MEQMUON: MATRIX-EQUILIBRATING MUON FOR LLM PRETRAINING

Anonymous authors
Paper under double-blind review

ABSTRACT

The success of large language models (LLMs) has been accompanied by continued growth in model size and pretraining costs. Muon offers high accuracy and training efficiency in LLM pretraining. Recent work introduces row-wise normalization into Muon to balance update magnitudes and improve pretraining performance. However, row-wise normalization alone cannot accommodate different imbalance patterns in update matrices. In this paper, we propose an improved Muon optimizer, called matrix-equilibrating Muon (MeqMuon), for LLM pretraining. MeqMuon balances both row and column magnitudes through normalization that can be automatically tailored to different imbalance patterns without manual intervention. Moreover, MeqMuon eliminates the need to store AdamW's second-moment estimates, reducing optimizer-state memory usage. Empirical results demonstrate that MeqMuon achieves better convergence performance than AdamW, Muon, and other baselines in LLM pretraining.

1 INTRODUCTION

The rapid development of large language models (LLMs) has enabled advances across diverse domains, such as code generation (Chen et al., 2021; DeepSeek-AI, 2026), mathematical problem solving (Shao et al., 2025), and protein structure prediction (Lin et al., 2023; Candido et al., 2026). These advances have been accompanied by continued growth in model size. For example, DeepSeek-V4-Pro (DeepSeek-AI, 2026) has 1.6 trillion total parameters and was pretrained on 33 trillion tokens, while Kimi K3 (Kimi Team, 2026) reaches 2.8 trillion total parameters. Pretraining models at such scales incurs substantial cost. Efficient optimizer design is therefore important for improving model performance and reducing training cost (Loshchilov & Hutter, 2019; Shazeer & Stern, 2018; Dettmers et al., 2022; Chen et al., 2023; Jordan et al., 2024; Liu et al., 2025; Pagliardini et al., 2025; Zhang et al., 2025; Zhao et al., 2024a; Zhu et al., 2025).

Adam (Kingma & Ba, 2015) has long been the dominant optimizer for LLM pretraining. It uses first- and second-moment estimates of gradients to compute coordinate-wise adaptive updates. AdamW (Loshchilov & Hutter, 2019), a variant of Adam with decoupled weight decay, has been used to pretrain LLMs such as Llama 2 (Touvron et al., 2023) and DeepSeek-V3 (Liu et al., 2024). Recently, Muon (Jordan et al., 2024; Liu et al., 2025) has demonstrated substantial gains over AdamW in training efficiency and model accuracy. Its effectiveness has also been demonstrated in pretraining frontier LLMs such as DeepSeek-V4 (DeepSeek-AI, 2026), Kimi K2 (Kimi Team, 2025), and Kimi K3 (Kimi Team, 2026). Muon applies different update rules to 2D weights in hidden layers and the remaining parameters. For 2D weights in hidden layers, it orthogonalizes momentum matrices to bring their nonzero singular values toward one. The remaining parameters, such as token embedding matrices, language modeling (LM) heads, and 1D parameters, are updated with AdamW.

Normalization is a widely used technique in optimizer design. The convergence properties of normalized gradient methods have been studied theoretically (Levy, 2016; Murray et al., 2019; Zhang et al., 2020; Cutkosky & Mehta, 2020; Zhao et al., 2024b; Yang et al., 2024b; Sun et al., 2025). In LLM pretraining, normalization has been incorporated into optimizers in different forms. AdamW (Loshchilov & Hutter, 2019) normalizes first-moment estimates coordinate-wise using the square roots of second-moment estimates. Muon (Jordan et al., 2024; Liu et al., 2025) normalizes singular values by approximately orthogonalizing momentum matrices for 2D weights in hidden layers. SCALE (Glentis et al., 2026) normalizes the update vector associated with each output dimension

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.

for all 2D weights. Li et al. (2026) observe substantial imbalance among row magnitudes in some of Muon's orthogonalized updates. They propose NorMuon, which applies Adam-style row-wise normalization to mitigate this imbalance and achieves better convergence performance than Muon in LLM pretraining. However, our observation shows that Muon's orthogonalized updates can exhibit imbalance in either row or column magnitudes. For example, some updates may have relatively balanced row magnitudes but imbalanced column magnitudes, while others may have relatively balanced column magnitudes but imbalanced row magnitudes. Row-wise normalization alone therefore cannot accommodate different imbalance patterns in update matrices.

In this paper, we propose an improved Muon optimizer, called matrix-equilibrating Muon (MeqMuon), for LLM pretraining. The main contributions of this paper are outlined as follows:

- We identify different imbalance patterns of row and column magnitudes in Muon. Muon's orthogonalized updates can exhibit imbalance predominantly in either row or column magnitudes. Unorthogonalized momentum matrices for the remaining 2D parameters exhibit imbalance in both row and column magnitudes.
- We propose MeqMuon, which balances both row and column magnitudes through normalization that can be automatically tailored to different imbalance patterns without manual intervention. MeqMuon selects row-wise or column-wise normalization for Muon's orthogonalized updates and applies two-sided normalization to unorthogonalized momentum matrices for the remaining 2D parameters.
- MeqMuon eliminates second-moment storage and reduces optimizer-state memory usage by replacing AdamW updates with normalized momentum for the remaining parameters.
- Empirical results demonstrate that MeqMuon achieves better convergence performance than AdamW, Muon, and other baselines in LLM pretraining.

2 PRELIMINARIES

Problem formulation. LLM pretraining is formulated as the following optimization problem

$$\min_{\boldsymbol{\theta}} \ \mathcal{L}(\boldsymbol{\theta}), \tag{1}$$

where $\mathcal{L}$ is the training loss and $\boldsymbol{\theta}$ denotes the collection of all model parameter tensors. LLM parameters are predominantly 1D or 2D. Higher-dimensional parameter tensors are reshaped or partitioned into 2D matrices for optimization in Muon implementations (Jordan et al., 2024). We therefore focus on 1D and 2D parameters in this paper.

AdamW. Adam (Kingma & Ba, 2015) maintains first- and second-moment estimates of the mini-batch gradient $\boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}_t)$ using exponential moving averages, where t denotes the iteration number. Starting from $\boldsymbol{m}_0 = \boldsymbol{v}_0 = \mathbf{0}$, Adam follows the update rules below:

$$\begin{aligned} \boldsymbol{m}_t &= \beta_1 \boldsymbol{m}_{t-1} + (1-\beta_1)\boldsymbol{g}_t, & \widehat{\boldsymbol{m}}_t &= \frac{\boldsymbol{m}_t}{1-\beta_1^t}, \\ \boldsymbol{v}_t &= \beta_2 \boldsymbol{v}_{t-1} + (1-\beta_2)(\boldsymbol{g}_t \odot \boldsymbol{g}_t), & \widehat{\boldsymbol{v}}_t &= \frac{\boldsymbol{v}_t}{1-\beta_2^t}, \\ \boldsymbol{\theta}_{t+1} &= \boldsymbol{\theta}_t - \eta_t \frac{\widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{v}}_t} + \epsilon}. \end{aligned} \tag{2}$$

Here $\beta_1, \beta_2 \in [0, 1)$ are exponential decay rates, η_t is the learning rate, and $\epsilon > 0$ is a constant for numerical stability. The symbol $\odot$ denotes element-wise multiplication.

In LLM pretraining, AdamW (Loshchilov & Hutter, 2019) is widely used in place of Adam. AdamW modifies Adam through decoupled weight decay:

$$\boldsymbol{\theta}_{t+1} = (1 - \eta_t\lambda)\boldsymbol{\theta}_t - \eta_t \frac{\widehat{\boldsymbol{m}}_t}{\sqrt{\widehat{\boldsymbol{v}}_t} + \epsilon}, \tag{3}$$

where $\lambda \geq 0$ is the weight decay coefficient.

Muon. Muon (Jordan et al., 2024) orthogonalizes momentum matrices for 2D weights in hidden layers. A weight matrix $\boldsymbol{W}_t \in \mathbb{R}^{m\times n}$ in a hidden layer with minibatch gradient $\boldsymbol{G}_t$ is updated as

$$\boldsymbol{B}_t = \mu \boldsymbol{B}_{t-1} + \boldsymbol{G}_t, \qquad \boldsymbol{U}_t = \mathit{Orth}(\boldsymbol{B}_t), \qquad \boldsymbol{W}_{t+1} = \boldsymbol{W}_t - \eta_t \boldsymbol{U}_t, \tag{4}$$

where $\boldsymbol{B}_{-1} = \boldsymbol{0}$, $\boldsymbol{B}_t$ is the momentum matrix, and $\mu \in [0, 1)$ is the momentum coefficient. For $\boldsymbol{H} \in \mathbb{R}^{m\times n}$ with SVD $\boldsymbol{H} = \boldsymbol{P}\boldsymbol{\Sigma}\boldsymbol{Q}^\top$ restricted to nonzero singular values, $\mathit{Orth}(\boldsymbol{H})$ is defined as $\boldsymbol{P}\boldsymbol{Q}^\top$. In practice, *Orth* is approximated by K Newton–Schulz (NS) iterations. For a nonzero input $\boldsymbol{H}$, the quintic iteration starts from $\boldsymbol{X}_0 = \boldsymbol{H}/\|\boldsymbol{H}\|_F$ and takes the form

$$\boldsymbol{A}_k = \boldsymbol{X}_k \boldsymbol{X}_k^\top, \qquad \boldsymbol{X}_{k+1} = a\boldsymbol{X}_k + (b\boldsymbol{A}_k + c\boldsymbol{A}_k^2)\boldsymbol{X}_k, \qquad k = 0, \ldots, K-1. \tag{5}$$

Existing works (Jordan et al., 2024; Liu et al., 2025) commonly use $(a, b, c) = (3.4445, -4.7750, 2.0315)$ and $K = 5$. Muon applies these orthogonalized updates only to 2D weights in hidden layers, while the remaining parameters, such as token embedding matrices, LM heads, and 1D parameters, are updated with AdamW.

For large-scale LLM pretraining, Liu et al. (2025) introduced decoupled weight decay and root mean square (RMS) alignment into Muon. These modifications were later adopted in Kimi K2 (Kimi Team, 2025) and DeepSeek-V4 (DeepSeek-AI, 2026). With these modifications, the update rule in (4) becomes

$$\boldsymbol{B}_t = \mu \boldsymbol{B}_{t-1} + \boldsymbol{G}_t, \quad \boldsymbol{U}_t = \mathit{Orth}(\boldsymbol{B}_t), \quad \boldsymbol{W}_{t+1} = (1 - \eta_t\lambda)\boldsymbol{W}_t - \eta_t \rho \sqrt{\max(m, n)}\, \boldsymbol{U}_t, \tag{6}$$

where $\lambda \geq 0$ is the weight decay coefficient. $\sqrt{\max(m, n)}$ compensates for the effect of matrix shape on the update RMS. The RMS alignment coefficient $\rho > 0$ sets the target update RMS and is chosen to approximately match that of AdamW. Moonlight and Kimi K2 use $\rho = 0.2$ (Liu et al., 2025; Kimi Team, 2025), while DeepSeek-V4 uses $\rho = 0.18$ (DeepSeek-AI, 2026).

3 METHOD

In this section, we first characterize the different imbalance patterns in row and column magnitudes and then introduce our proposed method, MeqMuon.

3.1 IMBALANCE PATTERNS

To characterize imbalance in matrix updates, we first define statistics for their row and column magnitudes. We follow the matrix-layout conventions used in PyTorch implementations. For linear weight matrices, rows correspond to output features and columns correspond to input features. For token embedding matrices and LM heads, rows correspond to vocabulary entries and columns correspond to hidden features. For a matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$, the row-wise and column-wise root mean square (RMS) magnitudes are collected in vectors $\boldsymbol{r} \in \mathbb{R}^m$ and $\boldsymbol{c} \in \mathbb{R}^n$, respectively, with entries defined as

$$r_i = \frac{\|\boldsymbol{X}_{i,:}\|_2}{\sqrt{n}}, \quad i = 1, \ldots, m, \qquad c_j = \frac{\|\boldsymbol{X}_{:,j}\|_2}{\sqrt{m}}, \quad j = 1, \ldots, n. \tag{7}$$

Imbalance in row and column magnitudes is quantified by the coefficients of variation (CVs) of $\boldsymbol{r}$ and $\boldsymbol{c}$, denoted by γ_r and γ_c, respectively. Each CV is defined as the ratio of the standard deviation to the mean:

$$\begin{aligned} \bar{r} &= \frac{1}{m_+} \sum_{i:\, r_i > 0} r_i, \quad \gamma_r = \frac{\sqrt{\frac{1}{m_+}\sum_{i:\, r_i>0}(r_i - \bar{r})^2}}{\bar{r}}, \\ \bar{c} &= \frac{1}{n_+} \sum_{j:\, c_j > 0} c_j, \quad \gamma_c = \frac{\sqrt{\frac{1}{n_+}\sum_{j:\, c_j>0}(c_j - \bar{c})^2}}{\bar{c}}, \end{aligned} \tag{8}$$

where m_+ and n_+ denote the numbers of nonzero entries in $\boldsymbol{r}$ and $\boldsymbol{c}$, respectively. Zero rows and columns are excluded from the corresponding CV calculations. In our implementation, RMS values at or below $\delta = 10^{-7}$ are treated as zero. The CVs are dimensionless and invariant to uniform scaling. Larger CVs indicate greater imbalance. Accordingly, matrix equilibration seeks to balance row and column magnitudes, as reflected by lower row and column CVs.

Table 1: Row and column CVs of Muon's orthogonalized updates after 5 NS iterations.

Parameter matrix	Shape	Row CV	Column CV
q_proj	1024×1024	**0.194**	0.027
k_proj	1024×1024	**0.180**	0.024
v_proj	1024×1024	**0.195**	0.041
o_proj	1024×1024	0.014	**0.091**
gate_proj	2736×1024	**0.161**	0.017
up_proj	2736×1024	**0.159**	0.016
down_proj	1024×2736	0.015	**0.181**

Table 2: Row and column CVs of unorthogonalized momentum matrices for the remaining 2D parameters.

Parameter matrix	Shape	Row CV	Column CV
embed_tokens	32000×1024	**1.225**	0.185
lm_head	32000×1024	**1.384**	0.493

Using these statistics, we first examine Muon's orthogonalized updates for 2D weights in hidden layers. These update matrices can exhibit imbalance predominantly in either row or column magnitudes. For example, some updates may have relatively balanced row magnitudes but imbalanced column magnitudes, while others may have relatively balanced column magnitudes but imbalanced row magnitudes. As a concrete example, we examine Transformer layer 4 of Llama-350M at training step 1,144. For each 2D weight in this layer, we record the saved post-NS update produced after 5 NS iterations, corresponding to $\boldsymbol{U}_t$ in (4). Table 1 shows the row and column CVs of these updates. The orthogonalized update matrices associated with q_proj, k_proj, v_proj, gate_proj, and up_proj have relatively balanced column magnitudes but imbalanced row magnitudes, whereas those associated with o_proj and down_proj exhibit the opposite pattern. Thus, after 5 NS iterations, either the row magnitudes or the column magnitudes can be nearly balanced, while the other remains imbalanced, even within the same Transformer layer. Additional imbalance patterns across different models and training steps are reported in Appendix A.

To examine how imbalance in row and column magnitudes changes during orthogonalization, we replay the NS iteration in (5) using the saved pre-NS matrices and track both CVs from $k = 0$ to $k = 20$. Figure 1 shows the resulting trajectories. For the square matrices, the row and column CVs decrease at different rates during the first few iterations. For v_proj, for example, the column CV initially exceeds the row CV, but this ordering reverses after 5 iterations: (γ_r, γ_c) changes from $(0.673, 0.773)$ at $k = 0$ to $(0.195, 0.041)$ at $k = 5$. By $k = 20$, both row and column CVs are close to zero for the square matrices. For the rectangular matrices, one CV approaches zero as the NS iterations proceed, while the other remains large. These observations can be understood from the constraints imposed by orthogonalization. Let $\boldsymbol{X} \in \mathbb{R}^{m \times n}$ be full-rank, and let $\boldsymbol{Y} = \mathit{Orth}(\boldsymbol{X})$ denote its exact orthogonalization. When $m > n$, $\boldsymbol{Y}^\top \boldsymbol{Y} = \boldsymbol{I}_n$, so all column magnitudes are equal and $\gamma_c = 0$, while the row magnitudes remain unconstrained. When $m < n$, $\boldsymbol{Y}\boldsymbol{Y}^\top = \boldsymbol{I}_m$, so all row magnitudes are equal and $\gamma_r = 0$, while the column magnitudes remain unconstrained. Thus, for rectangular matrices, exact orthogonalization constrains only one side, allowing one-sided imbalance to persist. For square matrices, exact orthogonalization gives $\gamma_r = \gamma_c = 0$. The residual imbalance observed after 5 NS iterations therefore reflects finite-iteration approximation error, with the row and column CVs decreasing at different rates.

For the remaining 2D parameters, we separately accumulate unorthogonalized momentum matrices for embed_tokens and lm_head in Llama-350M. Table 2 reports their row and column CVs at training step 1,144. Each matrix exhibits imbalance in both row and column magnitudes. Although their row CVs are substantially larger, their column CVs also remain non-negligible. Since these momentum matrices are not orthogonalized, no orthogonality condition enforces equal row or column magnitudes.

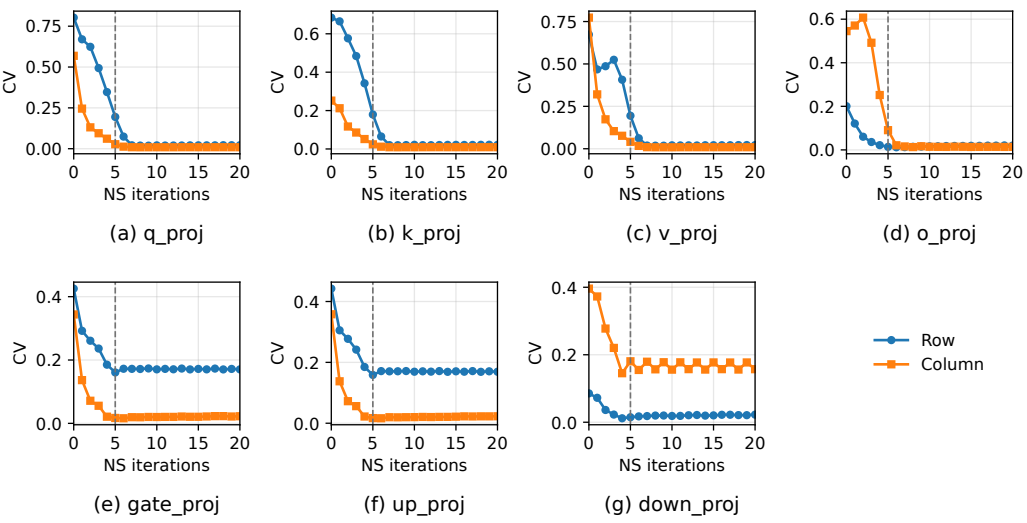


Figure 1: Row and column CV trajectories during NS iterations.

NorMuon (Li et al., 2026) applies row-wise normalization to address the row imbalance observed in Muon's orthogonalized updates. Our observation shows that Muon's orthogonalized updates can exhibit imbalance predominantly in either row or column magnitudes, whereas unorthogonalized momentum matrices for the remaining 2D parameters exhibit imbalance in both row and column magnitudes. Row-wise normalization alone therefore cannot accommodate these different imbalance patterns. To address this limitation, we propose MeqMuon, which balances both row and column magnitudes through normalization tailored to these patterns.

3.2 MeqMuon

MeqMuon maintains one momentum buffer for each parameter and applies normalization to the updates according to parameter type. It uses adaptive one-sided normalization for 2D weights in hidden layers, two-sided normalization for the remaining 2D parameters, and global-RMS normalization for each 1D parameter.

For a matrix $\boldsymbol{X} \in \mathbb{R}^{m\times n}$, we define RMS-based row-wise and column-wise normalization using the RMS vectors $\boldsymbol{r}$ and $\boldsymbol{c}$ in (7):

$$\mathcal{N}_r(\boldsymbol{X}) = \mathrm{diag}(\boldsymbol{r}^{-1})\boldsymbol{X}, \qquad \mathcal{N}_c(\boldsymbol{X}) = \boldsymbol{X}\,\mathrm{diag}(\boldsymbol{c}^{-1}), \tag{9}$$

where $\mathrm{diag}(\boldsymbol{r}^{-1})$ denotes a diagonal matrix with the diagonal elements being $\boldsymbol{r}^{-1}$. These operators rescale each nonzero row or column to unit RMS, respectively.

For each parameter, we initialize the momentum as $\boldsymbol{B}_{-1} = \boldsymbol{0}$ and update it as follows:

$$\boldsymbol{B}_t = \mu\boldsymbol{B}_{t-1} + \boldsymbol{G}_t, \tag{10}$$

where $\boldsymbol{G}_t$ is the corresponding gradient and μ is the momentum coefficient.

For 2D weights in hidden layers, we follow Muon to compute the orthogonalized update $\boldsymbol{U}_t = \mathit{Orth}(\boldsymbol{B}_t)$. We then select row-wise or column-wise normalization by comparing the row and column CVs of $\boldsymbol{U}_t$:

$$\widetilde{\boldsymbol{U}}_t = \begin{cases} \mathcal{N}_r(\boldsymbol{U}_t), & if\ \ \gamma_r(\boldsymbol{U}_t) \geq \gamma_c(\boldsymbol{U}_t), \\ \mathcal{N}_c(\boldsymbol{U}_t), & if\ \ \gamma_r(\boldsymbol{U}_t) < \gamma_c(\boldsymbol{U}_t). \end{cases} \tag{11}$$

The normalization direction is selected independently for each matrix at every optimization step.

For the remaining 2D parameters, such as token embedding matrices and LM heads, we replace Muon's AdamW updates with two-sided normalization of the unorthogonalized momentum:

$$\widetilde{\boldsymbol{U}}_t = \mathcal{N}_c(\mathcal{N}_r(\boldsymbol{B}_t))\,. \tag{12}$$

Table 3: Row and column CVs before and after normalization.

	Before		After	
Parameter matrix	Row CV	Column CV	Row CV	Column CV
`q_proj`	0.194	0.027	< 0.001	0.027
`k_proj`	0.180	0.024	< 0.001	0.023
`v_proj`	0.195	0.041	< 0.001	0.041
`o_proj`	0.014	0.091	0.014	< 0.001
`gate_proj`	0.161	0.017	< 0.001	0.018
`up_proj`	0.159	0.016	< 0.001	0.018
`down_proj`	0.015	0.181	0.015	< 0.001
`embed_tokens`	1.225	0.185	0.008	< 0.001
`lm_head`	1.384	0.493	0.114	< 0.001

The two-sided normalization consists of both row-wise and column-wise normalization.

Table 3 compares the row and column CVs of the matrices in Table 1 and Table 2 before and after the corresponding normalization. For Muon's orthogonalized updates, row-wise or column-wise normalization reduces the larger CV to nearly zero while keeping the other CV small. For the remaining 2D parameters, two-sided normalization substantially reduces both row and column CVs. MeqMuon thus balances both row and column magnitudes through normalization tailored to different imbalance patterns.

For 1D parameters, such as RMSNorm weights (Zhang & Sennrich, 2019) and bias vectors, we normalize the momentum $\boldsymbol{B}_t \in \mathbb{R}^d$ by its global RMS:

$$\mathcal{N}(\boldsymbol{B}_t) = \frac{\boldsymbol{B}_t}{\|\boldsymbol{B}_t\|_2/\sqrt{d}}. \tag{13}$$

The parameter update rule in MeqMuon is given by

$$\boldsymbol{W}_{t+1} = (1 - \eta_t\lambda)\boldsymbol{W}_t - \rho\eta_t\widetilde{\boldsymbol{U}}_t, \tag{14}$$

where the first term applies decoupled weight decay and the second term scales the normalized update by $\rho\eta_t$. Either row-wise or column-wise normalization makes the update RMS independent of matrix shape. Thus, $\sqrt{\max(m, n)}$ in (6) is no longer needed. Algorithm 1 summarizes the update rules of MeqMuon.

4 Experiments

In this section, we evaluate the performance of MeqMuon and other baselines for LLM pretraining. All the experiments are conducted on NVIDIA RTX A6000 GPUs. All the methods are implemented on PyTorch 2.6.0 with CUDA 12.4 and Transformers 4.57.6, using the DistributedDataParallel (DDP) framework. We use `torch.compile`, BF16 automatic mixed-precision training with FP32 master weights, and scaled dot-product attention with the Flash backend enabled.

We evaluate Llama (Touvron et al., 2023), SmolLM2 (Allal et al., 2025), and Qwen2 (Yang et al., 2024a) models. Llama uses the T5-base tokenizer (Raffel et al., 2020), while the other models use their native tokenizers. Llama uses separate weights for the token embeddings and the LM head, whereas SmolLM2 and Qwen2 share these weights. Details of the model architectures are provided in Appendix B. All models are trained from random initialization. We use English C4 (Raffel et al., 2020) for pretraining, with text packed into fixed-length sequences. Following the Chinchilla compute-optimal training rule (Hoffmann et al., 2022), we set the training token budget to 20 times the number of model parameters. We report perplexity (PPL) on the validation set, defined as the exponential of the average per-token cross-entropy loss.

We compare MeqMuon with AdamW (Loshchilov & Hutter, 2019), SCALE (Glentis et al., 2026), Muon (Liu et al., 2025), and NorMuon (Li et al., 2026). We use PyTorch's built-in AdamW. SCALE

Algorithm 1: MeqMuon

Input: Initial parameters $\{\boldsymbol{W}_0\}$, total training steps T, learning rates $\{\eta_t\}_{t=0}^{T-1}$, coefficients μ, ρ, λ;
Output: Parameters $\{\boldsymbol{W}_T\}$;
Initialize $\boldsymbol{B}_{-1} \leftarrow \mathbf{0}$ for every parameter;
for $t = 0, \ldots, T-1$ **do**
  foreach *trainable parameter* $\boldsymbol{W}_t$ **do**
    Obtain its gradient $\boldsymbol{G}_t$;
    $\boldsymbol{B}_t \leftarrow \mu \boldsymbol{B}_{t-1} + \boldsymbol{G}_t$;
    if $\boldsymbol{W}_t$ *is a 2D weight in a hidden layer* **then**
      $\boldsymbol{U}_t \leftarrow \mathit{Orth}(\boldsymbol{B}_t)$;
      if $\gamma_r(\boldsymbol{U}_t) \geq \gamma_c(\boldsymbol{U}_t)$ **then**
        $\widetilde{\boldsymbol{U}}_t \leftarrow \mathcal{N}_r(\boldsymbol{U}_t)$;
      else
        $\widetilde{\boldsymbol{U}}_t \leftarrow \mathcal{N}_c(\boldsymbol{U}_t)$;
    else if $\boldsymbol{W}_t$ *is one of the remaining 2D parameters* **then**
      $\widetilde{\boldsymbol{U}}_t \leftarrow \mathcal{N}_c(\mathcal{N}_r(\boldsymbol{B}_t))$;
    else
      $\widetilde{\boldsymbol{U}}_t \leftarrow \mathcal{N}(\boldsymbol{B}_t)$;
    $\boldsymbol{W}_{t+1} \leftarrow (1 - \eta_t \lambda)\boldsymbol{W}_t - \rho \eta_t \widetilde{\boldsymbol{U}}_t$;

Table 4: Validation PPL for pretraining different models (↓). "–" indicates that SCALE is not evaluated, since it updates the token embedding and the LM head differently and is thus incompatible with the tied embedding and LM head weights in SmolLM2 and Qwen2.

	Llama			SmolLM2		Qwen2
Model scale	60M	130M	350M	135M	360M	0.5B
Token budget	1.2B	2.7B	7.4B	2.7B	7.2B	9.9B
AdamW	37.40	24.46	17.02	24.88	18.46	19.65
SCALE	58.38	33.53	19.87	–	–	–
Muon	29.88	21.64	16.04	22.81	17.28	18.79
NorMuon	29.72	21.57	15.98	22.83	17.20	18.76
MeqMuon	**29.53**	**21.38**	**15.94**	**22.80**	**17.16**	**18.63**

and NorMuon are based on their official implementations. Muon follows Moonlight's implementation (Liu et al., 2025) as shown in (6). We set the sequence length to 1,024 and the global batch size to 512. All experiments use decoupled weight decay with a coefficient of 0.1. Gradient clipping and activation checkpointing are disabled. We use linear learning rate warmup for the first 5% of updates, followed by cosine learning rate decay. AdamW and the AdamW updates for the remaining parameters in Muon and NorMuon use $(\beta_1, \beta_2) = (0.9, 0.95)$. For 2D weights in hidden layers, Muon, NorMuon, and MeqMuon use Nesterov momentum with coefficient $\mu = 0.95$. They use 5 NS iterations with coefficients $(a, b, c) = (3.4445, -4.7750, 2.0315)$ and RMS alignment coefficient $\rho = 0.2$. SCALE is evaluated only on Llama models, because it updates the token embedding and the LM head in different ways, which is incompatible with the shared embedding and LM head weights in SmolLM2 and Qwen2.

Convergence performance. Table 4 reports the final validation PPL of different optimizers across the evaluated models, and Figure 2 compares the validation PPL curves of Muon, NorMuon, and MeqMuon during pretraining. Muon, NorMuon, and MeqMuon achieve substantially lower final PPL than AdamW and SCALE, while MeqMuon obtains the best final PPL for every reported model

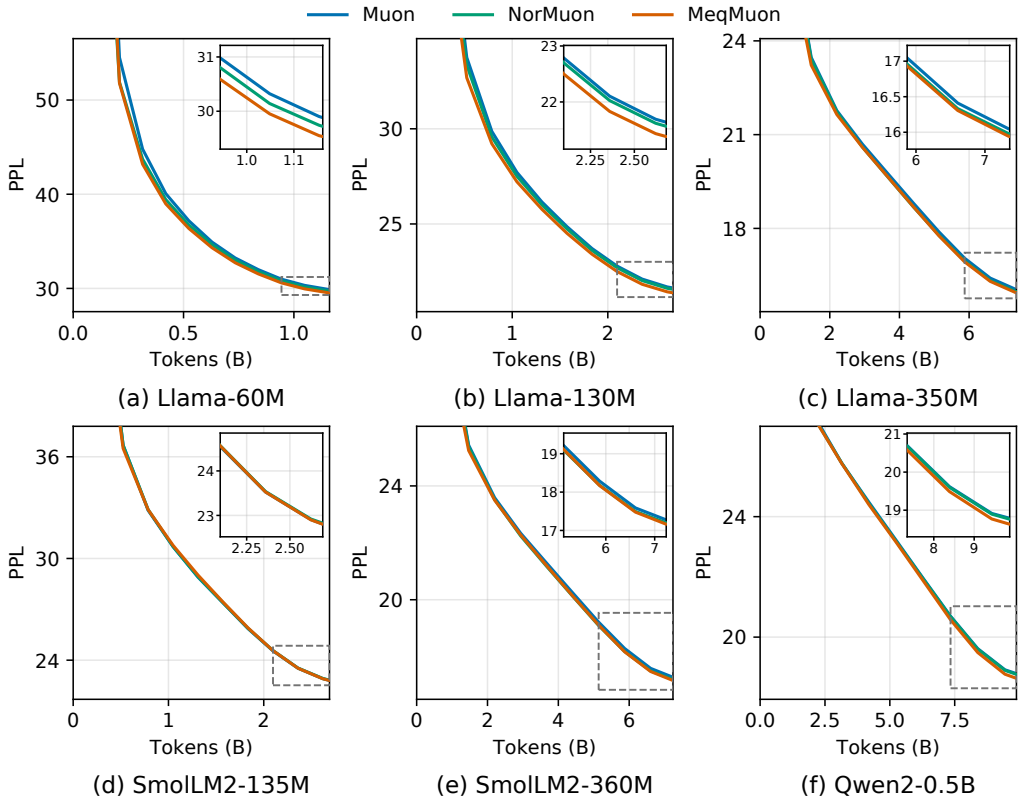


Figure 2: Validation PPL curves during pretraining.

Table 5: Optimizer-state memory of different optimizers (MiB, ↓).

	Llama			SmolLM2		Qwen2
	60M	130M	350M	135M	360M	0.5B
Muon	346.57	699.15	1653.88	621.27	1560.48	2404.17
NorMuon	346.73	699.51	1654.85	621.86	1561.53	2405.33
MeqMuon	**221.53**	**511.57**	**1403.69**	**513.13**	**1380.24**	**1884.59**

scale. Its validation PPL curves also generally remain below those of Muon and NorMuon during training, demonstrating consistently improved convergence.

Memory overhead. Let N_h denote the total number of elements across all 2D weights in hidden layers, and let N_a denote the number of elements in the remaining parameters. Let R_h denote the total number of rows across all 2D weights in hidden layers. Muon stores N_h momentum values and $2N_a$ AdamW first- and second-moment values. Compared with the memory overhead of Muon, NorMuon additionally stores R_h row-wise second-moment values. MeqMuon stores only $N_h + N_a$ momentum values. Table 5 shows the optimizer-state memory of different optimizers, which confirms the predicted savings of MeqMuon over Muon and NorMuon. For example, on Qwen2-0.5B, MeqMuon reduces the optimizer-state memory by 21.6% compared with Muon and NorMuon.

Normalization direction. For the 2D weights in hidden layers, Table 6 shows MeqMuon's normalization-direction selection frequencies by module type. The frequencies are aggregated over all training steps and layers for Llama-350M and SmolLM2-360M. The selections exhibit a strong relationship with matrix geometry: the tall `gate_proj` and `up_proj` matrices always select row-

Table 6: Row-wise and column-wise normalization selections during pretraining.

	Llama-350M			SmolLM2-360M		
Module	Shape	Row (%)	Col. (%)	Shape	Row (%)	Col. (%)
`q_proj`	1024×1024	**89.87**	10.13	960×960	**94.02**	5.98
`k_proj`	1024×1024	**99.21**	0.79	320×960	3.97	**96.03**
`v_proj`	1024×1024	**99.72**	0.28	320×960	1.80	**98.20**
`o_proj`	1024×1024	10.06	**89.94**	960×960	18.85	**81.15**
`gate_proj`	2736×1024	**100.00**	0.00	2560×960	**100.00**	0.00
`up_proj`	2736×1024	**100.00**	0.00	2560×960	**100.00**	0.00
`down_proj`	1024×2736	0.00	**100.00**	960×2560	0.00	**100.00**

Table 7: Component-wise ablation of validation PPL and optimizer-state memory on Llama models.

	Llama-60M		Llama-130M	
Variant	PPL (↓)	Memory (MiB, ↓)	PPL (↓)	Memory (MiB, ↓)
Muon	29.88	346.57	21.64	699.15
Only 2D weights in hidden layers	29.74	346.57	21.53	699.15
Only remaining 2D parameters	29.81	221.57	21.52	511.65
Only 1D parameters	29.81	346.53	21.69	699.07
MeqMuon	**29.53**	**221.53**	**21.38**	**511.57**

wise normalization, whereas the wide `down_proj` matrices always select column-wise normalization, and the wide `k_proj` and `v_proj` matrices in SmolLM2-360M also predominantly select the column-wise normalization. These selections agree with the one-sided imbalance induced by orthogonalizing rectangular matrices, as discussed in Section 3.1. For square matrices, the preference depends on the module and model: `q_proj` predominantly selects row-wise normalization, while `o_proj` predominantly selects column-wise normalization in both models.

Ablation study. We conduct component-wise ablations to separate the contributions of MeqMuon across three parameter types defined in Section 3.2. All variants start from Muon. For 2D weights in hidden layers, the corresponding variant applies adaptive one-sided normalization to their orthogonalized updates. For the remaining 2D parameters, the corresponding variant replaces AdamW with two-sided normalization of their unorthogonalized momentum. For 1D parameters, the corresponding variant replaces AdamW by normalizing each momentum vector by its global RMS. Each single-component variant modifies only one parameter group, while MeqMuon modifies all three. Table 7 shows the validation PPL and optimizer-state memory. The variants that modify only 2D weights in hidden layers or only the remaining 2D parameters both achieve lower validation PPL than Muon at both model scales, while MeqMuon achieves the lowest (best) PPL. The optimizer-state memory savings come from replacing AdamW updates with normalized momentum for the remaining 2D parameters and 1D parameters.

5 Conclusion

In this paper, we propose an improved Muon optimizer, called matrix-equilibrating Muon (MeqMuon), for LLM pretraining. MeqMuon balances both row and column magnitudes through normalization that can be automatically tailored to different imbalance patterns without manual intervention. It eliminates second-moment storage by replacing AdamW updates with normalized momentum for the remaining parameters. Empirical results demonstrate that MeqMuon achieves better convergence performance than AdamW, Muon, and other baselines in LLM pretraining.

AI USE STATEMENT

In this work, we used generative AI tools to implement methods, including the proposed optimizer and the training, evaluation, and logging code, and to clean and reformat datasets, including tokenizing and packing the pretraining corpus. We have not used generative AI tools to help develop theoretical models or conceptual frameworks, formulate mathematical claims, provide critical ingredients for proving mathematical claims, assist in the writing of proofs, propose or refine hypotheses, design or provide feedback on research methodology or experiments, assist with translation, or interpret results. Generating synthetic data sets and supporting qualitative and thematic data analysis are not applicable to this work. Additionally, we used generative AI tools to create or edit software code, create or modify scientific figures and tables, and edit the paper to improve readability. We have reviewed all AI-assisted work: the authors reviewed and tested all AI-assisted code for correctness, verified all reported numbers in tables and figures against the original training logs, and checked all AI-edited text for technical accuracy. We take responsibility for the final content of this work, including text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

Loubna Ben Allal, Anton Lozhkov, Elie Bakouch, Gabriel Martín Blázquez, Guilherme Penedo, Lewis Tunstall, Andrés Marafioti, Hynek Kydlíček, Agustín Piqueres Lajarín, Vaibhav Srivastav, et al. Smollm2: When smol goes big–data-centric training of a small language model. *arXiv preprint arXiv:2502.02737*, 2025.

Salvatore Candido, Thomas Hayes, Alexander Derry, Roshan Rao, Zeming Lin, Robert Verkuil, Bryan Z Wu, Jin Sub Lee, Elise S Bruguera, Jehan A Keval, et al. Language modeling materializes a world model of protein biology. *bioRxiv*, 2026.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Xiangning Chen, Chen Liang, Da Huang, Esteban Real, Kaiyuan Wang, Hieu Pham, Xuanyi Dong, Thang Luong, Cho-Jui Hsieh, Yifeng Lu, and Quoc V Le. Symbolic discovery of optimization algorithms. In *Advances in Neural Information Processing Systems*, pp. 49205–49233, 2023.

Ashok Cutkosky and Harsh Mehta. Momentum improves normalized SGD. In *Proceedings of the International Conference on Machine Learning*, 2020.

DeepSeek-AI. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 2026.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 8-bit optimizers via block-wise quantization. In *Proceedings of the International Conference on Learning Representations*, 2022.

Athanasios Glentis, Jiaxiang Li, Andi Han, and Mingyi Hong. Memory-efficient LLM pretraining via minimalist optimizer design. In *Proceedings of the International Conference on Machine Learning*, 2026.

Jordan Hoffmann, Sebastian Borgeaud, Arthur Mensch, Elena Buchatskaya, Trevor Cai, Eliza Rutherford, Diego de Las Casas, Lisa Anne Hendricks, Johannes Welbl, Aidan Clark, et al. Training compute-optimal large language models. *arXiv preprint arXiv:2203.15556*, 2022.

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

Kimi Team. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025.

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 *Proceedings of the International Conference on Learning Representations*, 2015.

Kfir Y Levy. The power of normalization: Faster evasion of saddle points. *arXiv preprint arXiv:1611.04831*, 2016.

Zichong Li, Liming Liu, Chen Liang, Weizhu Chen, and Tuo Zhao. NorMuon: Making Muon more efficient and scalable. In *Proceedings of the International Conference on Machine Learning*, 2026.

Zeming Lin, Halil Akin, Roshan Rao, Brian Hie, Zhongkai Zhu, Wenting Lu, Nikita Smetanin, Robert Verkuil, Ori Kabeli, Yaniv Shmueli, Allan dos Santos Costa, Maryam Fazel-Zarandi, Tom Sercu, Salvatore Candido, and Alexander Rives. Evolutionary-scale prediction of atomic-level protein structure with a language model. *Science*, 379(6637):1123–1130, 2023.

Aixin Liu, Bei Feng, Bing Xue, Bingxuan Wang, Bochao Wu, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chenyu Zhang, Chong Ruan, et al. Deepseek-v3 technical report. *arXiv preprint arXiv:2412.19437*, 2024.

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

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *Proceedings of the International Conference on Learning Representations*, 2019.

Ryan Murray, Brian Swenson, and Soummya Kar. Revisiting normalized gradient descent: Fast evasion of saddle points. *IEEE Transactions on Automatic Control*, 64(11):4818–4824, 2019.

Matteo Pagliardini, Pierre Ablin, and David Grangier. The ademamix optimizer: Better, faster, older. In *Proceedings of the International Conference on Learning Representations*, 2025.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21(140):1–67, 2020.

Zhihong Shao, Yuxiang Luo, Chengda Lu, ZZ Ren, Jiewen Hu, Tian Ye, Zhibin Gou, Shirong Ma, and Xiaokang Zhang. Deepseekmath-v2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

Noam Shazeer and Mitchell Stern. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the International Conference on Machine Learning*, 2018.

Tao Sun, Xinwang Liu, and Kun Yuan. Revisiting gradient normalization and clipping for nonconvex sgd under heavy-tailed noise: Necessity, sufficiency, and acceleration. *Journal of Machine Learning Research*, 26(237):1–42, 2025.

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

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, Guanting Dong, Haoran Wei, Huan Lin, Jialong Tang, Jialin Wang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Ma, Jianxin Yang, Jin Xu, Jingren Zhou, Jinze Bai, Jinzheng He, Junyang Lin, Kai Dang, Keming Lu, Keqin Chen, Kexin Yang, Mei Li, Mingfeng Xue, Na Ni, Pei Zhang, Peng Wang, Ru Peng, Rui Men, Ruize Gao, Runji Lin, Shijie Wang, Shuai Bai, Sinan Tan, Tianhang Zhu, Tianhao Li, Tianyu Liu, Wenbin Ge, Xiaodong Deng, Xiaohuan Zhou, Xingzhang Ren, Xinyu Zhang, Xipin Wei, Xuancheng Ren, Xuejing Liu, Yang Fan, Yang Yao, Yichang Zhang, Yu Wan, Yunfei Chu, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, Zhifang Guo, and Zhihao Fan. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*, 2024a.

Yi-Rui Yang, Chang-Wei Shi, and Wu-Jun Li. On the effect of batch size in byzantine-robust distributed learning. In *Proceedings of the International Conference on Learning Representations*, 2024b.

Biao Zhang and Rico Sennrich. Root mean square layer normalization. In *Advances in Neural Information Processing Systems*, 2019.

Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. Why gradient clipping accelerates training: A theoretical justification for adaptivity. In *Proceedings of the International Conference on Learning Representations*, 2020.

Yushun Zhang, Congliang Chen, Ziniu Li, Tian Ding, Chenwei Wu, Diederik (Durk) Kingma, Yinyu Ye, Zhi-Quan Luo, and Ruoyu Sun. Adam-mini: Use fewer learning rates to gain more. In *Proceedings of the International Conference on Learning Representations*, 2025.

Jiawei Zhao, Zhenyu Zhang, Beidi Chen, Zhangyang Wang, Anima Anandkumar, and Yuandong Tian. GaLore: Memory-efficient LLM training by gradient low-rank projection. In *Proceedings of the International Conference on Machine Learning*, 2024a.

Shen-Yi Zhao, Chang-Wei Shi, Yin-Peng Xie, and Wu-Jun Li. Stochastic normalized gradient descent with momentum for large-batch training. *Science China Information Sciences*, 67(11): 212101, 2024b.

Hanqing Zhu, Zhenyu Zhang, Wenyan Cong, Xi Liu, Sem Park, Vikas Chandra, Bo Long, David Z. Pan, Zhangyang Wang, and Jinwon Lee. Apollo: Sgd-like memory, adamw-level performance. In *Proceedings of Machine Learning and Systems*, 2025.

Table 8: Row and column CVs for Llama-350M after 5 NS iterations.

Layer	Matrix	Shape	Step 381		Step 1144		Step 1907	
			γ_r	γ_c	γ_r	γ_c	γ_r	γ_c
0	q_proj	1024×1024	**0.033**	0.012	**0.084**	0.022	**0.066**	0.024
	k_proj	1024×1024	**0.026**	0.012	**0.079**	0.022	**0.073**	0.027
	v_proj	1024×1024	**0.078**	0.019	**0.084**	0.027	**0.082**	0.024
	o_proj	1024×1024	0.015	**0.063**	0.013	**0.053**	0.013	**0.048**
	gate_proj	2736×1024	**0.603**	0.020	**0.534**	0.026	**0.528**	0.023
	up_proj	2736×1024	**0.616**	0.020	**0.538**	0.026	**0.531**	0.023
	down_proj	1024×2736	0.009	**0.558**	0.013	**0.505**	0.014	**0.505**
12	q_proj	1024×1024	**0.492**	0.023	**0.333**	0.021	**0.287**	0.018
	k_proj	1024×1024	**0.536**	0.025	**0.365**	0.022	**0.326**	0.020
	v_proj	1024×1024	**0.368**	0.023	**0.238**	0.021	**0.163**	0.017
	o_proj	1024×1024	0.017	**0.231**	0.014	**0.151**	0.014	**0.115**
	gate_proj	2736×1024	**0.248**	0.013	**0.155**	0.015	**0.132**	0.016
	up_proj	2736×1024	**0.227**	0.014	**0.147**	0.014	**0.129**	0.016
	down_proj	1024×2736	0.014	**0.250**	0.014	**0.150**	0.015	**0.119**
23	q_proj	1024×1024	**0.218**	0.014	**0.050**	0.011	**0.038**	0.011
	k_proj	1024×1024	**0.234**	0.016	**0.056**	0.011	**0.043**	0.011
	v_proj	1024×1024	**0.207**	0.019	**0.055**	0.012	**0.043**	0.011
	o_proj	1024×1024	0.025	**0.187**	0.014	**0.060**	0.013	**0.043**
	gate_proj	2736×1024	**0.315**	0.013	**0.214**	0.015	**0.198**	0.016
	up_proj	2736×1024	**0.317**	0.013	**0.230**	0.014	**0.214**	0.015
	down_proj	1024×2736	0.020	**0.303**	0.018	**0.205**	0.017	**0.187**

A Additional imbalance patterns

Table 8, Table 9, and Table 10 report the row and column CVs, γ_r and γ_c defined in (8), of Muon's orthogonalized updates $\boldsymbol{U}_t$ for Llama-350M, Llama-60M, and SmolLM2-135M, respectively. For each model, we select three different training steps and three Transformer layers. The tables report all seven update matrices for 2D weights in hidden layers after 5 NS iterations. For each training step, the larger of γ_r and γ_c is shown in bold.

B Model Architectures

Table 11 gives the architecture of each model. All models use pre-normalization with RMSNorm, SwiGLU feed-forward layers, and rotary position embeddings. The Llama models use multi-head attention, whereas SmolLM2 and Qwen2 use grouped-query attention. Qwen2 additionally uses biases in the query, key, and value projections. The Params column gives the number of unique parameters in millions, so tied token-embedding and LM-head weights are counted once. Q/KV gives the numbers of query and key/value heads, and Tied indicates whether the token embedding and the LM head share weights.

Table 9: Row and column CVs for Llama-60M after 5 NS iterations.

			Step 443		Step 1329		Step 2215	
Layer	Matrix	Shape	γ_r	γ_c	γ_r	γ_c	γ_r	γ_c
0	q_proj	512×512	**0.034**	0.019	**0.043**	0.023	**0.040**	0.021
	k_proj	512×512	**0.088**	0.031	**0.076**	0.031	**0.066**	0.026
	v_proj	512×512	**0.164**	0.081	**0.101**	0.041	**0.066**	0.025
	o_proj	512×512	0.034	**0.144**	0.020	**0.087**	0.017	**0.050**
	gate_proj	1376×512	**0.654**	0.022	**0.528**	0.017	**0.503**	0.017
	up_proj	1376×512	**0.665**	0.026	**0.522**	0.013	**0.497**	0.016
	down_proj	512×1376	0.012	**0.603**	0.016	**0.523**	0.016	**0.506**
4	q_proj	512×512	**0.167**	0.026	**0.027**	0.015	**0.023**	0.014
	k_proj	512×512	**0.206**	0.026	**0.029**	0.015	**0.027**	0.015
	v_proj	512×512	**0.145**	0.029	**0.032**	0.018	**0.019**	0.016
	o_proj	512×512	0.017	**0.041**	0.014	**0.017**	0.014	**0.016**
	gate_proj	1376×512	**0.128**	0.016	**0.075**	0.017	**0.067**	0.017
	up_proj	1376×512	**0.127**	0.016	**0.086**	0.018	**0.082**	0.016
	down_proj	512×1376	0.017	**0.128**	0.016	**0.084**	0.016	**0.077**
7	q_proj	512×512	**0.059**	0.015	**0.026**	0.015	**0.026**	0.014
	k_proj	512×512	**0.073**	0.017	**0.030**	0.014	**0.029**	0.014
	v_proj	512×512	**0.052**	0.021	**0.018**	0.015	**0.019**	0.015
	o_proj	512×512	0.019	**0.038**	0.016	**0.019**	0.016	**0.016**
	gate_proj	1376×512	**0.144**	0.018	**0.093**	0.019	**0.092**	0.016
	up_proj	1376×512	**0.167**	0.016	**0.123**	0.018	**0.120**	0.017
	down_proj	512×1376	0.019	**0.136**	0.020	**0.093**	0.021	**0.098**

Table 10: Row and column CVs for SmolLM2-135M after 5 NS iterations.

			Step 381		Step 1144		Step 1907	
Layer	Matrix	Shape	γ_r	γ_c	γ_r	γ_c	γ_r	γ_c
0	q_proj	576×576	**0.018**	0.014	**0.033**	0.015	**0.026**	0.013
	k_proj	192×576	0.023	**0.084**	0.027	**0.089**	0.030	**0.090**
	v_proj	192×576	0.020	**0.082**	0.020	**0.084**	0.020	**0.082**
	o_proj	576×576	0.019	**0.038**	0.016	**0.027**	0.015	**0.026**
	gate_proj	1536×576	**0.421**	0.023	**0.370**	0.021	**0.363**	0.019
	up_proj	1536×576	**0.434**	0.024	**0.370**	0.021	**0.363**	0.019
	down_proj	576×1536	0.012	**0.410**	0.015	**0.353**	0.015	**0.348**
15	q_proj	576×576	**0.033**	0.013	**0.040**	0.013	**0.029**	0.013
	k_proj	192×576	0.038	**0.063**	0.031	**0.061**	0.049	**0.063**
	v_proj	192×576	0.025	**0.062**	0.042	**0.068**	0.039	**0.064**
	o_proj	576×576	0.014	**0.017**	0.014	**0.017**	0.014	**0.014**
	gate_proj	1536×576	**0.248**	0.017	**0.122**	0.018	**0.096**	0.017
	up_proj	1536×576	**0.232**	0.016	**0.125**	0.018	**0.105**	0.018
	down_proj	576×1536	0.016	**0.231**	0.016	**0.116**	0.016	**0.090**
29	q_proj	576×576	**0.017**	0.015	**0.021**	0.015	**0.020**	0.013
	k_proj	192×576	0.028	**0.076**	0.029	**0.077**	0.027	**0.071**
	v_proj	192×576	0.026	**0.083**	0.025	**0.078**	0.023	**0.067**
	o_proj	576×576	0.018	**0.019**	0.015	**0.015**	**0.017**	0.014
	gate_proj	1536×576	**0.321**	0.017	**0.206**	0.017	**0.190**	0.017
	up_proj	1536×576	**0.319**	0.017	**0.210**	0.019	**0.193**	0.018
	down_proj	576×1536	0.019	**0.302**	0.022	**0.198**	0.023	**0.181**

Table 11: Model architectures.

Family	Size	Params (M)	Layers	Hidden	FFN	Q/KV	Head dim	Vocab.	Tied
Llama	60M	58.074	8	512	1376	8/8	64	32,000	No
	130M	134.106	12	768	2048	12/12	64	32,000	No
	350M	367.969	24	1024	2736	16/16	64	32,000	No
SmolLM2	135M	134.515	30	576	1536	9/3	64	49,152	Yes
	360M	361.821	32	960	2560	15/5	64	49,152	Yes
Qwen2	0.5B	494.033	24	896	4864	14/2	64	151,936	Yes