64 experts in different layers for efficiency.

3.5.1 Ablation Studies on the Interchangeability of Pretrain and SFT Optimizers

To further investigate Muon's potential, we finetuned Moonlight@1.2T and Moonlight-A@1.2T using both the Muon and AdamW optimizers. These models were finetuned for two epochs on the open-source tulu-3-sft-mixture dataset (Lambert et al. 2024), which contains 4k sequence length data. The learning rate followed a linear decay schedule, starting at 5×10^{-5} and gradually reducing to 0. The results, shown in Table 6, highlight the superior performance of Moonlight@1.2T compared to Moonlight-A@1.2T.

Table 6: Examining the impact of optimizer interchangeability between pretraining and SFT phases.

Benchmark (Metric)	**# Shots**	**Moonlight-1.2T**			
Pretraining Optimizer	-	Muon	AdamW	Muon	AdamW
SFT Optimzier	-	Muon	Muon	AdamW	AdamW
MMLU (EM)	0-shot (CoT)	**55.7**	55.3	50.2	52.0
HumanEval (Pass@1)	0-shot	**57.3**	53.7	52.4	53.1
MBPP (Pass@1)	0-shot	**55.6**	55.5	55.2	55.2
GSM8K (EM)	5-shot	**68.0**	62.1	64.9	64.6

3.5.2 SFT with Muon on public pretrained models

We further applied Muon to the supervised fine-tuning (SFT) of a public pretrained model, specifically the Qwen2.5-7B base model (Yang et al. 2024), using the open-source tulu-3-sft-mixture dataset (Lambert et al. 2024). The dataset was packed with an 8k sequence length, and we employed a cosine decay learning rate schedule, starting at 2×10^{-5} and gradually decreasing to 2×10^{-6}. The results are presented in Table 7. For comparison, we show that the Muon-finetuned model achieves performance on par with the Adam-finetuned model. These results indicate that for optimal performance, it is more effective to apply Muon during the pretraining phase rather than during supervised fine-tuning.

Table 7: Comparison of Adam and Muon optimizers applied to the SFT of the Qwen2.5-7B pretrained model.

Benchmark (Metric)	**# Shots**	**Adam-SFT**	**Muon-SFT**
Pretrained Model	-	Qwen2.5-7B	
MMLU (EM)	0-shot (CoT)	**71.4**	70.8
HumanEval (Pass@1)	0-shot	**79.3**	77.4
MBPP (Pass@1)	0-shot	**71.9**	71.6
GSM8K (EM)	5-shot	**89.8**	85.8

4 Discussions

There are several possible directions for future research that could further explore and expand upon the current findings.

Incorporating All Parameters into the Muon Framework Currently, the Muon optimizer is utilized in conjunction with the Adam optimizer, where certain parameters remain under the purview of Adam optimization. This hybrid approach, while functional, presents an opportunity for improvement. The integration of the optimization of all parameters exclusively within the Muon framework is a topic of significant research interest.

Extending Muon to Schatten Norms The Muon optimizer can be interpreted as the steepest descent method under the spectral norm. Given the broad applicability and versatility of Schatten norms, extending Muon to encompass the general Schatten norm is a promising direction. This extension may unlock additional optimization capabilities and potentially yield superior results compared to the current spectral norm-based implementation.

Understanding and Solving the Pretraining-Finetuning Mismatch A notable phenomenon observed in practice is the suboptimal performance of models pretrained with AdamW when fine-tuned with Muon, and vice versa. This optimizer mismatch presents a significant barrier to effectively leveraging the extensive repository of AdamW-pretrained checkpoints, thereby necessitating a rigorous theoretical investigation. A precise understanding of the underlying mechanisms is essential for devising robust and effective solutions.

5 Conclusions

In this technical report, we presented a comprehensive study on the scalability of Muon in LLM training. Through systematic analysis and improvements, we successfully applied Muon to a 3B/16B-parameter MoE model trained on 5.7 trillion tokens. Our results demonstrate that Muon can effectively replace AdamW as the standard optimizer for large-scale LLM training, offering significant advantages in both training efficiency and model performance. By open-sourcing our implementation, the Moonlight model, and intermediate training checkpoints, we aim to facilitate further research in scalable optimization techniques and accelerate the development of training methods for LLMs.

References

Alter, Orly, Patrick O. Brown, and David Botstein. "Singular value decomposition for genome-wide expression data processing and modeling". In: *Proceedings of the National Academy of Sciences* 97.18 (2000), pp. 10101–10106. DOI: 10.1073/pnas.97.18.10101. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.97.18.10101. URL: https://www.pnas.org/doi/abs/10.1073/pnas.97.18.10101.

Austin, Jacob et al. *Program Synthesis with Large Language Models*. 2021. arXiv: 2108.07732 [cs.PL]. URL: https://arxiv.org/abs/2108.07732.

Bernstein, Jeremy and Laker Newhouse. *Old Optimizer, New Norm: An Anthology*. 2024. arXiv: 2409.20325 [cs.LG]. URL: https://arxiv.org/abs/2409.20325.

Bi, Xiao et al. "Deepseek llm: Scaling open-source language models with longtermism". In: *arXiv preprint arXiv:2401.02954* (2024).

Cesista, Franz Louis. *Deep Learning Optimizers as Steepest Descent in Normed Spaces*. 2024. URL: http://leloykun.github.io/ponder/steepest-descent-opt/.

Chen, Mark et al. "Evaluating Large Language Models Trained on Code". In: (2021). arXiv: 2107.03374 [cs.LG].

Cobbe, Karl et al. *Training Verifiers to Solve Math Word Problems*. 2021. arXiv: 2110.14168 [cs.LG]. URL: https://arxiv.org/abs/2110.14168.

DeepSeek-AI. *DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model*. 2024. arXiv: 2405.04434 [cs.CL].

DeepSeek-AI et al. *DeepSeek-V3 Technical Report*. 2024. arXiv: 2412.19437 [cs.CL]. URL: https://arxiv.org/abs/2412.19437.

Franz, Louis Cesista. *The Case for Muon*. Oct. 2024. URL: https://x.com/leloykun/status/1846842887839125941 (visited on 02/18/2025).

Grattafiori, Aaron et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.

Hendrycks, Dan, Collin Burns, Steven Basart, et al. *Measuring Massive Multitask Language Understanding*. 2021. arXiv: 2009.03300 [cs.CY]. URL: https://arxiv.org/abs/2009.03300.

Hendrycks, Dan, Collin Burns, Saurav Kadavath, et al. *Measuring Mathematical Problem Solving With the MATH Dataset*. 2021. arXiv: 2103.03874 [cs.LG]. URL: https://arxiv.org/abs/2103.03874.

Hoffmann, Jordan et al. *Training Compute-Optimal Large Language Models*. 2022. arXiv: 2203.15556 [cs.CL]. URL: https://arxiv.org/abs/2203.15556.

Huang, Yuzhen et al. *C-Eval: A Multi-Level Multi-Discipline Chinese Evaluation Suite for Foundation Models*. 2023. arXiv: 2305.08322 [cs.CL]. URL: https://arxiv.org/abs/2305.08322.

Jordan, Keller et al. *Muon: An optimizer for hidden layers in neural networks*. 2024. URL: https://kellerjordan.github.io/posts/muon/.

Joshi, Mandar et al. *TriviaQA: A Large Scale Distantly Supervised Challenge Dataset for Reading Comprehension*. 2017. arXiv: 1705.03551 [cs.CL]. URL: https://arxiv.org/abs/1705.03551.

Kaplan, Jared et al. *Scaling Laws for Neural Language Models*. 2020. arXiv: 2001.08361 [cs.LG]. URL: https://arxiv.org/abs/2001.08361.

Kingma, Diederik P. and Jimmy Ba. "Adam: A Method for Stochastic Optimization". In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

Lambert, Nathan et al. "Tülu 3: Pushing Frontiers in Open Language Model Post-Training". In: (2024).

Li, Haonan et al. *CMMLU: Measuring massive multitask language understanding in Chinese*. 2024. arXiv: 2306.09212 [cs.CL]. URL: https://arxiv.org/abs/2306.09212.

Li, Xi-Lin. "Preconditioned Stochastic Gradient Descent". In: *IEEE Transactions on Neural Networks and Learning Systems* 29.5 (May 2018), pp. 1454–1466. ISSN: 2162-2388. DOI: 10.1109/tnnls.2017.2672978. URL: http://dx.doi.org/10.1109/TNNLS.2017.2672978.

– *Preconditioner on Matrix Lie Group for SGD*. 2018. arXiv: 1809.10232 [stat.ML]. URL: https://arxiv.org/abs/1809.10232.

– *Stochastic Hessian Fittings with Lie Groups*. 2024. arXiv: 2402.11858 [stat.ML]. URL: https://arxiv.org/abs/2402.11858.

Li, Xilin. *Black Box Lie Group Preconditioners for SGD*. 2022. arXiv: 2211.04422 [stat.ML]. URL: https://arxiv.org/abs/2211.04422.

Liu, Hong et al. "Sophia: A Scalable Stochastic Second-order Optimizer for Language Model Pre-training". In: *The Twelfth International Conference on Learning Representations*. 2024. URL: https://openreview.net/forum?id=3xHDeA8Noi.

Loshchilov, Ilya and Frank Hutter. "Decoupled Weight Decay Regularization". In: *International Conference on Learning Representations*. 2019. URL: https://openreview.net/forum?id=Bkg6RiCqY7.

OLMo, Team et al. "2 OLMo 2 Furious". In: *arXiv preprint arXiv:2501.00656* (2024).
OpenAI et al. *GPT-4 Technical Report*. 2024. arXiv: 2303.08774 [cs.CL]. URL: https://arxiv.org/abs/2303.08774.
Pethick, Thomas et al. *Training Deep Learning Models with Norm-Constrained LMOs*. 2025. arXiv: 2502.07529 [cs.LG]. URL: https://arxiv.org/abs/2502.07529.
Pooladzandi, Omead and Xi-Lin Li. *Curvature-Informed SGD via General Purpose Lie-Group Preconditioners*. 2024. arXiv: 2402.04553 [cs.LG]. URL: https://arxiv.org/abs/2402.04553.
Rajbhandari, Samyam et al. "ZeRO: Memory optimizations Toward Training Trillion Parameter Models". In: (Nov. 2020), pp. 1–16. DOI: 10.1109/sc41405.2020.00024. URL: http://dx.doi.org/10.1109/SC41405.2020.00024.
Roy, Olivier and Martin Vetterli. "The effective rank: A measure of effective dimensionality". In: *2007 15th European Signal Processing Conference*. 2007, pp. 606–610.
Shazeer, Noam. *Fast Transformer Decoding: One Write-Head is All You Need*. 2019. arXiv: 1911.02150 [cs.NE]. URL: https://arxiv.org/abs/1911.02150.
Shoeybi, Mohammad et al. *Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism*. 2020. arXiv: 1909.08053 [cs.CL]. URL: https://arxiv.org/abs/1909.08053.
Suzgun, Mirac et al. *Challenging BIG-Bench Tasks and Whether Chain-of-Thought Can Solve Them*. 2022. arXiv: 2210.09261 [cs.CL]. URL: https://arxiv.org/abs/2210.09261.
Team, Gemini et al. *Gemini: A Family of Highly Capable Multimodal Models*. 2024. arXiv: 2312.11805 [cs.CL]. URL: https://arxiv.org/abs/2312.11805.
Team, Gemma et al. "Gemma 2: Improving open language models at a practical size". In: *arXiv preprint arXiv:2408.00118* (2024).
Team, Kimi. "Kimi k1.5: Scaling Reinforcement Learning with LLMs". In: (2025).
Vyas, Nikhil et al. "SOAP: Improving and Stabilizing Shampoo using Adam". In: *The Thirteenth International Conference on Learning Representations*. 2025. URL: https://openreview.net/forum?id=IDxZhXrpNf.
Wang, Yubo et al. *MMLU-Pro: A More Robust and Challenging Multi-Task Language Understanding Benchmark*. 2024. arXiv: 2406.01574 [cs.CL]. URL: https://arxiv.org/abs/2406.01574.
Wei, Tianwen et al. *CMATH: Can Your Language Model Pass Chinese Elementary School Math Test?* 2023. arXiv: 2306.16636 [cs.CL]. URL: https://arxiv.org/abs/2306.16636.
Yang, An et al. "Qwen2.5 Technical Report". In: *arXiv preprint arXiv:2412.15115* (2024).
You, Jiacheng. *Jiacheng You's discussion on Muon's Update RMS*. 2025. URL: https://x.com/YouJiacheng/status/1890094769386451309.
Yuan, Huizhuo et al. *MARS: Unleashing the Power of Variance Reduction for Training Large Models*. 2024. arXiv: 2411.10438 [cs.LG].

A Update RMS

Proof of Lemma 1

Proof. Without loss of generality, consider the orthogonal matrices $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{m\times m}$ where $n \geq m \geq r$. We will show that for $X = U_{[:,:r]}V_{[:r,:]}$ (the update of the Muon has the same format), the RMS value is $\sqrt{r/mn}$. From the definition of matrix multiplication:

$$X_{i,j} = \sum_{k=1}^{r} U_{i,k}V_{k,j}$$

The RMS can be expressed as:

$$\begin{aligned}
\text{RMS}(X)^2 &= \frac{1}{mn}\sum_{i=1}^{n}\sum_{j=1}^{m}\sum_{k=1}^{r} U_{i,k}^2 V_{k,j}^2 \\
&= \frac{1}{mn}\sum_{k=1}^{r}\left(\sum_{i=1}^{n} U_{i,k}^2\right)\left(\sum_{j=1}^{m} V_{k,j}^2\right) \\
&= \frac{1}{mn}\sum_{k=1}^{r} 1 \\
&= \frac{r}{mn}
\end{aligned}$$

Therefore, $\text{RMS}(X) = \sqrt{r/mn}$. For the common case where the matrices are full-rank, $r = m$, yielding $\text{RMS}(X) = \sqrt{1/n}$. □

Consistent Update RMS Across Muon and AdamW As discussed in 2.2, we'd like to match the update RMS between Muon and AdamW optimizers. This is validated by experiments on small-scale models. We set Muon's Update RMS in the range of $[0.05, 0.1, 0.2, 0.4, 0.8]$ and AdamW as baseline. We reported the loss and representative weight matrix RMS at 2k steps (about 2B tokens) in the Table 8. From the results, we find that 0.2 RMS and 0.4 RMS performed similarly and much better than other settings. These findings are consistent with our empirical observation that AdamW's update RMS is in the range of $0.2 \sim 0.4$. We opted to control the update RMS of Muon to 0.2.

Table 8: Muon Update RMS Experiments

Optimizer	AdamW	0.05 RMS*	0.1 RMS	0.2 RMS	0.4 RMS	0.8 RMS
LM training loss	3.512	3.355	3.239	**3.198**	3.199	3.386
LM validation loss	3.679	3.503	3.374	3.325	**3.314**	3.543
AttnQ weight RMS	1.01e-2	5.74e-3	8.44e-3	1.57e-2	2.95e-2	7.23e-2
Mlp weight RMS	1.25e-2	8.01e-3	1.27e-2	2.35e-2	4.51e-2	8.73e-2

*Except the first column, all other candidates are using Muon with controlled RMS.

B AdamW Baseline Scaling Law

To ensure the fairness and accuracy of our experiments, we conducted a series of experiments on our proprietary dataset to derive scaling law parameters that are optimal for AdamW. This includes determining the optimal model size(N), number of training tokens(D), learning rate(η), batch size(B) under a constrained computational budget (FLOPs, C). (Kaplan et al. 2020; Hoffmann et al. 2022; Bi et al. 2024) Table 9 presents the results of our systematic parameter search process.

Table 9: Empirical Relationships Between Scaling Law Parameters and Computational Budget (FLOPs)

$N(C)$	$D(C)$	$\eta(C)$	$B(C)$
$0.0483359 \cdot C^{0.5112684}$	$3.4480927 \cdot C^{0.4887316}$	$0.0127339 \cdot C^{-0.0574752}$	$0.0065202 \cdot C^{0.4137915}$

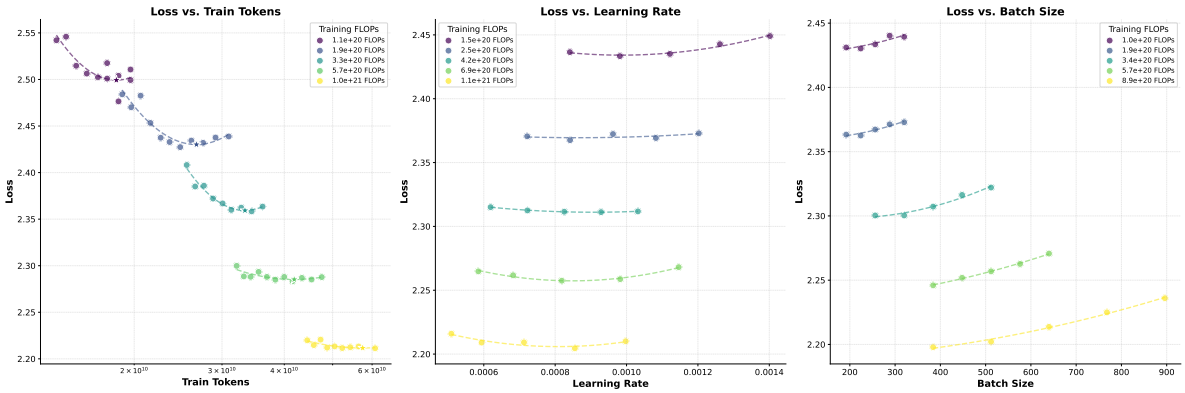


Figure 5: Optimization Landscapes for Scaling Law Hyper-parameters Across FLOPs Budgets

Hyper-Parameters Search To systematically identify optimal scaling law hyper-parameters in the AdamW baseline, we adopted a multistage search protocol. First, we selected multiple computational budgets (FLOPs levels) and initialized model sizes, learning rates, and batch sizes based on empirical guidelines from prior studies. For each fixed FLOPs constraint, we varied the model size N while adjusting the training token count D inversely to maintain $C = 6ND$, thereby exploring the trade-off between model capacity and data efficiency. Each configuration was trained to convergence, and the validation loss was recorded to determine the Pareto-optimal combinations of N and D. Subsequently, with the optimal $N - D$ pairs fixed, we refined the learning rate and batch size through grid searches, ensuring stability and convergence across configurations. To mitigate local minima and enhance robustness, this iterative procedure was repeated 2–3 times, progressively narrowing the hyper-parameter space.

The optimization process is further illustrated in Figure 5, which depicts the loss landscapes as functions of training tokens, learning rate, and batch size across varying FLOPs budgets. Each bowl-shaped curve represents the loss surface for a specific FLOPs level, with a distinct global minimum corresponding to the optimal hyper-parameter configuration.

C Model Architecture

Muon is agnostic to model architectures, and we used a model similar to Deepseek-V3-Small as described in DeepSeek-AI et al. 2024, because it is a strong model with open weights as a baseline. We made several small modifications in the Moonlight model and listed them here:

Multi-token Prediction (MTP) MTP has not shown significant benefits to pretraining in our experiments. For simplicity, we do not introduce MTP layers into the Moonlight model.

Auxfree Bias Update In DeepSeek-AI et al. 2024, auxfree bias is updated by: $b_i = b_i + u \times \text{sign}(e_i)$, where u is the update ratio, b_i is the bias for the ith expert, and e_i is the expert's violating ratio. We slightly modified the update rule as: $b_i = b_i + u \times (\text{sign}(e_i) - \text{sign}(e).\text{mean}())$, where $\text{sign}(e).\text{mean}()$ is the average of the signs of all expert's violating ratio, in order to control the magnitude of the bias, while does not change the topk selection logic.

Gate Scaling Factor Deepseek-V2-Lite did not use the gate scaling factor, and Deepseek-V3 used a scaling factor of 2.5. We used a scaling factor of 2.446 to control a similar output rms like dense models. The code for calculating our gate scaling factor can be found in Figure 6.

D Training Stability

No Loss or Grad Norm Spike The Moonlight training process was very smooth and we did not meet any loss spike or gradient norm spike. The loss and grad norm curve can be seen in Figure 7 (Moonlight is colored in blue and Moonlight-A trained by AdamW is colored in red)

Max Attention Logit During training, we observed that while both the training loss and gradient norm remained stable throughout the process, the maximum attention logit (computed as the single largest logit value across the global

```python
import numpy as np

def sigmoid(x):
    return 1 / (1 + np.exp(-x))

def calc_gate_scaling_factor(num_experts: int, topk: int, iter_times: int):
    """Calculate the gate scaling factor for MoE.

    Args:
        num_experts (int): The number of experts.
        topk (int): The number of experts to select.
        iter_timers (int): The number of iterations.

    Returns:
        float: The gate scaling factor.
    """
    factors = []
    for _ in range(iter_times):

        # mock gaussian logits
        logits = np.random.randn(num_experts)
        # select topk logits
        p = np.sort(sigmoid(logits))[::-1]
        p = p[:topk]
        # renormalize
        p = p / p.sum()
        # calculate the scaling factor
        factors.append( 1/ (p**2).sum()**0.5)
    return np.mean(factors)
```

Figure 6: Python implementation for calculating the gate scaling factor.

batch) exhibited a distinct upward trajectory in specific layers during the initial training phase, exceeding a threshold of 100. Notably, AdamW demonstrated healthier behavior in controlling this metric compared to alternative optimizers.

To further investigate the impacts of this phenomenon, we introduced the large attention logits ratio metric, defined as the proportion of attention logits exceeding 100 within a batch. As shown in Fig.7, this ratio remained consistently low (about 10^{-4}), indicating that extreme large logit values were sparse. Furthermore, the maximum logit values gradually decrease as training progressed, suggesting that the optimization dynamics become healthier.

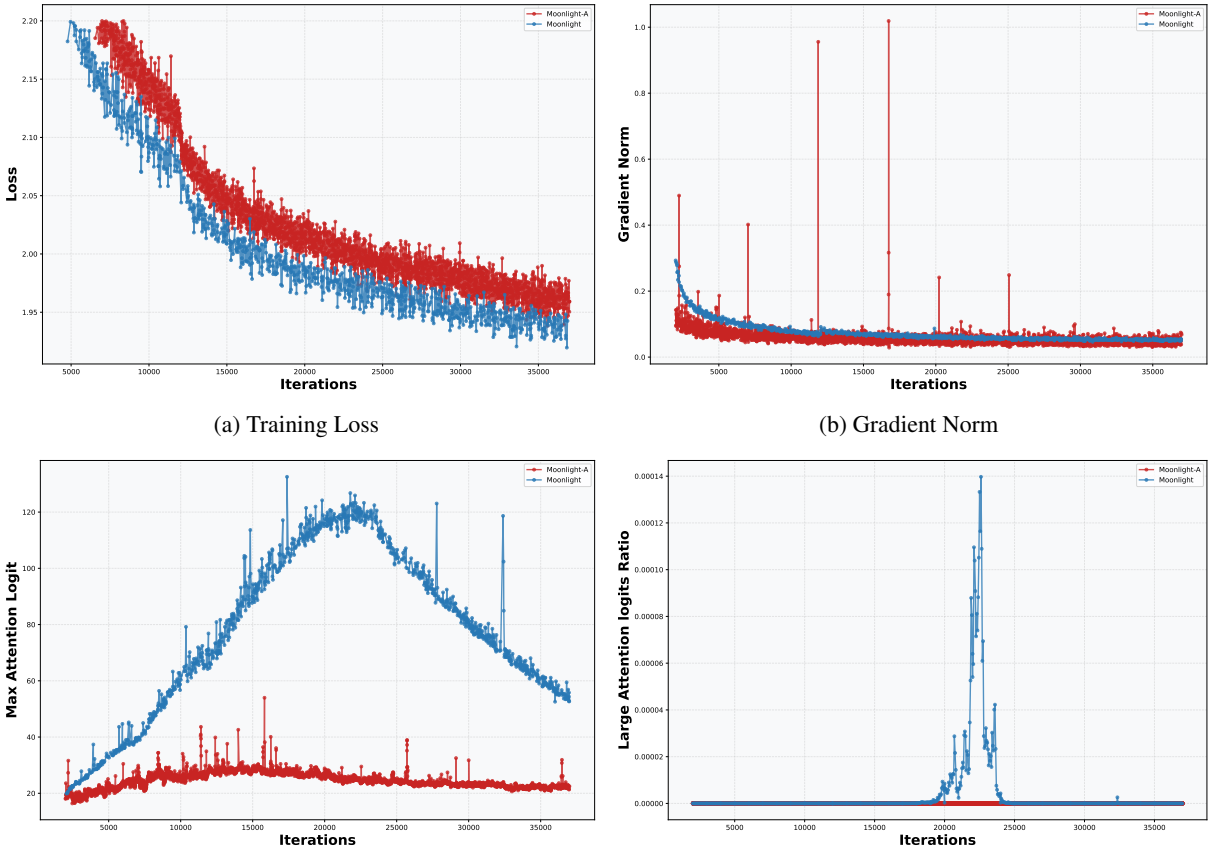
RMSNorm Gamma Weight Decay It is noteworthy that applying weight decay to the RMSNorm gamma parameter is crucial for ensuring training stability, as it effectively prevents excessively high output RMS values in each layer.

E Comparison with More Expensive Models

Table 10 presents a comparative analysis between our Moonlight model (optimized with Muon) and publicly available models trained with greater computational resources, including `LLama3.1-8B` (Grattafiori et al. 2024), `Gemma-9B` (Gemma Team et al. 2024) and `Qwen2.5-7B` (Yang et al. 2024). Figure 8 illustrates the GSM8k performance benchmarks of Moonlight against comparable models in the field.

F Singular Value Distributions of Weight Matrices

We visualize the singular value distributions of weight matrices by plotting a line graph of its singular values in descending order for each matrix, normalized by the largest one. As shown in Figures 9 and 10, we find that, for most of the weight matrices, the singular value distributions of them optimized by Muon are more flattened than that of AdamW, which further confirms the hypothesis that Muon can provide a more diverse spectrum of updates.



(a) Training Loss

(b) Gradient Norm

(c) Max Attention Logit (Layer 1)

(d) Large Attention Logits Ratio (Layer 1)

Figure 7: Training dynamics comparison between Moonlight and Moonlight-A

Table 10: Comparison of different models on various benchmarks.

	Benchmark (Metric)	**Moonlight**	**LLAMA3.1-8B**	**Gemma2-9B**	**Qwen2.5-7B**
			Larger Training Compute Model		
	Activated Param†	2.24B	7.38B	8.32B	6.83B
	Total Params†	15.29B	7.38B	8.32B	6.83B
	Training Tokens	5.7T	15T	8T	18T
	Optimizer	Muon	AdamW	Unknown	Unknown
English	MMLU	70.0	66.7	71.3	74.2
	MMLU-pro	42.4	37.1	44.7	45.0
	BBH	65.2	57.7	68.2	70.4
	TriviaQA‡	66.3	70.3	-	60.0
Code	HumanEval	48.1	37.2	37.8	57.9
	MBPP	63.8	47.6	62.2	74.9
Math	GSM8K	77.4	57.2	70.7	85.4
	MATH	45.3	20.3	37.7	49.8

† The reported parameter counts exclude the embedding parameters.‡ We test all listed models with the full set of TriviaQA.

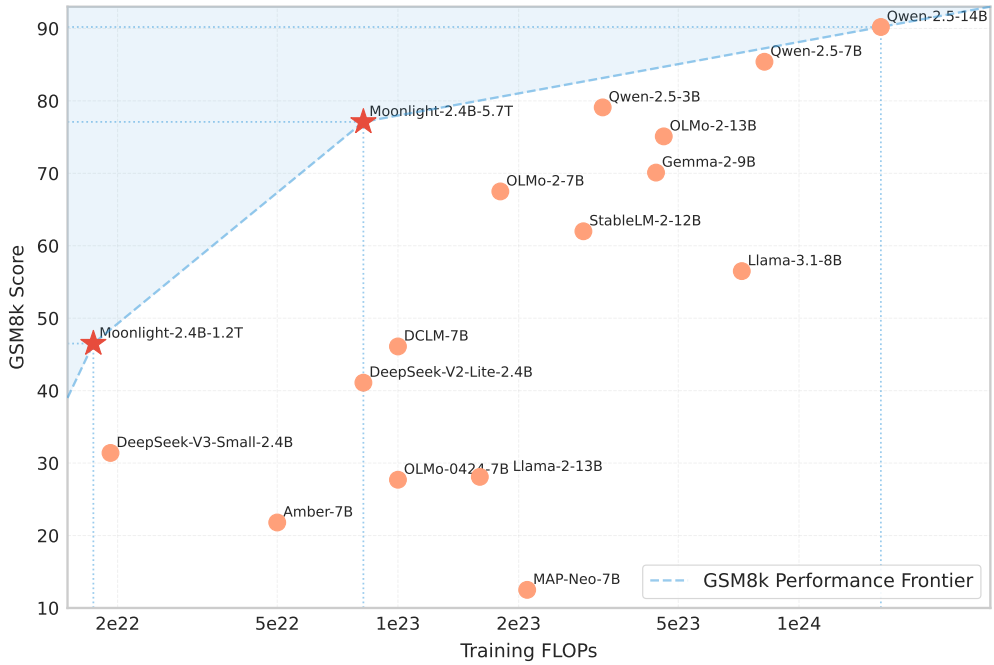


Figure 8: The GSM8k performance of our Moonlight model optimized with Muon and other comparable models.

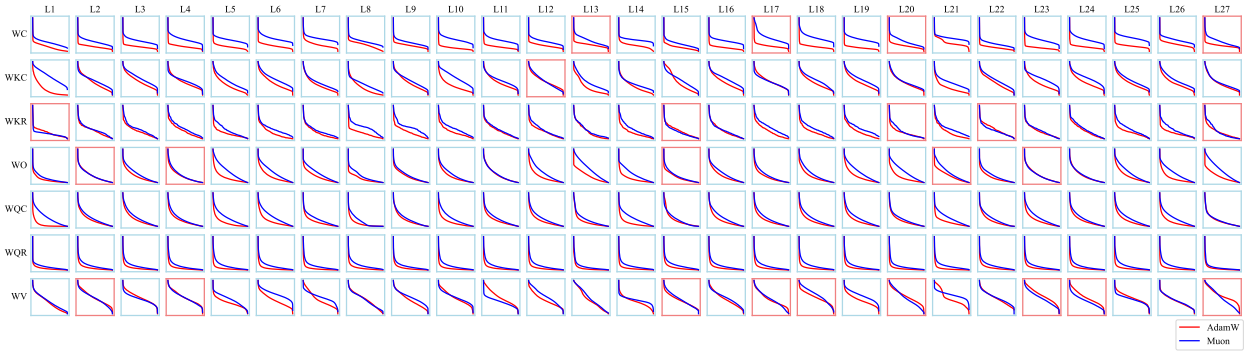


Figure 9: Distribution of singular values for each weight matrix in the attention layers. We use WC to denote the weight matrices at each layer that compress the hidden states to the shared latent spaces for keys and values, WV to denote the weight matrices up-projecting the values from the latent space, WO to denote the output projection matrices, and WKR, WKC, WQR and WQC to denote the projection matrices for the part of keys and queries with and without RoPE respectively. We set the spines of each line graph red if the corresponding weight matrix optimized by Muon has a lower singular entropy than AdamW.

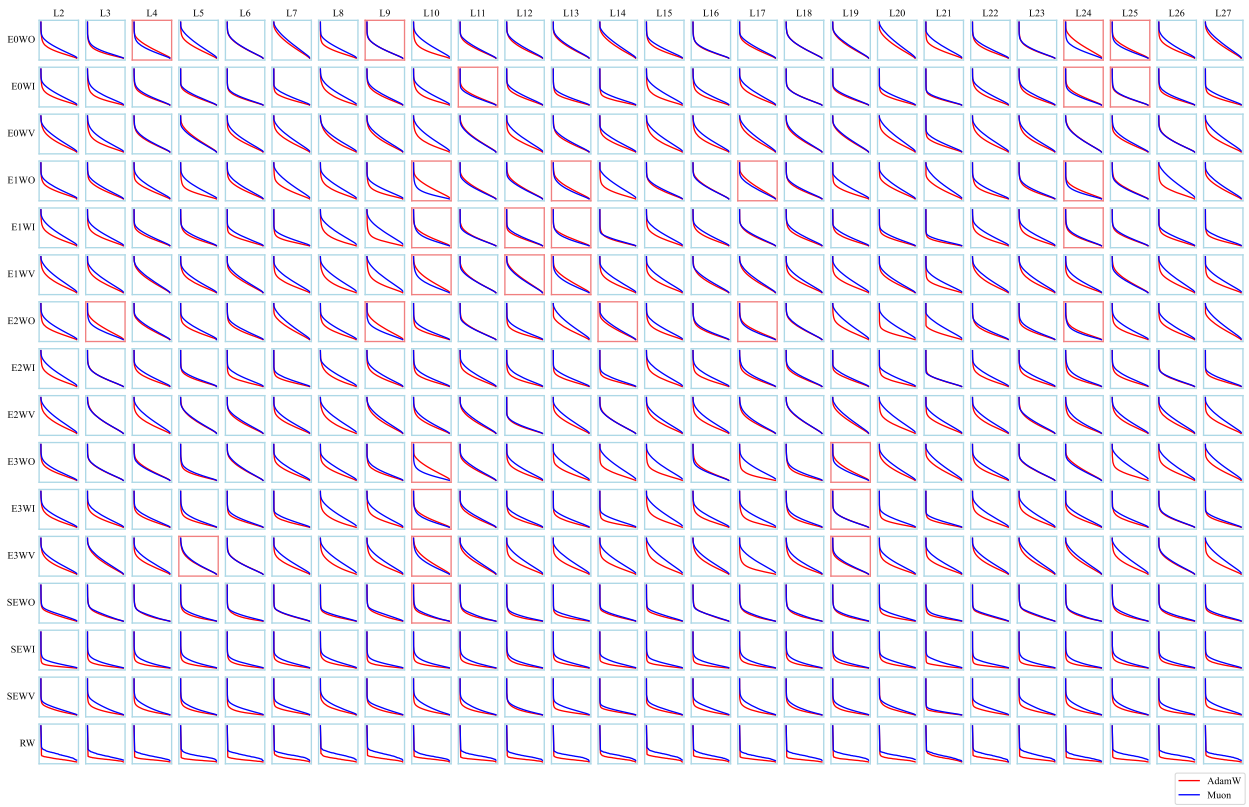


Figure 10: Distribution of singular values for each weight matrix in the feed-forward network (FFN) layers. We use WI, WV and WO to denote the weight matrices involved in the FFN layer with SwiGLU activation function, where WI represents the input projection to the Swish_1 function, WV represents the extra input projection interacting with Swish_1 activations, and WO represents the output projection. We use E0, E2, E3 to denote three arbitrarily selected expert models and SE to denote the weights in the shared expert model. We use RW to denote the weights in the router. We set the spines of each line graph red if the corresponding weight matrix optimized by Muon has a lower singular entropy than AdamW.