AMUSE: Anytime Muon with Stable Gradient Evaluation

author names withheld

Under Review for the Workshop on High-dimensional Learning Dynamics, 2026

Abstract

Modern deep learning commonly relies on AdamW with learning rate schedules. Schedule-Free optimization and Muon challenge this standard recipe from complementary directions: the former removes the need for explicit schedules, while the latter offers an alternative to AdamW. Despite Muon's strong empirical performance, the mechanism behind its improvement remains only partially understood. We study this question through the river-valley perspective by examining how Muon updates decompose into river and valley directions. We show that Muon's orthogonalization increases the river component of its updates, which helps accelerate progress, but can also leave residual valley components that lead to oscillatory trajectories. Building on this, we propose **Anytime MUon with Stable gradient Evaluation (AMUSE)**, which integrates Muon's rapid river progress with the stabilizing effect of Schedule-Free optimizer. AMUSE initially evaluates gradients near the fast Muon sequence for rapid adaptation, then gradually shifts toward the stable averaged sequence to reduce valley-wall oscillations. As a result, AMUSE requires no learning rate schedules and supports anytime training. Across vision tasks and large language model pretraining, AMUSE consistently improves the performance-iteration Pareto frontier over (Schedule-Free) AdamW and Muon.

1. Introduction

Optimization remains a primary lever for improving training efficiency in deep learning. The standard recipe for Transformer-based language models has long relied on Adam(W) [18, 23] with warmup and prescribed learning rate decay, such as cosine decay [22]. Recent work challenges both parts: Schedule-Free (SF) optimization [9] removes explicit decay schedules by evaluating gradients at interpolated points between fast and averaged iterates, while Muon [16] improves the update geometry by applying momentum to matrix-valued parameters and orthogonalizing the direction.

We study this interaction through the lens of loss landscape geometry. Empirical Hessian spectra in deep networks often consist of a few large outlier eigenvalues and a broad bulk of small eigenvalues, inducing a dominant high-curvature subspace and a bulk low-curvature subspace [11, 28–30, 37, 38]. In the *river-valley landscape* view, the dominant subspace forms steep valley walls, while the bulk subspace forms the relatively flat river along which training progresses [41, 47] (see Figure 1 for illustration). This suggests that effective optimizers should take *(i)* large, consistent steps along the river while *(ii)* suppressing unnecessary motion across the valley.

We find that Muon naturally promotes the first property. Orthogonalization prevents momentum updates from being dominated by a few large singular components, producing larger bulk components than SGD or AdamW. However, this non-selective amplification can also magnify small dominant-direction noise, inducing valley-wall oscillations [14, 33, 51]. SF optimization provides

© .

a complementary stabilizing mechanism: evaluating gradients near an averaged sequence reduces high-curvature components before orthogonalization, stabilizing Muon's trajectory [42].

Motivated by this complementarity, we propose **Anytime MUon with Stable gradient Evaluation (AMUSE)**, which combines Muon's orthogonalized matrix updates with an SF formulation using a time-varying interpolation coefficient. AMUSE evaluates gradients near the fast Muon sequence early in training for rapid adaptation, then gradually shifts toward the averaged sequence for stable gradient evaluation. Because this schedule is independent of the training horizon and does not require learning rate decay, AMUSE naturally supports *anytime training*. Across diverse benchmarks, AMUSE improves the performance-iteration Pareto frontier over (Schedule-Free) AdamW and Muon.

2. A River-Valley Analysis of Muon and Schedule-Free Dynamics

In this section, we analyze Muon and Schedule-Free dynamics in controlled settings where the leading Hessian eigenspaces can be computed. We first decompose directions using the Hessian eigenspaces of the loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$, following Song et al. [41].

Dominant and bulk ratios. Let $\mathcal{S}_k(\theta)$ be the span of the top-k eigenvectors of the Hessian $\nabla^2\mathcal{L}(\theta)$, and $\boldsymbol{P}_k(\theta)$ be the corresponding projection, with $\boldsymbol{P}_k^{\perp}(\theta) = \boldsymbol{I} - \boldsymbol{P}_k(\theta)$. For any nonzero vector $\boldsymbol{v}$, we measure

$$r^{\mathrm{dom}}(\boldsymbol{v};\theta) = \frac{\|\boldsymbol{P}_k(\theta)\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_2}, \qquad r^{\mathrm{bulk}}(\boldsymbol{v};\theta) = \frac{\|\boldsymbol{P}_k^{\perp}(\theta)\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_2}. \tag{1}$$

These quantities measure how strongly $\boldsymbol{v}$ is aligned with the dominant and bulk subspaces, respectively. Following Song et al. [41], we set k to the number of classes throughout.

Before analyzing Muon's update decomposition, we first verify in Muon that *bulk amplification* accelerates training while *dominant amplification* destabilizes it, supporting the river-valley interpretation in our setting and motivating our dominant/bulk update-ratio analysis; see Appendix D.1.

2.1. Orthogonalization Makes Muon Updates More Bulk-Oriented

We first measure how Muon updates align with the dominant and bulk Hessian subspaces. Figure 2 shows that Muon has substantially smaller dominant ratios than SGD or AdamW, indicating more bulk-oriented updates. Moreover, comparing the raw momentum with the post-orthogonalized update shows that orthogonalization itself shifts Muon's update away from dominant directions and toward the bulk. This supports the view that Muon's efficiency comes from accelerating motion along the river directions. However, because this normalization is non-selective, it prevents dominant components from dying out and induces valley-wall oscillations, as in normalized-update dynamics [1]; see Appendix C for an example.

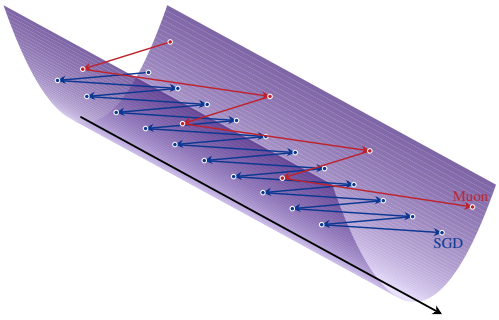


Figure 1: Illustration of a river-valley landscape. SGD oscillates across the valley walls and progresses slowly along the river, while Muon advances faster but remains oscillatory.

2.2. Schedule-Free Stabilizes Muon Trajectories

The previous subsection shows that Muon promotes bulk movement, but its orthogonalized updates can also amplify valley-wall oscillations. Since the averaged trajectory $\boldsymbol{x}_t$ (defined in Eq. (4)) of

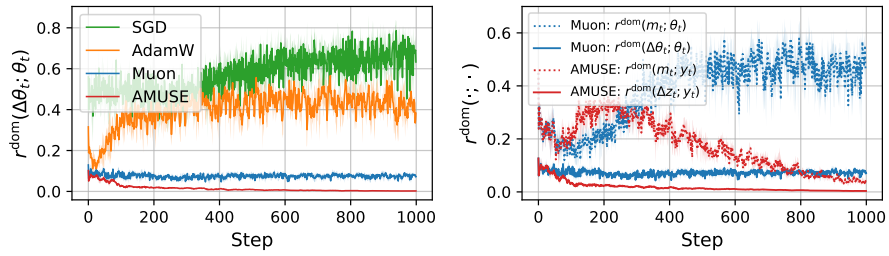


Figure 2: Comparison of dominant component ratios. Evaluated on a 5k MNIST subset using a 3-layer MLP. (**Left**) Muon consistently produces smaller dominant updates than SGD/AdamW, and AMUSE further suppresses the dominant component. (**Right**) Orthogonalization reduces Muon's dominant ratio compared to momentum $\boldsymbol{m}_t$; in contrast, AMUSE maintains low dominant ratios throughout, reflecting more stable gradient dynamics. Averaged over three runs.

SF-AdamW can follow the river with a suitable β [42], SF averaging may mitigate this instability. Motivated by this, we use Muon as the base optimizer in the SF framework, referred to as *SF-Muon*:

$$\begin{aligned} \boldsymbol{Y}_t &= (1-\beta)\boldsymbol{Z}_t + \beta\boldsymbol{X}_t, \\ \boldsymbol{M}_t &= \mu\boldsymbol{M}_{t-1} + \nabla L(\boldsymbol{Y}_t), \\ \boldsymbol{Z}_{t+1} &= \boldsymbol{Z}_t - \eta\mathcal{O}(\boldsymbol{M}_t), \\ \boldsymbol{X}_{t+1} &= (1-c_{t+1})\boldsymbol{X}_t + c_{t+1}\boldsymbol{Z}_{t+1}. \end{aligned} \tag{2}$$

Here, $\boldsymbol{Z}_t$ is the Muon-updated sequence, $\boldsymbol{Y}_t$ is the gradient-evaluation point, and $\boldsymbol{X}_t$ is the averaged sequence used as the final model parameter, with $c_{t+1} = 1/(t+1)$. Using the notation of Eq. (4) in Appendix A, let $\boldsymbol{x}_t, \boldsymbol{y}_t, \boldsymbol{z}_t \in \mathbb{R}^d$ denote the flattened versions of $\boldsymbol{X}_t, \boldsymbol{Y}_t$, and $\boldsymbol{Z}_t$, respectively. Figure 2 shows that SF-Muon makes the update $\Delta\boldsymbol{x}_t := \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$ more bulk-oriented, while the gradient evaluated at $\boldsymbol{y}_t$ remains strongly aligned with the dominant subspace. This suggests that averaging stabilizes the trajectory, but does not fully remove the dominant components in the gradient passed to Muon. This raises a natural question: *can we further reduce the dominant component before orthogonalization by evaluating gradients at a point closer to the river?*

To test this hypothesis, we treat $\boldsymbol{x}_t$ as a proxy for a river point and probe virtual interpolation points $\boldsymbol{y}_t^{(\alpha)} = (1-\alpha)\boldsymbol{z}_t + \alpha\boldsymbol{x}_t$ for $\alpha \in [0,1]$, while keeping the actual training trajectory fixed with a constant β. At each point, we compute the gradient and measure its bulk ratio using Eq. (1).

Figure 3 shows that larger α makes the gradient more bulk-oriented, indicating that evaluation closer to $\boldsymbol{x}_t$ suppresses valley-wall components before Muon orthogonalization. Consistently, AMUSE, the time-varying SF-Muon variant introduced in Section 3, suppresses dominant components both before and after orthogonalization (see Figure 2).

This suggests using a large fixed β to evaluate gradients near $\boldsymbol{x}_t$. However, large fixed β harms early training (Figure 13 in Appendix E.3): $\boldsymbol{x}_t$ has not yet reached the river, whereas $\boldsymbol{z}_t$ is rapidly approaching it, so evaluating near $\boldsymbol{x}_t$ causes severe early instability.

3. AMUSE: Anytime MUon with Stable gradient Evaluation

To address the trade-off between early progress and later stability induced by a fixed β, we introduce **Anytime MUon with Stable gradient Evaluation (AMUSE)**. Instead of fixing the interpolation coefficient throughout training, AMUSE employs a time-varying coefficient β_t. It starts with a small

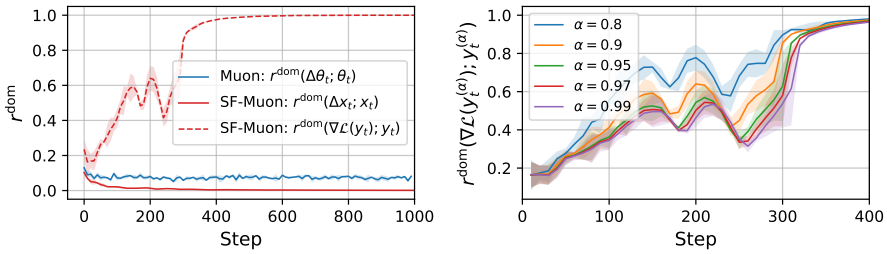


Figure 3: Comparison of dominant component ratios. Settings as in Figure 2. (**Left**) Dominant ratios for Muon (blue) and SF-Muon (red). The solid and dashed red lines denote the dominant components of $\Delta \boldsymbol{x}_t$ and $\nabla \mathcal{L}(\boldsymbol{y}_t)$, respectively. (**Right**) Dominant ratios for gradients at $\boldsymbol{y}_t^{(\alpha)} = (1-\alpha)\boldsymbol{z}_t + \alpha \boldsymbol{x}_t$ across varying α, demonstrating that larger α values correlate with lower dominant components.

β_t to allow fast early adaptation and gradually increases it, shifting the gradient-evaluation point from the fast base trajectory ($\boldsymbol{z}_t$) toward the smoother averaged trajectory ($\boldsymbol{x}_t$). Formally, AMUSE uses the same update rule as Eq. (2), except the gradient is now evaluated at $\boldsymbol{Y}_t = (1-\beta_t)\boldsymbol{Z}_t + \beta_t \boldsymbol{X}_t$.

Let T_0 denote the number of warmup steps. During warmup, we fix $\beta_t = \beta_1$. For $t \geq T_0$, we set:

$$\beta_t = 1 - \left(\frac{T_0 - 1}{t-1}\right)^{\rho} (1-\beta_1), \tag{3}$$

where $\rho \in [0,1]$ controls how quickly β_t increases to 1. When $\rho = 0$, AMUSE reduces to the fixed-β SF-Muon with $\beta_t = \beta_1$; a larger ρ moves the gradient-evaluation point toward $\boldsymbol{x}_t$ more rapidly. The detailed rationale and derivation for Eq. (3) are deferred to Appendix E.5. Notably, *this time-varying β_t is independent of the total number of training steps*, gradually approaching 1 as training progresses. Furthermore, AMUSE does not require any learning rate schedule. See Algorithm 1 for details.

We compare AMUSE with fixed-β SF-Muon on 124M LLaMA, where no fixed β outperforms AMUSE; see Appendix E.3. Furthermore, following Song et al. [42], we use EWA and additional learning rate decay to test whether AMUSE already follows the river. Neither yields meaningful additional improvement, suggesting that AMUSE already stays near the river; see Appendix E.2.

4. Experiments

We evaluate AMUSE across a diverse set of benchmarks covering standard image classification, segmentation, and large language model (LLM) pretraining. As baselines, we use SGD and SF-SGD except for MAE fine-tuning and LLM pretraining. Across all settings, we additionally include Muon as a shared baseline. For SGD, AdamW and Muon, we employ cosine learning rate schedule.

4.1. Image Domain Experiments

Experimental Setup. Following the setup of Defazio et al. [9], we evaluate AMUSE on diverse datasets and architectures: Wide-ResNet-16-8 on CIFAR-10 [19, 50], DenseNet on CIFAR-100 [15], ResNet-3-96 on SVHN [12, 27], and ResNet-50 on ImageNet-1k [36]. We further evaluate two additional settings: U-Net on ISIC 2018 for image segmentation [4, 34], and MAE fine-tuning with a ViT-B/16 on ImageNet [13]. Hyperparameter tuning details are provided in Appendix F.1.

Experimental Results. As illustrated in Figure 4, AMUSE consistently achieves the best performance among the compared optimizers across all evaluated datasets, tasks, and architectures, demonstrating *superior anytime performance*.

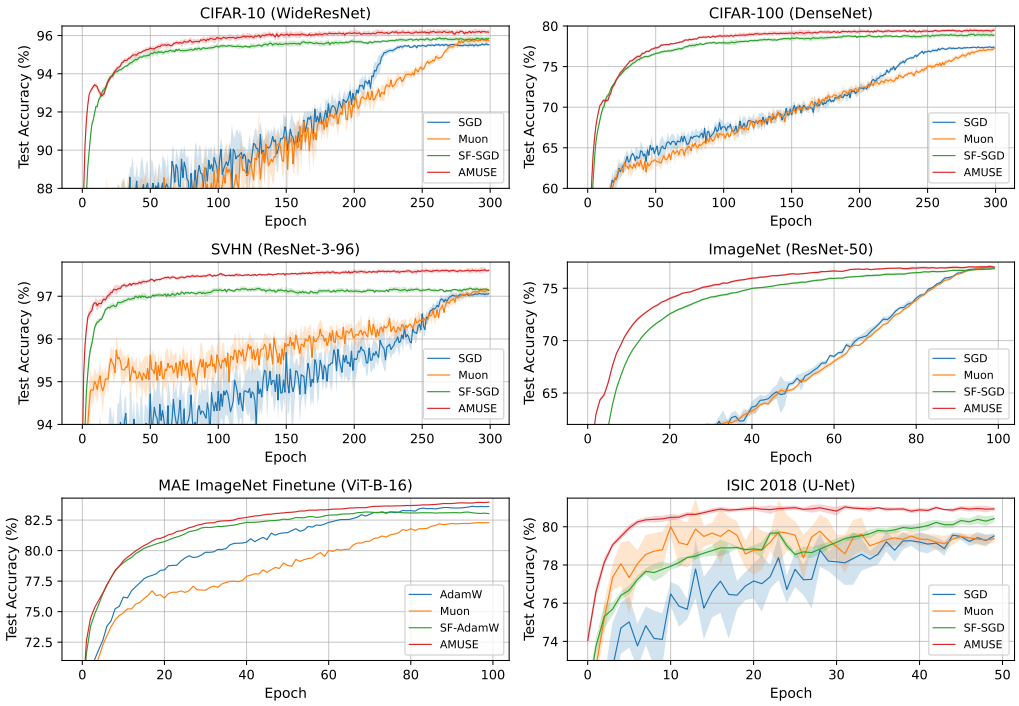


Figure 4: Test accuracy across image domain experiments. Averaged over five random seeds.[1]

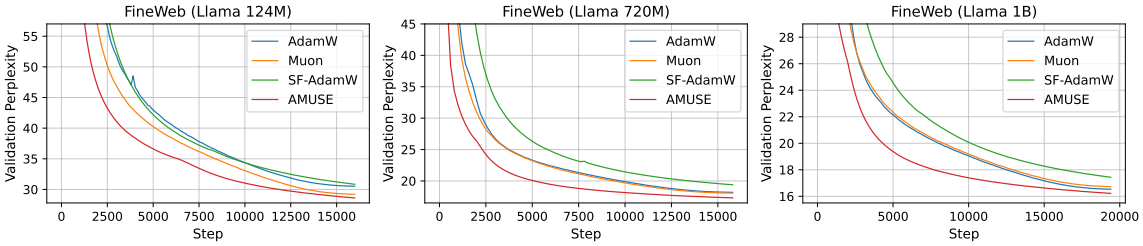


Figure 5: Validation perplexity on FineWeb pretraining across Llama model scales.

4.2. Large Language Model Pretraining

Experimental Setup. We follow the Llama-style Transformer setup of Semenov et al. [39]. We train the 124M and 720M models on FineWeb-100B [31] for 16k iterations, corresponding to approximately 2.1B and 16.3B tokens, respectively. The 1B model is trained for 19.6k iterations, corresponding to approximately 20.6B tokens. Hyperparameter details are provided in Appendix G.2.
Experimental Results. Figure 5 shows that AMUSE achieves better performance than all baselines across diverse scales. AMUSE also maintains its advantage under extended 124M training and against additional recent optimizers, improving the performance-iteration Pareto frontier (Appendices H.3 and H.1). Moreover, Muon with EWA or decay still falls short of AMUSE, suggesting that AMUSE provides more than external averaging or learning rate decay (Appendix H.2).

1. MAE fine-tuning is the only exception, reported with a single random seed due to computational constraints.

References

[1] Sanjeev Arora, Zhiyuan Li, and Abhishek Panigrahi. Understanding gradient descent on the edge of stability in deep learning. In *International Conference on Machine Learning*, pages 948–1024. PMLR, 2022.

[2] Annalisa Belloni, Lorenzo Noci, and Antonio Orvieto. Universal dynamics of warmup stable decay: understanding WSD beyond transformers. In *High-dimensional Learning Dynamics 2025*, 2025. URL https://openreview.net/forum?id=QJfH0lBZ8d.

[3] Jeremy Bernstein and Laker Newhouse. Old optimizer, new norm: An anthology. In *OPT 2024: Optimization for Machine Learning*, 2024. URL https://openreview.net/forum?id=ux18f5nOpD.

[4] Noel Codella, Veronica Rotemberg, Philipp Tschandl, M Emre Celebi, Stephen Dusza, David Gutman, Brian Helba, Aadi Kalloo, Konstantinos Liopyris, Michael Marchetti, et al. Skin lesion analysis toward melanoma detection 2018: A challenge hosted by the international skin imaging collaboration (isic). *arXiv preprint arXiv:1902.03368*, 2019.

[5] Jeremy Cohen, Simran Kaur, Yuanzhi Li, J Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=jh-rTtvkGeM.

[6] Jeremy Cohen, Alex Damian, Ameet Talwalkar, J Zico Kolter, and Jason D. Lee. Understanding optimization in deep learning with central flows. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=sIE2rI3ZPs.

[7] George E Dahl, Frank Schneider, Zachary Nado, Naman Agarwal, Chandramouli Shama Sastry, Philipp Hennig, Sourabh Medapati, Runa Eschenhagen, Priya Kasimbeg, Daniel Suo, et al. Benchmarking neural network training algorithms. *arXiv preprint arXiv:2306.07179*, 2023.

[8] Alex Damian, Eshaan Nichani, and Jason D Lee. Self-stabilization: The implicit bias of gradient descent at the edge of stability. *arXiv preprint arXiv:2209.15594*, 2022.

[9] Aaron Defazio, Xingyu Alice Yang, Ahmed Khaled, Konstantin Mishchenko, Harsh Mehta, and Ashok Cutkosky. The road less scheduled. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=0XeNkkENuI.

[10] Shenyang Deng, Boyao Liao, Zhuoli Ouyang, Tianyu Pang, Minhak Song, and Yaoqing Yang. Suspicious alignment of sgd:a fine-grained step size condition analysis. In *Proceedings of The 37th International Conference on Algorithmic Learning Theory*, volume 313 of *Proceedings of Machine Learning Research*, pages 1–66, 2026.

[11] Behrooz Ghorbani, Shankar Krishnan, and Ying Xiao. An investigation into neural net optimization via hessian eigenvalue density. In *International Conference on Machine Learning*, pages 2232–2241. PMLR, 2019.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[13] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 16000–16009, 2022.

[14] Wei He, Kai Han, Hang Zhou, Hanting Chen, Zhicheng Liu, Xinghao Chen, and Yunhe Wang. Root: Robust orthogonalized optimizer for neural network training. *arXiv preprint arXiv:2511.20626*, 2025.

[15] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[16] 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/.

[17] Changmin Kang, Jihun Yun, Baekrok Shin, Yeseul Cho, and Chulhee Yun. Uniform spectral growth and convergence of muon in lora-style matrix factorization. *arXiv preprint arXiv:2602.06385*, 2026.

[18] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In *3rd International Conference on Learning Representations*, 2015.

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

[20] 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.

[21] Yizhou Liu, Ziming Liu, and Jeff Gore. Focus: First order concentrated updating scheme. *arXiv preprint arXiv:2501.12243*, 2025.

[22] Ilya Loshchilov and Frank Hutter. SGDR: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Skq89Scxx.

[23] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *International Conference on Learning Representations*, 2019.

[24] Kairong Luo, Zhenbo Sun, Haodong Wen, Xinyu Shi, Jiarui Cui, Chenyi Dang, Kaifeng Lyu, and Wenguang Chen. How learning rate decay wastes your best data in curriculum-based LLM pretraining. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=T5wkZJqzkz.

[25] Jianhao Ma, Yu Huang, Yuejie Chi, and Yuxin Chen. Preconditioning benefits of spectral orthogonalization in muon. *arXiv preprint arXiv:2601.13474*, 2026.

[26] Yurii Nesterov. Primal-dual subgradient methods for convex problems. *Mathematical programming*, 120(1):221–259, 2009.

[27] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Baolin Wu, Andrew Y Ng, et al. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 4. Granada, 2011.

[28] Vardan Papyan. The full spectrum of deepnet hessians at scale: Dynamics with sgd training and sample size. *arXiv preprint arXiv:1811.07062*, 2018.

[29] Vardan Papyan. Measurements of three-level hierarchical structure in the outliers in the spectrum of deepnet hessians. In *International Conference on Machine Learning*, pages 5012–5021. PMLR, 2019.

[30] Vardan Papyan. Traces of class/cross-class structure pervade deep learning spectra. *Journal of Machine Learning Research*, 21(252):1–64, 2020. URL http://jmlr.org/papers/v21/20-933.html.

[31] Guilherme Penedo, Hynek Kydlíček, Loubna Ben allal, Anton Lozhkov, Margaret Mitchell, Colin Raffel, Leandro Von Werra, and Thomas Wolf. The fineweb datasets: Decanting the web for the finest text data at scale. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024. URL https://openreview.net/forum?id=n6SCkn2QaG.

[32] Boris T Polyak and Anatoli B Juditsky. Acceleration of stochastic approximation by averaging. *SIAM journal on control and optimization*, 30(4):838–855, 1992.

[33] Xianbiao Qi, Marco Chen, Jiaquan Ye, Yelin He, and Rong Xiao. Delving into muon and beyond: Deep analysis and extensions. *arXiv preprint arXiv:2602.04669*, 2026.

[34] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015.

[35] David Ruppert. Efficient estimations from a slowly convergent robbins-monro process. Technical report, Cornell University Operations Research and Industrial Engineering, 1988.

[36] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[37] Levent Sagun, Leon Bottou, and Yann LeCun. Eigenvalues of the hessian in deep learning: Singularity and beyond. *arXiv preprint arXiv:1611.07476*, 2016.

[38] Levent Sagun, Utku Evci, V Ugur Guney, Yann Dauphin, and Leon Bottou. Empirical analysis of the hessian of over-parametrized neural networks. *arXiv preprint arXiv:1706.04454*, 2017.

[39] Andrei Semenov, Matteo Pagliardini, and Martin Jaggi. Benchmarking optimizers for large language model pretraining, 2026. URL https://openreview.net/forum?id=Jw7khYzYzl.

[40] Wei Shen, Ruichuan Huang, Minhui Huang, Cong Shen, and Jiawei Zhang. On the convergence analysis of muon. *arXiv preprint arXiv:2505.23737*, 2025.

[41] Minhak Song, Kwangjun Ahn, and Chulhee Yun. Does SGD really happen in tiny subspaces? In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=v6iLQBoIJw.

[42] Minhak Song, Beomhan Baek, Kwangjun Ahn, and Chulhee Yun. Through the river: Understanding the benefit of schedule-free methods for language model training. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=CGx4XU9rCA.

[43] Wei Tao, Zhisong Pan, Gaowei Wu, and Qing Tao. Primal averaging: A new gradient evaluation step to attain the optimal individual convergence. *IEEE transactions on cybernetics*, 50(2): 835–845, 2018.

[44] Bhavya Vasudeva, Puneesh Deora, Yize Zhao, Vatsal Sharan, and Christos Thrampoulidis. How muon's spectral design benefits generalization: A study on imbalanced data. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=YzjS4jcfmS.

[45] Jinbo Wang, Mingze Wang, Zhanpeng Zhou, Junchi Yan, Weinan E, and Lei Wu. The sharpness disparity principle in transformers for accelerating language model pre-training. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=DZ6iFdVDrx.

[46] Mingze Wang, Jinbo Wang, Haotian He, Zilin Wang, Guanhua Huang, Feiyu Xiong, Zhiyu li, Weinan E, and Lei Wu. Improving generalization and convergence by enhancing implicit regularization. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*, 2024. URL https://openreview.net/forum?id=cjM2bhLOiC.

[47] Kaiyue Wen, Zhiyuan Li, Jason S. Wang, David Leo Wright Hall, Percy Liang, and Tengyu Ma. Understanding warmup-stable-decay learning rates: A river valley loss landscape view. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=m51BgoqvbP.

[48] Yuxin Wu and Justin Johnson. Rethinking "batch" in batchnorm. *arXiv preprint arXiv:2105.07576*, 2021.

[49] Zhewei Yao, Amir Gholami, Kurt Keutzer, and Michael W Mahoney. Pyhessian: Neural networks through the lens of the hessian. In *2020 IEEE international conference on big data (Big data)*, pages 581–590. IEEE, 2020.

[50] Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. *CoRR*, abs/1605.07146, 2016. URL http://arxiv.org/abs/1605.07146.

[51] Minxin Zhang, Yuxuan Liu, and Hayden Schaeffer. Adam improves muon: Adaptive moment estimation with orthogonalized momentum. *arXiv preprint arXiv:2602.17080*, 2026.

[52] Yechen Zhang, Shuhao Xing, Junhao Huang, Kai Lv, Yunhua Zhou, Xipeng Qiu, Qipeng Guo, and Kai Chen. Mousse: Rectifying the geometry of muon with curvature-aware preconditioning. *arXiv preprint arXiv:2603.09697*, 2026.

[53] Wenjie Zhou, Bohan Wang, Wei Chen, and Xueqi Cheng. Bsfa: Leveraging the subspace dichotomy to accelerate neural network training. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 18845–18860, 2025.

[54] Shuchen Zhu, Rizhen Hu, Mingze Wang, Mou Sun, Xue Wang, Kun Yuan, and Zaiwen Wen. Accelerating llm pre-training through flat-direction dynamics enhancement. *arXiv preprint arXiv:2602.22681*, 2026.

Contents

Appendix A. Related Works

A.1. River-Valley Loss Landscape

Previous studies have characterized neural network training through a low-rank Hessian structure, consisting of a few high-curvature directions and a much broader bulk component [11, 28–30, 37, 38, 49]. Recent work further interprets this geometry through the river-valley landscape (see Figure 1 for illustration), where dominant directions form steep valley walls while the bulk defines the flatter river floor where most useful progress occurs [2, 6, 10, 41, 45–47, 53].

Wen et al. [47] use this geometry to explain the success of Warmup-Stable-Decay (WSD) schedules in large language model pretraining: the stable phase facilitates rapid traversal along the river, while the decay phase helps the iterate settle onto the valley floor. Extending this perspective, Belloni et al. [2] show that a similar geometry also arises in convolutional networks trained on image classification tasks. Song et al. [41] show that projecting updates only onto the dominant subspace is insufficient to reduce training loss, while updates projected onto the bulk subspace remain effective. Recently, several works have leveraged the river-valley landscape to design optimizers that either amplify updates along river directions or stabilize oscillations along valley directions [21, 24, 52, 54].

A.2. Schedule-Free Optimizer

Defazio et al. [9] introduce the schedule-free (SF) optimizer as an interpolation between Polyak–Ruppert (PR) averaging [32, 35] and primal averaging [26, 43]. Its update is given by

$$\begin{aligned} \boldsymbol{y}_t &= (1-\beta)\boldsymbol{z}_t + \beta \boldsymbol{x}_t, \\ \boldsymbol{z}_{t+1} &= \boldsymbol{z}_t - \eta \Delta_t, \\ \boldsymbol{x}_{t+1} &= (1-c_{t+1})\,\boldsymbol{x}_t + c_{t+1}\boldsymbol{z}_{t+1}, \end{aligned} \tag{4}$$

where η is the learning rate, Δ_t is the base optimizer update computed from the gradient at $\boldsymbol{y}_t$, $c_{t+1} = 1/(t+1)$, and the initialization satisfies $\boldsymbol{z}_1 = \boldsymbol{x}_1$. The framework can be combined with different base optimizers through the choice of Δ_t. For example, defining Δ_t as a stochastic gradient at $\boldsymbol{y}_t$ gives SF-SGD, whereas computing Δ_t using an RMSProp update with decoupled weight decay gives SF-AdamW.

The three sequences play distinct roles: $\boldsymbol{y}_t$ is the gradient evaluation point, $\boldsymbol{z}_t$ is the primary iterate that follows the base optimizer update, and $\boldsymbol{x}_t$ is the averaged sequence used for inference. The interpolation parameter β controls whether gradient evaluation tracks the fast-moving base trajectory ($\boldsymbol{z}_t$) or the more stable averaged trajectory ($\boldsymbol{x}_t$), thereby balancing rapid adaptation and stability. Under this formulation, SF optimizers are known to achieve a performance–time Pareto frontier, as evidenced by winning the Self-Tuning track of the 2024 AlgoPerf Challenge [7].

Song et al. [42] later connect the empirical success of SF optimization to the river-valley landscape, showing that, with an appropriate choice of β, the inference trajectory $\boldsymbol{x}_t$ of SF-AdamW follows the river throughout training. Their findings suggest that the implicit averaging in SF optimizer filters out high-curvature fluctuations from the valley walls. As a result, SF optimizer can remain near the river without requiring an explicit decay phase or an additional average of the model parameters.

A.3. Muon Optimizer

Muon [16] is a recently introduced optimizer that leverages matrix structure by orthogonalizing matrix-valued updates. At each iteration t, given a matrix $\boldsymbol{W}_t$ and its gradient $\boldsymbol{G}_t$, the updates are

defined as:

$$\begin{aligned} \boldsymbol{M}_t &= \mu \boldsymbol{M}_{t-1} + \boldsymbol{G}_t, \\ \boldsymbol{W}_{t+1} &= \boldsymbol{W}_t - \eta \mathcal{O}(\boldsymbol{M}_t), \end{aligned} \tag{5}$$

where η is the learning rate, μ is the momentum coefficient, and $\mathcal{O}$ denotes an orthogonalization operator, which is approximated using a Newton-Schulz iteration [3] for computational efficiency. For non-matrix-valued parameters, including embeddings and classification heads, updates are instead performed using standard optimizers such as SGD or Adam(W).

Recent theoretical studies have begun to explain Muon's rapid convergence through the normalization effect of its matrix-wise orthogonalization, especially in ill-conditioned settings. Shen et al. [40] show that Muon can be advantageous when neural network Hessians exhibit low-rank or structured spectral geometry, while Ma et al. [25] show that its orthogonalization can act as an effective preconditioner, yielding condition-number-free linear convergence in matrix factorization and linear transformer settings. Other recent studies further show that Muon's orthogonalized updates induce balanced component learning [17], improving minority-group performance in imbalanced settings [44]. However, this same mechanism can also strengthen noisy components, potentially leading to less stable dynamics [33, 52].

Appendix B. Conclusion

We introduce AMUSE, an optimization algorithm designed to stabilize Muon via SF method and a time-varying interpolation parameter β_t. We observe that Muon's orthogonalization operator promotes training by leveraging large bulk components, but also induces oscillations along valley walls, thereby contributing to its inherent instability. AMUSE adopts a SF framework, a principled approach that controls where the gradient is evaluated and stabilizes training by keeping the optimization trajectory close to the river geometry. With a carefully designed schedule for the interpolation parameter β_t, AMUSE outperforms conventional learning-rate-scheduled Adam(W) or Muon optimizers across a wide range of deep learning tasks. Our results highlight how a systematic understanding of training dynamics and the loss landscape provides a principled approach to optimizer design.

Despite the superior performance of AMUSE compared with various optimizers including Muon, it requires additional memory overhead relative to vanilla Muon. Developing an optimization strategy that enhances updates along the river direction and suppresses valley-wall oscillations without incurring such overhead is a promising direction for future investigation.

Appendix C. A Quadratic Example of Bouncing in Matrix-Normalized Updates

We compare gradient descent with an idealized matrix-normalized update on a two-dimensional matrix quadratic. Let $\boldsymbol{W} \in \mathbb{R}^{2\times 2}$ be the parameter matrix, and let $\boldsymbol{A} \in \mathbb{R}^{2\times 2}$ be a positive definite curvature matrix with anisotropic spectrum,

$$\boldsymbol{A} = \begin{pmatrix} \lambda & 0 \\ 0 & 1 \end{pmatrix}, \qquad \lambda \gg 1.$$

Here, λ controls the curvature gap between the first and second coordinates. We consider the quadratic objective

$$f(\boldsymbol{W}) = \frac{1}{2}\,\mathrm{tr}(\boldsymbol{W}^\top \boldsymbol{A}\boldsymbol{W}).$$

Its gradient is

$$\nabla f(\boldsymbol{W}) = \boldsymbol{A}\boldsymbol{W}.$$

For the diagonal iterate

$$\boldsymbol{W}_t = \begin{pmatrix} a_t & 0 \\ 0 & b_t \end{pmatrix},$$

we have

$$\nabla f(\boldsymbol{W}_t) = \begin{pmatrix} \lambda a_t & 0 \\ 0 & b_t \end{pmatrix}.$$

Gradient descent with step size η gives

$$\boldsymbol{W}_{t+1}^{\mathrm{GD}} = \boldsymbol{W}_t - \eta \nabla f(\boldsymbol{W}_t),$$

and hence

$$a_{t+1}^{\mathrm{GD}} = (1-\eta\lambda)a_t, \qquad b_{t+1}^{\mathrm{GD}} = (1-\eta)b_t.$$

If $0 < \eta < 1/\lambda$, then $0 < 1-\eta\lambda < 1$, so

$$|a_{t+1}^{\mathrm{GD}}| < |a_t|, \qquad \mathrm{sign}(a_{t+1}^{\mathrm{GD}}) = \mathrm{sign}(a_t).$$

The update magnitude in the first coordinate is

$$|\Delta a_t^{\mathrm{GD}}| = \eta\lambda|a_t|,$$

which goes to zero as $a_t \to 0$.

Now consider a matrix-normalized update, which idealizes the normalization effect of Muon. For a full-rank matrix $\boldsymbol{G}$, define its polar factor as

$$\mathrm{Polar}(\boldsymbol{G}) = \boldsymbol{G}(\boldsymbol{G}^\top \boldsymbol{G})^{-1/2}.$$

The idealized matrix-normalized update is

$$\boldsymbol{W}_{t+1}^{\mathrm{MN}} = \boldsymbol{W}_t - \eta\ \mathrm{Polar}(\nabla f(\boldsymbol{W}_t)).$$

For the same diagonal iterate, let

$$\boldsymbol{G}_t = \nabla f(\boldsymbol{W}_t) = \begin{pmatrix} \lambda a_t & 0 \\ 0 & b_t \end{pmatrix}.$$

When $a_t \neq 0$ and $b_t \neq 0$,

$$\boldsymbol{G}_t^\top \boldsymbol{G}_t = \begin{pmatrix} \lambda^2 a_t^2 & 0 \\ 0 & b_t^2 \end{pmatrix}, \qquad (\boldsymbol{G}_t^\top \boldsymbol{G}_t)^{-1/2} = \begin{pmatrix} |\lambda a_t|^{-1} & 0 \\ 0 & |b_t|^{-1} \end{pmatrix}.$$

Therefore,

$$\mathrm{Polar}(\boldsymbol{G}_t) = \begin{pmatrix} \mathrm{sign}(a_t) & 0 \\ 0 & \mathrm{sign}(b_t) \end{pmatrix}.$$

Thus, the matrix-normalized update gives

$$a_{t+1}^{\mathrm{MN}} = a_t - \eta\ \mathrm{sign}(a_t), \qquad b_{t+1}^{\mathrm{MN}} = b_t - \eta\ \mathrm{sign}(b_t).$$

If $0 < |a_t| < \eta$, then

$$a_t a_{t+1}^{\mathrm{MN}} < 0,$$

so the first coordinate changes sign in one step. Moreover,

$$|\Delta a_t^{\mathrm{MN}}| = \eta,$$

which does not decrease as $a_t \to 0$.

The contrast is therefore

$$|\Delta a_t^{\mathrm{GD}}| = \eta\lambda|a_t| \to 0 \qquad \text{as } a_t \to 0,$$

whereas

$$|\Delta a_t^{\mathrm{MN}}| = \eta.$$

Gradient descent preserves the magnitude information of the gradient, while the matrix-normalized update removes it through polar normalization. Consequently, a small nonzero component can still receive a finite update and change sign.

Appendix D. Additional Results and Experimental Details for Section 3

In this section we provide additional experimental details and results for the analysis in Section 2. We include experimental setups and additional results that were omitted from the main text due to space constraints.

D.1. Dominant Updates Can Cause Instability, while Bulk Updates Accelerate Training

In this subsection, we empirically show that updates along the dominant directions can cause training instability, while amplifying the bulk component can accelerate Muon training. Following Zhou et al. [53], we do this by decomposing each Muon update into its dominant and bulk components and scaling them separately.

We conduct experiments on three classification datasets: MNIST, CIFAR-10, and SST-2. Detailed experimental settings, including dataset construction and model architectures, are provided in Section D.2; hyperparameters are reported in the corresponding figure captions.

For the Hessian $\nabla^2\mathcal{L}(\theta_t)$, the dominant subspace $\mathcal{S}_k(\theta_t)$ is spanned by the top-k eigenvectors, and the bulk subspace is its orthogonal complement $\mathcal{S}_k^{\perp}(\theta_t)$. Following the observation that the Hessian spectrum of classification models contains a small number of outlier eigenvalues related to the number of classes, we set k to the number of classes: $k = 10$ for MNIST-5k and CIFAR-10-5k, and $k = 2$ for SST-2-1k.

The Muon update direction $\boldsymbol{u}_t = \mathcal{O}(\boldsymbol{m}_t)$ is decomposed into dominant and bulk components as $\boldsymbol{P}_k(\theta_t)\boldsymbol{u}_t$ and $\boldsymbol{P}_k^{\perp}(\theta_t)\boldsymbol{u}_t$, respectively. We then rescale the two components separately:

$$\widetilde{\boldsymbol{u}}_t = \alpha \boldsymbol{P}_k(\theta_t)\boldsymbol{u}_t + \gamma \boldsymbol{P}_k^{\perp}(\theta_t)\boldsymbol{u}_t,$$

where α and γ are multiplicative scaling factors for the dominant and bulk components, respectively. For example, when $\alpha = \gamma = 1$, this recovers the original Muon update. The parameter is updated by

$$\theta_{t+1} = \theta_t - \eta \widetilde{\boldsymbol{u}}_t.$$

As shown in Figure 6, larger α leads to more unstable training, indicating that dominant-subspace components can induce valley-wall oscillations. In contrast, larger γ improves training speed and leads to more stable convergence, suggesting that bulk directions are responsible for useful optimization progress. These results suggest that reducing dominant components before orthogonalization is important for training stability, while preserving Muon's strong bulk-oriented updates is important for fast training.

D.2. Experimental Details

We follow the experimental setup of Cohen et al. [5], Song et al. [41] for all experiments in D.3. For experiments in D.1, we use the same experimental setup except for introducing scaling factors following Zhou et al. [53].

D.2.1. ARCHITECTURE DETAILS

We consider three model architectures, chosen according to the input modality. For fully connected image classification, we use a three-layer MLP with width 200 and Tanh activations, following Cohen et al. [5], Song et al. [41]. For convolutional image classification, we use a three-layer CNN with

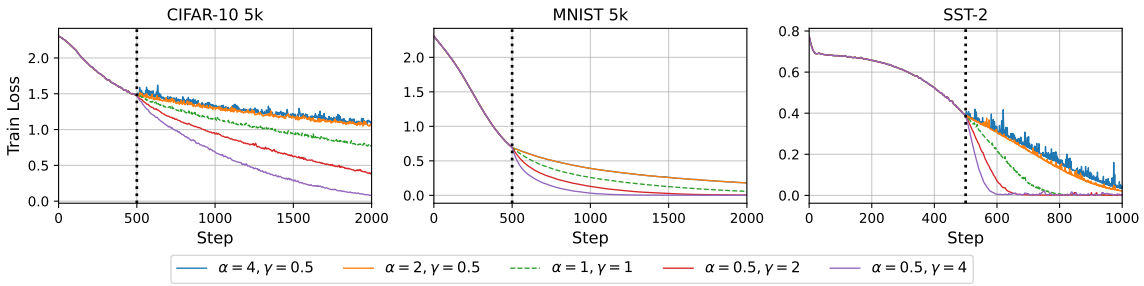


Figure 6: **Scaling Dominant (α) and Bulk (γ) components of Muon update.** We apply subspace-wise scaling to Muon updates starting at step 500. For CIFAR-10-5k and MNIST-5k, Muon is combined with SGD for non-Muon parameters, using momentum 0.9 and learning rate 5×10^{-4}. For SST-2-1k, Muon is combined with AdamW for non-Muon parameters, using Muon momentum 0.9, AdamW coefficients $(0.9, 0.99)$, and learning rate 1×10^{-4}.

width 32 and ReLU activations, following Cohen et al. [5], Song et al. [41]. For text classification, we use a two-layer Transformer with hidden dimension 64 and 8 attention heads, following Damian et al. [8], Song et al. [41].

D.2.2. DATASETS

We use three small-scale classification benchmarks constructed from MNIST, CIFAR-10, and SST-2. For MNIST and CIFAR-10, we use the first 5,000 training examples, each with 10 classes. For SST-2, we use the first 1,000 examples for binary sentiment classification. The main text reports the MNIST-5k experiment with MLP architecture in Section 2.

D.2.3. EXPERIMENTAL SETUP

In all experiments in this section, including those in Section 2, we use cross-entropy loss with batch size 50. We use a constant learning rate for all optimizers: SGD, AdamW, Muon, and SF-Muon. For Muon and SF-Muon, non-hidden matrix layers are optimized with SGD in the CIFAR-10-5k and MNIST-5k experiments, and with AdamW (and its SF variant in SF-Muon) in the SST-2-1k experiment.

The hyperparameters for each experiment are reported in Tables 2, 1, and 3. Unless otherwise specified, each figure uses the hyperparameters corresponding to its dataset.

For figures requiring additional hyperparameters, we use the following settings:

- Figure 2, Figure 3, and Figure 7: MNIST hyperparameters in Table 1.
- Figure 2: AMUSE with learning rate 0.01, $\beta_1 = 0.8$, and $\rho = 0.6$.
- Figure 7, Figure 8, and Figure 10: CIFAR-10 hyperparameters in Table 2.
- Figure 7, Figure 8, and Figure 11: SST-2 hyperparameters in Table 3.

Table 1: **Hyperparameters for MNIST-5k Experiments.**

Optimizer	Learning rate	Momentum
SGD	1e-2	–
AdamW	5e-4	(0.9,0.99)
Muon	1e-3	0.9
SF-Muon ($\beta = 0.9$)	2e-3	0.95

Table 2: **Hyperparameters for CIFAR10-5k Experiments.**

Optimizer	Learning rate	Momentum
SGD	1e-2	–
AdamW	5e-4	(0.9,0.99)
Muon	2e-3	0.9
SF-Muon ($\beta = 0.8$)	2e-3	0.9

Table 3: **Hyperparameters for SST2-1k Experiments.**

Optimizer	Learning rate	AdamW Momentum	Muon momentum
AdamW	5e-4	(0.9,0.99)	–
Muon	2e-3	(0.9,0.99)	0.9
SF-Muon ($\beta = 0.8$)	1e-5	(–,0.99)	0.9

D.3. Additional Experimental Results

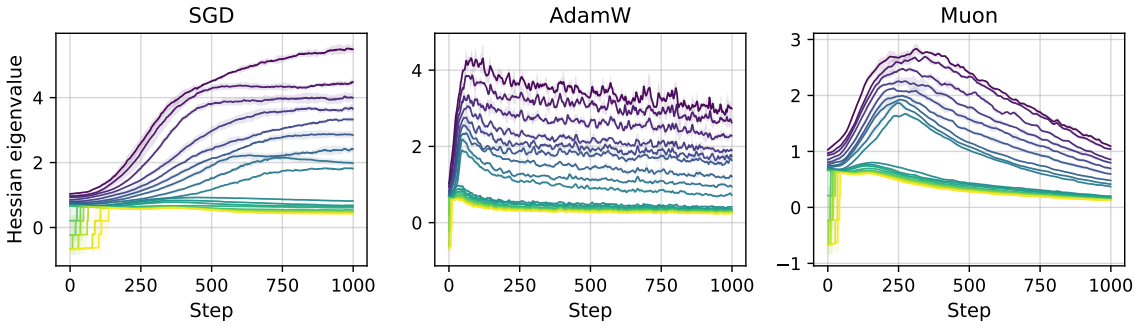
In this section, we provide additional results for the analysis in Section 2. We organize the results into three parts: Hessian eigenspectra, Muon bulk-ratio behavior, and SF-Muon dynamics.

Hessian eigenspectrum. We first examine the loss Hessian eigenspectrum across datasets and architectures. Figure 7 shows that the Hessian spectrum consistently contains a small number of large outlier eigenvalues, while the remaining eigenvalues are much smaller. This separation supports the dominant/bulk decomposition used in the main text.

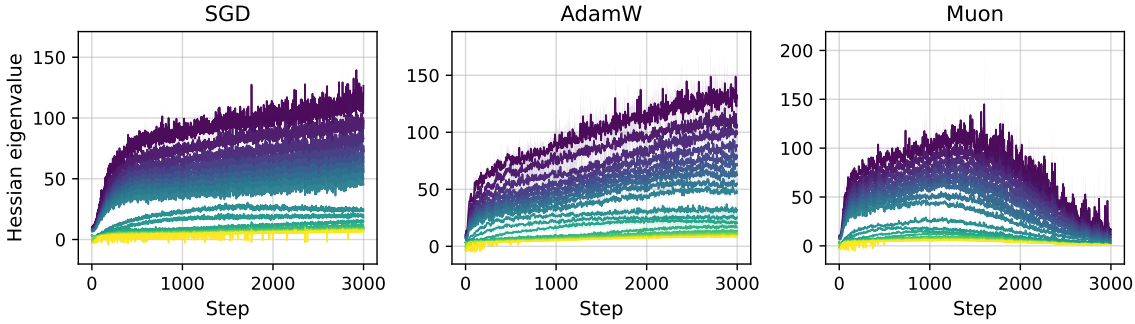
Bulk-oriented updates of Muon. We next verify that the bulk-oriented behavior of Muon extends beyond the main setting. Figures 8 and 9 show that Muon produces substantially larger bulk-ratio updates than SGD or AdamW across CIFAR-10 with a CNN and SST-2 with a Transformer. The same figures also compare the pre- and post-orthogonalized Muon updates, showing that the orthogonalization step itself increases the bulk component of the update. These results reinforce the explanation that Muon accelerates training by promoting movement in bulk directions, as mentioned in Section D.1.

Schedule-Free Muon dynamics. Finally, we repeat the SF-Muon analysis from the main text (Figure 3) on additional settings. Figures 10 and 11 compare the dominant/bulk components of Muon and SF-Muon, and also gradients at multiple virtual interpolation points $\boldsymbol{y}_t^{(\alpha)} = (1-\alpha)\boldsymbol{z}_t + \alpha\boldsymbol{x}_t$. In both settings, the averaged SF-Muon trajectory has a much smaller dominant component than vanilla

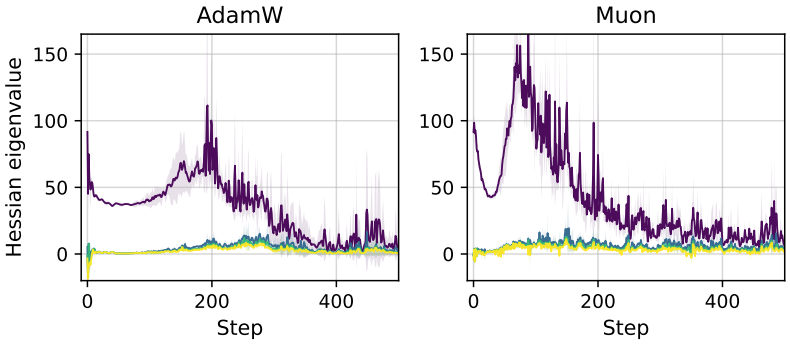
Muon, and gradients evaluated closer to x_t become more bulk-oriented. These results further support the claim that evaluating gradient near averaged iterate can suppress valley-wall contributions before the Muon orthogonalization step, as mentioned in Section 2.2.



(a) MNIST with MLP

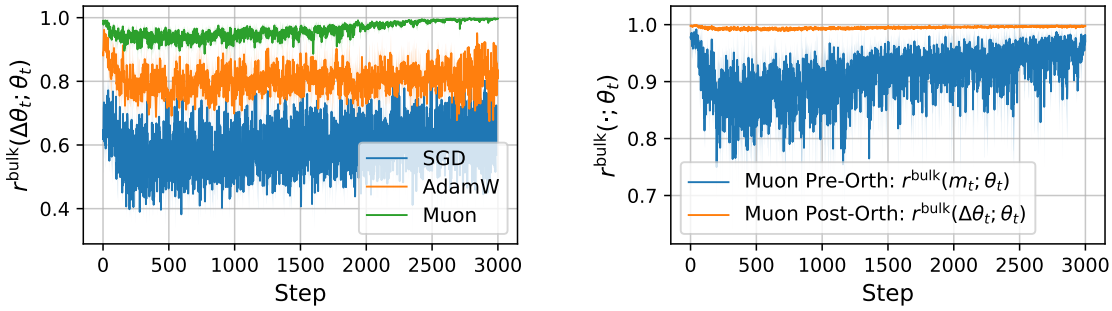


(b) CIFAR-10 with CNN



(c) SST-2 with Transformer

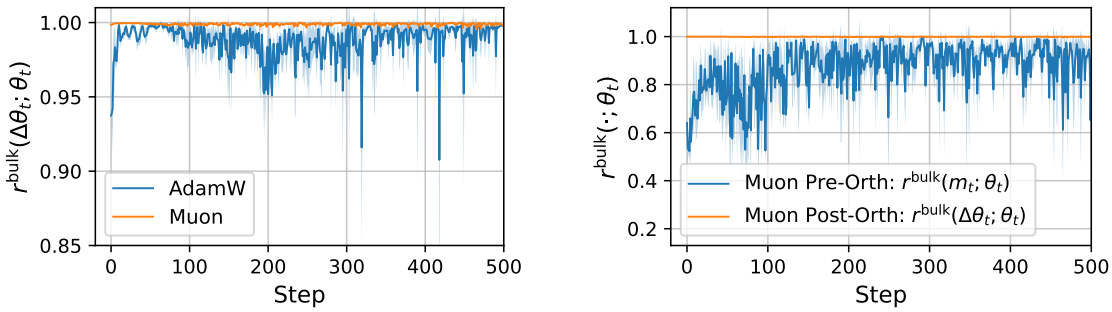
Figure 7: **Eigenspectra of the loss Hessian across optimizers, datasets, and architectures.** Across all cases, a small number of top-k eigenvalues, where k corresponds to the number of classes, form clear outliers, while the remaining eigenvalues are significantly smaller. This separation highlights a low-dimensional dominant subspace and a high-dimensional bulk subspace.



(a) Muon vs. SGD/AdamW

(b) Pre- vs. post-orthogonalization

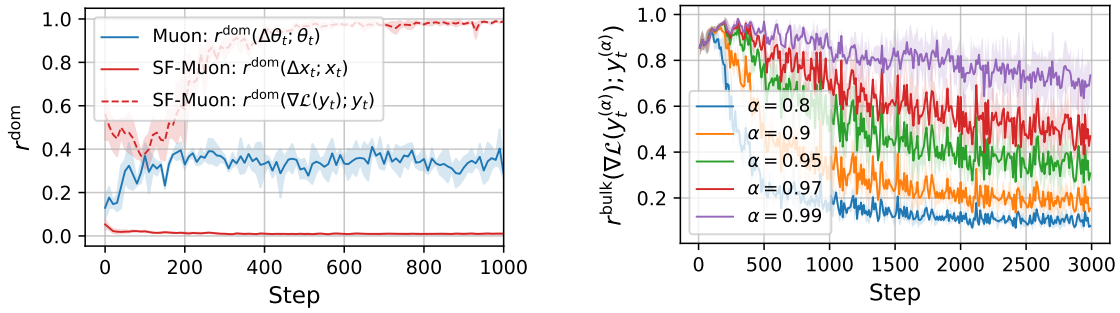
Figure 8: **Comparison of bulk component ratios on CIFAR-10 with CNN.** Metrics are measured on a CIFAR-10 5k subset using a CNN architecture. (**a**) Muon consistently produces substantially larger bulk component updates than SGD and AdamW. (**b**) Muon's orthogonalization increases the bulk ratio relative to the pre-orthogonalized momentum. Results are averaged over three runs.



(a) Muon vs. SGD/AdamW

(b) Pre- vs. post-orthogonalization

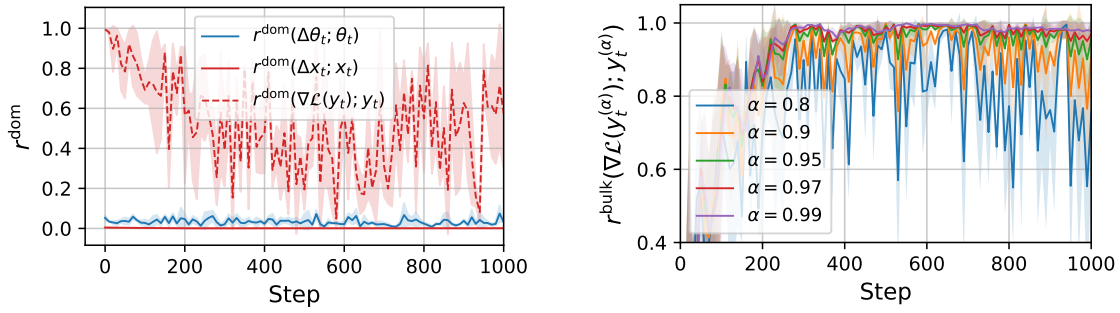
Figure 9: **Comparison of bulk component ratios on SST-2 with Transformer.** Metrics are measured on SST-2 using a Transformer architecture. (**a**) Muon consistently produces substantially larger bulk component updates than AdamW. (**b**) Muon's orthogonalization increases the bulk ratio relative to the pre-orthogonalized momentum. Results are averaged over three runs.



(a) Dominant ratio of SF-Muon vs. Muon

(b) Bulk ratio of gradient at $\boldsymbol{y}_t^{(\alpha)}$

Figure 10: **Comparison of dominant and bulk component ratios on CIFAR-10 with CNN.** (**a**) Dominant ratios for Muon (blue) and SF-Muon (red). The solid and dashed red lines denote the dominant components of $\Delta\boldsymbol{x}_t$ and $\nabla\mathcal{L}(\boldsymbol{y}_t)$, respectively. (**b**) Bulk ratios for gradients at $\boldsymbol{y}_t^{(\alpha)} = (1-\alpha)\boldsymbol{z}_t + \alpha\boldsymbol{x}_t$ across varying α, demonstrating that larger α values correlate with higher bulk components.



(a) SF-Muon vs. Muon

(b) Bulk ratio of gradient at $\boldsymbol{y}_t^{(\alpha)}$

Figure 11: **Comparison of bulk component ratios on SST-2 with Transformer.** (**a**) Dominant ratios for Muon (blue) and SF-Muon (red). The solid and dashed red lines denote the dominant components of $\Delta\boldsymbol{x}_t$ and $\nabla\mathcal{L}(\boldsymbol{y}_t)$, respectively. (**b**) Bulk ratios for gradients at $\boldsymbol{y}_t^{(\alpha)} = (1-\alpha)\boldsymbol{z}_t + \alpha\boldsymbol{x}_t$ across varying α, demonstrating that larger α values correlate with higher bulk components.

Appendix E. Details for AMUSE

E.1. Algorithmic Description of AMUSE

In this section, we present the full procedure of AMUSE, summarized in Algorithm 1.

Algorithm 1 AMUSE: Anytime Muon with Stable Gradient Evaluation

Require: learning rate η, weight decay λ, AMUSE β_1, AMUSE ρ, Muon momentum μ, AdamW second-moment coefficient β_2, numerical constant ϵ, warmup step T_0

Initialize $\boldsymbol{X}_1 \leftarrow \boldsymbol{Z}_1$, $\boldsymbol{M}_0 \leftarrow \boldsymbol{0}$, $\boldsymbol{V}_0 \leftarrow \boldsymbol{0}$
for $t = 1, 2, \ldots, T$ **do**
  $\eta_t \leftarrow \eta \cdot \min(1, t/T_0)$
  $c_{t+1} \leftarrow \dfrac{\eta_t^2}{\sum_{i=1}^{t} \eta_i^2}$
  if $t \le T_0$ **then**
    $\beta_1^{(t)} \leftarrow \beta_1$
  else
    $\beta_1^{(t)} \leftarrow 1 - \left(\dfrac{T_0 - 1}{t - 1}\right)^{\rho} (1 - \beta_1)$
  end if
  $\boldsymbol{Y}_t \leftarrow (1 - \beta_1^{(t)})\boldsymbol{Z}_t + \beta_1^{(t)} \boldsymbol{X}_t$
  $\boldsymbol{G}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{Y}_t)$
  if parameter group uses Muon **then**
    $\boldsymbol{M}_t \leftarrow \mu \boldsymbol{M}_{t-1} + \boldsymbol{G}_t$
    $\boldsymbol{O}_t \leftarrow \text{Newton-Schulz}(\boldsymbol{M}_t)$
    $\boldsymbol{Z}_{t+1} \leftarrow (1 - \eta_t \lambda)\boldsymbol{Z}_t - \eta_t \boldsymbol{O}_t$
  else
    $\boldsymbol{V}_t \leftarrow \beta_2 \boldsymbol{V}_{t-1} + (1 - \beta_2)(\boldsymbol{G}_t \odot \boldsymbol{G}_t)$
    $\widehat{\boldsymbol{V}}_t \leftarrow \boldsymbol{V}_t / (1 - \beta_2^t)$
    $\boldsymbol{U}_t \leftarrow \boldsymbol{G}_t \oslash \left(\sqrt{\widehat{\boldsymbol{V}}_t} + \epsilon\right)$
    $\boldsymbol{Z}_{t+1} \leftarrow (1 - \eta_t \lambda)\boldsymbol{Z}_t - \eta_t \boldsymbol{U}_t$
  end if
  $\boldsymbol{X}_{t+1} \leftarrow (1 - c_{t+1})\boldsymbol{X}_t + c_{t+1}\boldsymbol{Z}_{t+1}$
end for
return $\boldsymbol{X}_{T+1}$

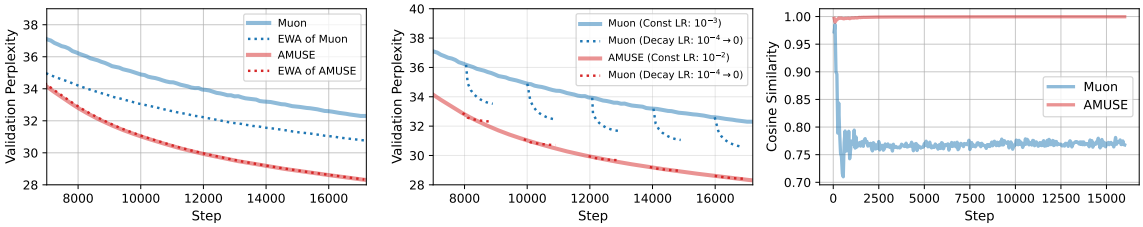


Figure 12: **Comparison between constant learning rate Muon and AMUSE in the 124M LLaMA setting**. **The left panel** shows the effect of EWA, where solid lines represent the original training trajectories and dotted lines represent their corresponding EWA trajectories. **The middle panel** shows the effect of learning rate decay, where after warmup we linearly decay the learning rate from 10^{-4} to 0 at selected iterations using Muon optimizer; solid lines denote the original trajectories, while dotted lines show the corresponding decay-phase trajectories. **The right panel** shows cosine similarity between consecutive updates, plotting $\cos(\Delta \boldsymbol{x}_t, \Delta \boldsymbol{x}_{t+1})$ for AMUSE and $\cos(\Delta \theta_t, \Delta \theta_{t+1})$ for Muon.

E.2. AMUSE Follows the River

Following Song et al. [42], we test whether AMUSE already follows the river using two post-hoc evaluations: Exponential Weight Averaging (EWA) and an additional learning rate decay phase. If the optimizer still oscillates across valley walls, these procedures should substantially reduce the loss by moving the iterate closer to the river; if it already tracks the river, they should offer little benefit.

As shown in Figure 12, constant learning rate Muon benefits significantly from both EWA and learning rate decay, indicating strong valley-wall oscillation. In contrast, AMUSE shows little improvement from either procedure, suggesting it already stays near the river. This stable progress is further confirmed by AMUSE's high cosine similarity between consecutive updates of $\boldsymbol{x}_t$.

E.3. Comparison of fixed-β SF-Muon with different β values and AMUSE

We compare AMUSE with fixed-β SF-Muon using multiple constant β values on 124M LLaMA pretraining. All hyperparameters are fixed except the interpolation coefficient: SF-Muon uses constant β values, while AMUSE uses $\beta_1 = 0.6$ and $\rho = 0.8$. As shown in Figure 13, none of the fixed-β variants outperforms AMUSE at any point during training.

Fixed-β variants exhibit an early–late trade-off: smaller β values improve early perplexity but saturate later, while larger β values improve late-stage stability at the cost of slower early progress. AMUSE avoids this trade-off by gradually increasing β_t, achieving lower validation perplexity while maintaining a larger averaged-update norm $\|\Delta \boldsymbol{x}_t\|$. This suggests that AMUSE preserves faster progress along the river while stabilizing the gradient-evaluation point over training.

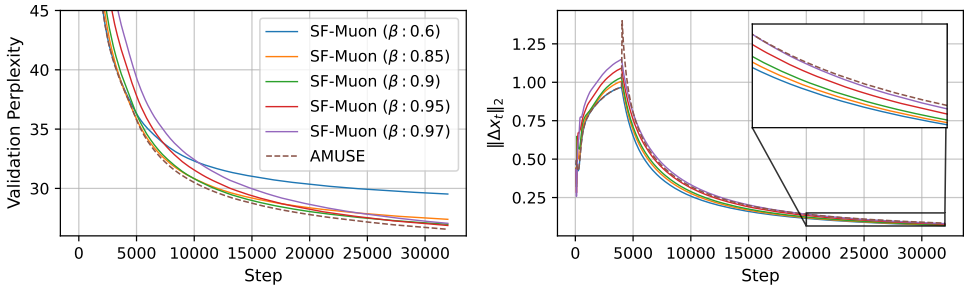


Figure 13: **Comparison of fixed-β SF-Muon with different β values and AMUSE in the 124M LLaMA pretraining on FineWeb.** Solid lines show fixed-β SF-Muon, and dashed lines show AMUSE with $\beta_1 = 0.6$ and $\rho = 0.8$. We report validation perplexity **(left)** and the update norm $\|\Delta \boldsymbol{x}_t\|$ **(right)**.

E.4. Averaging Orthogonalized Updates

We provide the derivation of the closed-form expression for the averaged sequence $\boldsymbol{X}_t$. This shows that the motion of $\boldsymbol{X}_t$ is a weighted average of past orthogonalized Muon updates.

Recall that

$$\boldsymbol{Z}_{t+1} = \boldsymbol{Z}_t - \eta \mathcal{O}(\boldsymbol{M}_t), \qquad \boldsymbol{X}_{t+1} = \left(1 - \frac{1}{t+1}\right) \boldsymbol{X}_t + \frac{1}{t+1} \boldsymbol{Z}_{t+1}, \tag{6}$$

with initialization $\boldsymbol{Z}_1 = \boldsymbol{X}_1$. Since $c_{t+1} = 1/(t+1)$, the recursion for $\boldsymbol{X}_t$ is the online average of the base iterates:

$$\boldsymbol{X}_t = \frac{1}{t} \sum_{i=1}^{t} \boldsymbol{Z}_i. \tag{7}$$

We verify this by induction. For $t = 1$, Eq. (7) holds since $\boldsymbol{X}_1 = \boldsymbol{Z}_1$. Assume that $\boldsymbol{X}_t = \frac{1}{t} \sum_{i=1}^{t} \boldsymbol{Z}_i$. Then

$$\boldsymbol{X}_{t+1} = \left(1 - \frac{1}{t+1}\right) \boldsymbol{X}_t + \frac{1}{t+1} \boldsymbol{Z}_{t+1} \tag{8}$$

$$= \frac{t}{t+1} \left(\frac{1}{t} \sum_{i=1}^{t} \boldsymbol{Z}_i\right) + \frac{1}{t+1} \boldsymbol{Z}_{t+1} \tag{9}$$

$$= \frac{1}{t+1} \sum_{i=1}^{t+1} \boldsymbol{Z}_i. \tag{10}$$

Thus Eq. (7) holds for all t.

Next, each base iterate can be written as

$$\boldsymbol{Z}_i = \boldsymbol{Z}_1 - \eta \sum_{j=1}^{i-1} \mathcal{O}(\boldsymbol{M}_j). \tag{11}$$

Substituting Eq. (11) into Eq. (7) gives

$$\boldsymbol{X}_t = \frac{1}{t}\sum_{i=1}^{t}\left(\boldsymbol{Z}_1 - \eta\sum_{j=1}^{i-1}\mathcal{O}(\boldsymbol{M}_j)\right) \tag{12}$$

$$= \boldsymbol{Z}_1 - \frac{\eta}{t}\sum_{i=1}^{t}\sum_{j=1}^{i-1}\mathcal{O}(\boldsymbol{M}_j). \tag{13}$$

For a fixed j, the term $\mathcal{O}(\boldsymbol{M}_j)$ appears for $i = j+1, \dots, t$, hence $t-j$ times. Therefore,

$$\boldsymbol{X}_t = \boldsymbol{Z}_1 - \frac{\eta}{t}\sum_{j=1}^{t-1}(t-j)\mathcal{O}(\boldsymbol{M}_j). \tag{14}$$

We now derive the averaged-sequence update

$$\Delta\boldsymbol{X}_t := \boldsymbol{X}_{t+1} - \boldsymbol{X}_t.$$

Using Eq. (14),

$$\boldsymbol{X}_{t+1} = \boldsymbol{Z}_1 - \frac{\eta}{t+1}\sum_{j=1}^{t}(t+1-j)\mathcal{O}(\boldsymbol{M}_j), \tag{15}$$

$$\boldsymbol{X}_t = \boldsymbol{Z}_1 - \frac{\eta}{t}\sum_{j=1}^{t-1}(t-j)\mathcal{O}(\boldsymbol{M}_j). \tag{16}$$

Thus,

$$\Delta\boldsymbol{X}_t = -\frac{\eta}{t+1}\sum_{j=1}^{t}(t+1-j)\mathcal{O}(\boldsymbol{M}_j) + \frac{\eta}{t}\sum_{j=1}^{t-1}(t-j)\mathcal{O}(\boldsymbol{M}_j). \tag{17}$$

For $j = 1, \dots, t-1$, the coefficient of $\mathcal{O}(\boldsymbol{M}_j)$ is

$$\eta\left(\frac{t-j}{t} - \frac{t+1-j}{t+1}\right) = -\frac{\eta j}{t(t+1)}. \tag{18}$$

For $j = t$, only the first sum contributes, giving

$$-\frac{\eta}{t+1} = -\frac{\eta t}{t(t+1)}.$$

Combining both cases,

$$\Delta\boldsymbol{X}_t = -\frac{\eta}{t(t+1)}\sum_{j=1}^{t} j\,\mathcal{O}(\boldsymbol{M}_j). \tag{19}$$

Therefore, $\boldsymbol{X}_t$ averages the trajectory after orthogonalized Muon updates have been applied, and $\Delta\boldsymbol{X}_t$ is a weighted average of previous orthogonalized updates. This is distinct from momentum: momentum smooths gradients before orthogonalization, while $\boldsymbol{X}_t$ averages the already orthogonalized updates.

E.5. Derivation of the β_t Schedule

In designing the schedule for β_t in AMUSE, one might initially consider a monotonically increasing schedule toward 1. However, following the derivation by Song et al. [42], a closer inspection of the schedule-free (SF) update reveals that the $\boldsymbol{x}_t$ iterates perform an implicit averaging of the $\boldsymbol{y}_t$ iterates:

$$\boldsymbol{x}_t = (1-\omega_t)\boldsymbol{x}_{t-1} + \omega_t \boldsymbol{y}_t,$$

where the effective weight ω_t is given by:

$$\omega_t = \frac{1}{(t-1)(1-\beta_t)+1}.$$

Crucially, if β_t increases too rapidly toward 1, the term $(1-\beta_t)$ vanishes, causing the overall denominator to shrink and ω_t to increase. An increasing ω_t implies that the effective averaging window size actually *shrinks*. From an optimization perspective, shrinking the averaging window at later stages of training is highly counterproductive. Therefore, to prevent this adverse effect, we must establish a strict upper bound for β_t.

We define this upper bound by finding a schedule that strictly maintains the averaging window size—meaning ω_t remains constant—after a certain step T_0. Setting $\omega_t = \omega_{T_0}$, we solve for the corresponding boundary β_t:

$$\begin{aligned}\omega_t &= \frac{1}{(t-1)(1-\beta_t)+1} \\ &= \frac{1}{(T_0-1)(1-\beta_{T_0})+1} = \omega_{T_0},\end{aligned}$$

which simplifies to:

$$\beta_t = 1 - \frac{T_0-1}{t-1}(1-\beta_{T_0}).$$

Building upon this boundary condition, we introduce an interpolation parameter ρ to control the rate of window growth. This parameter interpolates between the standard fixed-β schedule ($\beta_t = \beta_{T_0}$) and the constant-window boundary derived above, yielding our final schedule for $t \geq T_0$:

$$\beta_t = 1 - \left(\frac{T_0-1}{t-1}\right)^{\rho}(1-\beta_{T_0}).$$

By restricting $\rho \in [0,1]$, we ensure a flexible design where the averaging window is allowed to grow ($0 \leq \rho < 1$) or at least remain constant ($\rho = 1$), effectively preventing it from ever shrinking.

Visualization of Averaging Window Size and β_t. To better understand the impact of our proposed schedule on the averaging dynamics, we visualize both the effective averaging window and the trajectory of β_t. By unrolling the recursive schedule-free update, the final evaluated parameter $\boldsymbol{x}_T$ can be explicitly expressed as a weighted sum of all past iterates $\boldsymbol{y}_t$, i.e., $\boldsymbol{x}_T = \sum_{t=1}^{T} \alpha_t \boldsymbol{y}_t$, where the global effective weight α_t assigned to each step t is defined as:

$$\alpha_t \triangleq \frac{1}{(t-1)(1-\beta_t)+1} \prod_{s=t+1}^{T} \left[\frac{(s-1)(1-\beta_s)}{(s-1)(1-\beta_s)+1}\right].$$

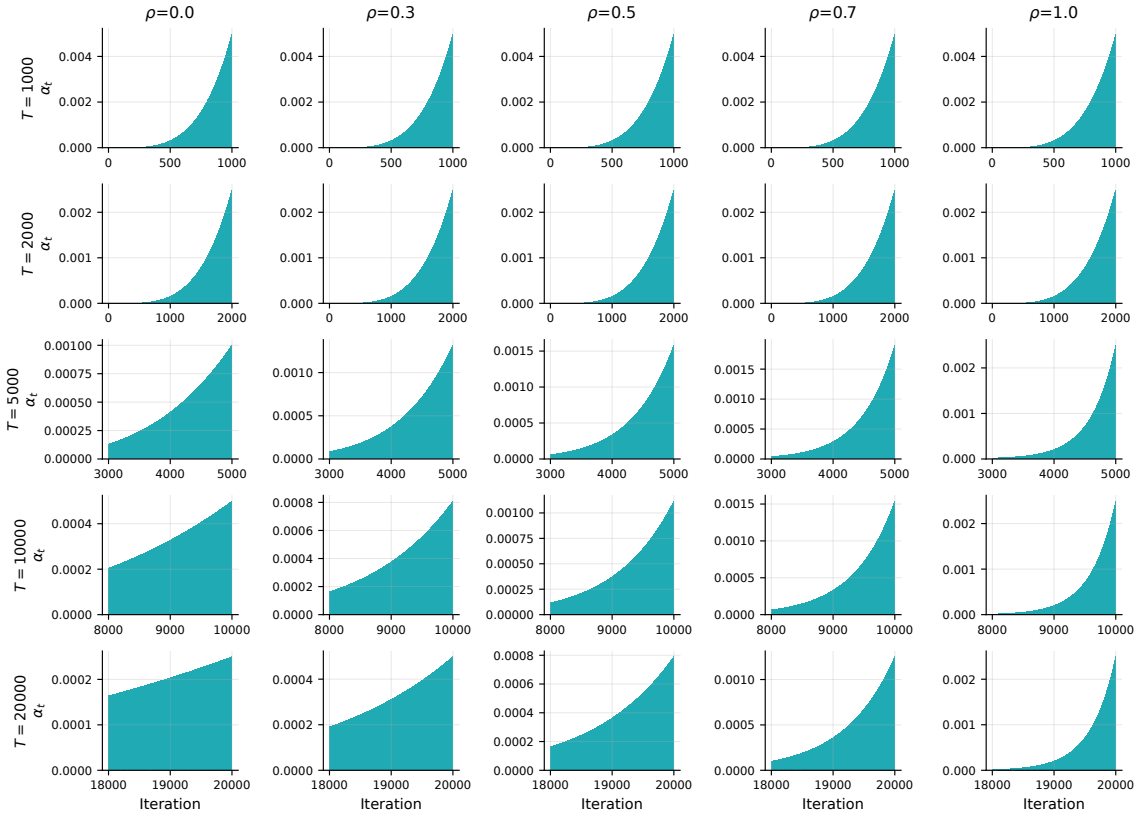


Figure 14: **Effect of ρ on effective weights α_t.** Histograms of α_t across varying sequence lengths T. For $T > T_0 = 2000$, lower ρ values allow the effective averaging window to continuously grow, whereas $\rho = 1$ strictly fixes the window to its size at T_0.

This weight α_t directly dictates the size and shape of the effective averaging window. In Figure 14, we plot the histogram of these weights α_t (initialized with $\beta_1 = 0.8$) to illustrate how much "memory" the final model retains from past steps. The exact shape of this distribution is governed by the interpolation parameter ρ. Specifically, when $\rho = 0$, the schedule allows the averaging window size to continuously increase in a linear fashion throughout training. In contrast, setting $\rho = 1$ strictly fixes the window size after the initial warmup phase ($T_0 = 2000$). For any intermediate value ($0 < \rho < 1$), the schedule smoothly interpolates between these two extremes, providing a controlled, sub-linear growth of the memory window. Furthermore, we visualize the actual evolution of β_t over time in Figure 15.

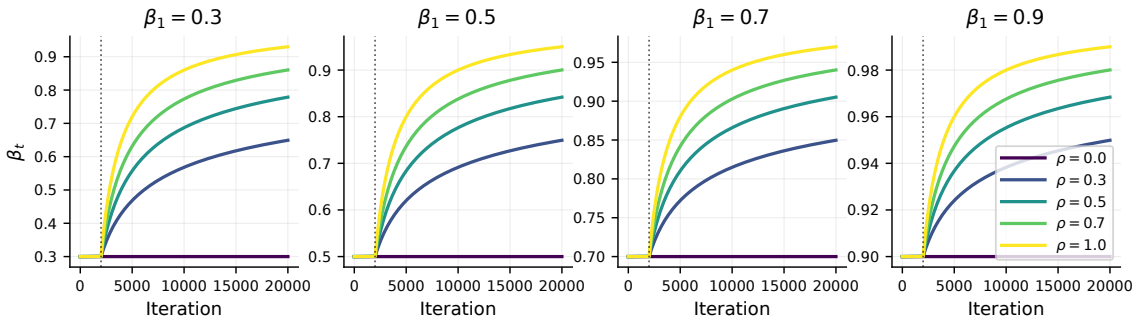


Figure 15: **Evolution of β_t controlled by ρ.** After the initial warmup phase ($T_0 = 2000$, dotted vertical line), ρ smoothly interpolates between a constant β baseline ($\rho = 0$) and the strict constant-window upper bound ($\rho = 1$).

E.6. Implementation Details

For language model pretraining and MAE fine-tuning, we optimize non-Muon parameters, such as embeddings, normalization layers, and output heads, with SF-AdamW; for other image-domain experiments, we use SF-SGD, following standard Muon practice.

Although AMUSE is formulated with three sequences $(\boldsymbol{x}_t, \boldsymbol{y}_t, \boldsymbol{z}_t)$ from the standard SF formulation and a muon momentum buffer $\boldsymbol{m}_t$, its practical implementation requires storing only the model parameters (kept at $\boldsymbol{y}_t$), the sequence state $\boldsymbol{z}_t$, and the momentum buffer $\boldsymbol{m}_t$. For evaluation, we temporarily compute $\boldsymbol{x}_t$ via in-place interpolation using $\boldsymbol{y}_t$ and $\boldsymbol{z}_t$, and revert the parameters afterward. Because of this design, AMUSE requires exactly one extra state copy ($\boldsymbol{z}_t$) compared to vanilla Muon, *incurring no additional memory overhead compared to AdamW and SF-AdamW*. While AMUSE still utilizes a standard linear warmup phase, $\eta_t = \eta \min(1, t/T_0)$, it does not require a subsequent learning rate decay phase.

Averaging coefficient. In the main text, we use the simplified intuition $c_t = 1/t$. In the implementation, we follow the learning-rate-weighted averaging rule used in Defazio et al. [9]. Let η_t denote the effective learning rate at step t, including linear warmup. Rather than using a uniform average during warmup, we set

$$c_{t+1} = \frac{\eta_t^2}{\sum_{i=1}^{t} \eta_i^2}.$$

This prevents very early iterations, where the learning rate is small, from being overweighted in the averaged sequence. After warmup, η_t becomes constant and the coefficient recovers the usual $1/t$-type decay.

Exact β_t schedule. In Appendix E.5, we use the simplified intuition $c_t = 1/t$ to derive the schedule for β_t. However, because c_t is not strictly equal to $1/t$ in practice, our actual implementation computes β_t using the generalized formulation based on the exact c_t:

$$\beta_t = 1 - \left(\frac{c_t(1 - c_{T_0})}{c_{T_0}(1 - c_t)} \right)^{\rho} (1 - \beta_{T_0}).$$

Batch normalization. Averaging methods, including Schedule-Free [9], require special care for models with BatchNorm. During training, gradients are evaluated at the interpolated sequence $\boldsymbol{Y}_t$, so BatchNorm running statistics are accumulated for the training-time parameter state. During evaluation, however, the returned model is the averaged sequence $\boldsymbol{X}_t$. Therefore, the stored BatchNorm running statistics may not match the parameters used for evaluation.

For models with BatchNorm, we first convert the optimizer/model to the $\boldsymbol{X}_t$-state and then refresh BatchNorm statistics by forwarding a small number of training batches without gradient updates. This is analogous to PreciseBN-style recalibration [48]. No such step is needed for normalization layers that do not maintain batch-dependent running statistics, such as LayerNorm or RMSNorm, which are used in our language model experiments.

Weight decay. Following Defazio et al. [9], weight decay may be applied either to the gradient-evaluation sequence $\boldsymbol{y}_t$ or to the base sequence $\boldsymbol{z}_t$. In our implementation, we apply the standard weight-decay term to $\boldsymbol{z}_t$. We keep decay at $\boldsymbol{y}_t$ as a separate optional variant, but do not use it unless explicitly stated.

Computational resources. We use up to 8 NVIDIA RTX A6000 GPUs for the image-domain experiments and the 124M LLaMA experiments. For experiments at the 720M scale and above, we use up to 8 NVIDIA A100 GPUs with data-parallel training.

E.7. Hyperparameter Sensitivity

In all experiments, we fix the Muon momentum μ and do not treat it as a tunable parameter. Therefore, compared to standard SF optimizers, AMUSE introduces only one additional hyperparameter, ρ, which controls how quickly the gradient evaluation point moves toward the averaged iterate. We evaluate the hyperparameter sensitivity of AMUSE to ρ and β_1 on Llama 124M (See Appendix H.4).

Appendix F. Image Domain Experiments

F.1. Experimental Details

For the image-domain experiments, we mostly follow the experimental setup of Defazio et al. [9]. We additionally include Muon as a new baseline and add the ISIC 2018 image segmentation task.

We follow the standard Muon hybrid setup [16]: Muon is applied to hidden matrix-valued layers, while SGD with momentum is used for non-Muon parameters. Only for MAE fine-tuning in image domain experiments, we instead use the Muon+AdamW hybrid setup commonly used for Transformer models [16, 20]: Muon is applied to the hidden-layer weight matrices of the ViT backbone, while AdamW is used for the classifier head and all one-dimensional parameters, including biases and LayerNorm parameters.

When available, we use the tuned hyperparameters reported by Defazio et al. [9] for SGD, SF-SGD, AdamW, and SF-AdamW. For Muon and AMUSE, we tune the learning rate, and weight decay following the same tuning protocol. For vanilla Muon, we also tune the learning rate of the auxiliary SGD optimizer separately, since using the same learning rate for Muon and SGD led to poor performance.

For AMUSE, we fix the Muon momentum to 0.95 and use the same learning rate for the auxiliary SGD optimizer, reducing the number of tuned hyperparameters. In the Muon+SGD image-domain setting, AMUSE uses SGD without momentum for the non-Muon parameters (SF-SGD update for non-Muon parameters). For MAE fine-tuning, we use Muon with AdamW for non-Muon parameters, AMUSE uses AdamW without first momentum and we fix the second-moment coefficient to $\beta_2 = 0.999$ (SF-AdamW update for non-Muon parameters). Thus, beyond the base learning rate and weight decay, AMUSE only additionally adjusts the AMUSE-specific parameters β_1 and ρ.

CIFAR-10. We train a Wide-ResNet-16-8 architecture [50] on the CIFAR-10 image classification [19] dataset. We use a batch size of 128 and train for 300 epochs.

Table 4: **Hyperparameters for CIFAR-10.**

SGD and SF-SGD

Hyperparameter	SGD	SF-SGD
Epochs	300	
Batch size	128	
Warmup ratio	0.05	
Seeds	5	
Learning Rate	0.2	10.0
Weight decay	0.0001	0.0001
SGD momentum β	0.9	–
SF-SGD β_1	–	0.9
LR decay scheduler	cosine	–

Muon and AMUSE

Hyperparameter	Muon	AMUSE
Epochs	300	
Batch size	128	
Warmup ratio	0.05	
Seeds	5	
Muon LR	0.05	0.2
SGD LR	0.0001	0.2
Weight decay	0.01	0.02
Muon momentum μ	0.9	0.95
SGD momentum β	0.9	–
AMUSE β_1	–	0.8
AMUSE ρ	–	0.3
LR decay scheduler	cosine	–

CIFAR-100. We train a DenseNet architecture [15] in the CIFAR-100 image classification dataset. We use a batch size of 128 and train for 300 epochs.

Table 5: **Hyperparameters for CIFAR-100.**

SGD and SF-SGD

Hyperparameter	SGD	SF-SGD
Epochs	300	
Batch size	64	
Warmup ratio	0.05	
Seeds	5	
Learning Rate	0.05	5.0
Weight decay	0.0002	0.0002
SGD momentum β	0.9	–
SF-SGD β_1	–	0.9
LR decay scheduler	cosine	–

Muon and AMUSE

Hyperparameter	Muon	AMUSE
Epochs	300	
Batch size	64	
Warmup ratio	0.05	
Seeds	5	
Muon LR	0.05	0.5
SGD LR	0.001	0.5
Weight decay	0.005	0.002
Muon momentum μ	0.9	0.95
SGD momentum β	0.9	–
AMUSE β_1	–	0.7
AMUSE ρ	–	0.4
LR decay scheduler	cosine	–

SVHN. We train a ResNet-3-96 architecture [12] on the Street View House Numbers (SVHN) dataset [27]. We use a batch size of 32 and train for 300 epochs.

Table 6: **Hyperparameters for SVHN.**

SGD and SF-SGD

Hyperparameter	SGD	SF-SGD
Epochs	300	
Batch size	32	
Warmup ratio	0.05	
Seeds	5	
Learning Rate	0.1	1.0
Weight decay	0.0001	0.0002
SGD momentum β	0.9	–
SF-SGD β_1	–	0.9
LR decay scheduler	cosine	–

Muon and AMUSE

Hyperparameter	Muon	AMUSE
Epochs	300	
Batch size	32	
Warmup ratio	0.05	
Seeds	5	
Muon LR	0.05	0.1
SGD LR	0.001	0.1
Weight decay	0.002	0.01
Muon momentum μ	0.9	0.95
SGD momentum β	0.9	–
AMUSE β_1	–	0.85
AMUSE ρ	–	0.2
LR decay scheduler	cosine	–

ImageNet-1k. We train a ResNet-50 architecture [12] on the ILSVRC 2012 [36] ImageNet-1k classification dataset. We use a batch size of 256 and train for 100 epochs.

Table 7: **Hyperparameters for ImageNet-1k.**

SGD and SF-SGD

Hyperparameter	SGD	SF-SGD
Epochs	100	
Batch size	256	
Warmup ratio	0.05	
Seeds	5	
Learning Rate	0.05	1.5
Weight decay	0.0001	0.00005
SGD momentum β	0.9	–
SF-SGD β_1	–	0.9
LR decay scheduler	cosine	–

Muon and AMUSE

Hyperparameter	Muon	AMUSE
Epochs	100	
Batch size	256	
Warmup ratio	0.05	
Seeds	5	
Muon LR	0.1	0.5
SGD LR	0.00001	0.5
Weight decay	0.001	0.0005
Muon momentum μ	0.9	0.95
SGD momentum β	0.9	–
AMUSE β_1	–	0.6
AMUSE ρ	–	0.8
LR decay scheduler	cosine	–

ISIC2018. We train a base U-Net architecture [34] on the ISIC 2018 dataset [4] for binary medical image segmentation. We use a batch size of 8 and train for 50 epochs. Our implementation is based on the U-Bench codebase[2].

Since this setup is newly added in our paper, we extensively tune the learning rate, weight decay, and momentum for each baseline. For vanilla Muon, we additionally tune the auxiliary SGD learning rate used for non-Muon parameters.

Table 8: **Hyperparameters for ISIC2018.**

SGD and SF-SGD

Hyperparameter	SGD	SF-SGD
Epochs	50	
Batch size	8	
Warmup ratio	0.05	
Seeds	5	
Learning Rate	0.02	2e-3
Weight decay	0.0005	0.01
SGD momentum β	0.9	–
SF-SGD β_1	–	0.9
LR decay scheduler	cosine	–

Muon and AMUSE

Hyperparameter	Muon	AMUSE
Epochs	50	
Batch size	8	
Warmup ratio	0.05	
Seeds	5	
Muon LR	0.01	0.2
SGD LR	0.001	0.2
Weight decay	0.001	0.05
Muon momentum μ	0.9	0.95
SGD momentum β	0.9	–
AMUSE β_1	–	0.6
AMUSE ρ	–	0.2
LR decay scheduler	cosine	–

MAE. We fine-tune a pretrained MAE ViT-Base model (`vit_base_patch16`) on ImageNet-1K for 100 epochs. The model uses 16×16 patches and the standard ViT-Base architecture with 12 transformer blocks, hidden size 768, and 12 attention heads.

All runs use the same AdamW-pretrained MAE pretrained initialization and differ only in optimizer-specific hyperparameters. Since MAE fine-tuning starts from an AdamW-pretrained checkpoint, we carefully tune the initial interpolation value β_1 for AMUSE. We use smaller β_1 values

2. https://github.com/FengheTan9/U-Bench

than in the other experiments to avoid a large early discrepancy between the base sequence and the averaged sequence, allowing the base trajectory to adapt smoothly to the fine-tuning objective. Our implementation uses the official mae code[3].

Table 9: **Hyperparameters for MAE fine-tuning.**

AdamW and SF-AdamW

Hyperparameter	**AdamW**	**SF-AdamW**
Epochs	100	
Batch size	256	
Warmup ratio	0.05	
Learning rate	5e-4	2e-3
Weight decay	0.05	0.05
AdamW β_1	0.9	–
SF-AdamW β_1	–	0.9
AdamW β_2	0.999	0.999
LR decay scheduler	cosine	–
Drop Path	0.1	
Reprob	0.25	
Mixup	0.8	
Cutmix	1.0	

Muon and AMUSE

Hyperparameter	**Muon**	**AMUSE**
Epochs	100	
Batch size	256	
Warmup ratio	0.05	
Learning rate	5e-4	2e-3
Weight decay	0.01	0.05
Muon momentum	0.95	0.95
AdamW β_1	0.9	–
AMUSE β_1	–	0.4
AdamW β_2	0.999	0.999
AMUSE ρ	–	0.3
LR decay scheduler	cosine	–
Drop Path	0.1	
Reprob	0.25	
Mixup	0.8	
Cutmix	1.0	

3. https://github.com/facebookresearch/mae

Appendix G. Large Language Model Experiments

G.1. Architecture Details

Our experimental setup follows Semenov et al. [39], using the codebase at https://github.com/epfml/llm-optimizer-benchmark. We use three decoder-only Llama-style models with approximately 124M, 720M, and 1B parameters. Across all model sizes, we use a sequence length of 512, the GPT-2 tokenizer with vocabulary size 50,304, RMSNorm with $\epsilon = 10^{-5}$, bias-free projections, SwiGLU MLPs with `multiple_of`=256, no dropout, and bfloat16 training. Table 10 summarizes the model-specific dimensions.

Table 10: **Model configurations for LLM experiments.**

Model size	Layers	Attention heads	Hidden size
124M	12	12	768
720M	12	16	2048
1B	24	28	1792

G.2. Hyperparameter Tuning

For AdamW, D-Muon, and SF-AdamW baselines, we use the tuned hyperparameters reported by Semenov et al. [39]. At the 124M scale, their tuning includes learning rate, warmup steps, weight decay, learning-rate scheduler, gradient clipping, and optimizer-specific momentum parameters. For AdamW, the searched optimizer-specific parameters are β_1 and β_2. For Muon (D-Muon), the search includes learning rate, Muon momentum, AdamW hyperparameters for one-dimensional parameters, and whether to use Nesterov momentum. For SF-AdamW, the search includes learning rate, warmup length, weight decay, gradient clipping, and the Schedule-Free AdamW parameters β_1 and β_2, without a learning-rate decay schedule.

For AMUSE, we do not introduce additional tuning of the underlying optimizer momentums. We fix the Muon momentum to 0.95, and keep the AdamW momentum parameters for non-Muon parameters fixed to 0.999 to avoid additional hyperparameter tuning. We tune hyperparameters introduced by AMUSE: the initial interpolation value β_1 and ρ. Specifically, we sweep $\beta_1 \in \{0.4, 0.6\}$ and $\rho \in \{0.6, 0.8\}$. Across our experiments, $\rho = 0.8$ consistently performs well.

Due to computational limitations, we do not perform additional hyperparameter tuning for the 1B-scale experiments. For all optimizers, we directly use the best hyperparameters from the 720M setting. We also provide the AMUSE hyperparameter search grid in Table 14.

G.2.1. HYPERPARAMETERS FOR 124M PARAMETERS MODEL

Table 11: **Selected hyperparameters at the 124M scale.**

AdamW and Muon (D-Muon)

Hyperparameter	AdamW	Muon
Learning rate	0.001	0.002
Batch size	256	256
Sequence length	512	512
Warmup steps	2000	2000
Weight decay	0.1	0.1
Gradient clipping	0.5	0.5
AdamW β_1	0.8	0.8
AdamW β_2	0.999	0.999
Muon momentum	–	0.95
LR decay scheduler	cosine	cosine

SF-AdamW and AMUSE

Hyperparameter	SF-AdamW	AMUSE
Learning rate	0.002	0.01
Batch size	256	256
Sequence length	512	512
Warmup steps	8000	6000
Weight decay	0.1	0.1
Gradient clipping	0.5	0.5
AMUSE β_1	–	0.6
AMUSE ρ	–	0.8
SF-AdamW β_1	0.9	–
AdamW β_2	0.9999	0.999
Muon momentum	–	0.95
LR decay scheduler	–	–

G.2.2. HYPERPARAMETERS FOR 720M PARAMETERS MODEL

Table 12: **Selected hyperparameters at the 720M scale.**

AdamW and Muon (D-Muon)

Hyperparameter	AdamW	Muon
Learning rate	0.001	0.001
Batch size	1984	1984
Sequence length	512	512
Warmup steps	2000	2000
Weight decay	0.1	0.1
Gradient clipping	0.1	0.1
AdamW β_1	0.9	0.9
AdamW β_2	0.999	0.99
Muon momentum	–	0.95
LR decay scheduler	cosine	cosine

SF-AdamW and AMUSE

Hyperparameter	SF-AdamW	AMUSE
Learning rate	0.001	0.01
Batch size	1984	1984
Sequence length	512	512
Warmup steps	8000	2000
Weight decay	0.1	0.1
Gradient clipping	0.1	0.1
AMUSE β_1	–	0.4
AMUSE ρ	–	0.8
SF-AdamW β_1	0.9	–
AdamW β_2	0.9999	0.999
Muon momentum	–	0.95
LR decay scheduler	–	–

G.2.3. HYPERPARAMETERS FOR 1B PARAMETERS MODEL

Table 13: **Selected hyperparameters at the 1B scale.**

AdamW and Muon (D-Muon)

Hyperparameter	AdamW	Muon
Learning rate	0.001	0.001
Batch size	2048	2048
Sequence length	512	512
Warmup steps	2000	2000
Weight decay	0.1	0.1
Gradient clipping	0.1	0.1
AdamW β_1	0.9	0.9
AdamW β_2	0.999	0.99
Muon momentum	–	0.95
LR decay scheduler	cosine	cosine

SF-AdamW and AMUSE

Hyperparameter	SF-AdamW	AMUSE
Learning rate	0.001	0.01
Batch size	2048	2048
Sequence length	512	512
Warmup steps	8000	2000
Weight decay	0.1	0.1
Gradient clipping	0.1	0.1
AMUSE β_1	–	0.4
AMUSE ρ	–	0.8
SF-AdamW β_1	0.9	–
AdamW β_2	0.9999	0.999
Muon momentum	–	0.95
LR decay scheduler	–	–

G.2.4. SEARCH GRID TABLE FOR AMUSE

For AdamW, SF-AdamW, and Muon (D-Muon), we use the best reported hyperparameters from the LLM optimizer benchmark of Semenov et al. [39]; their full search grids are provided in their work. We tune AMUSE separately and report our search grid below.

Table 14: **AMUSE hyperparameter search grid for LLM pretraining.** Selected values are shown in bold.

Scale	Hyperparameter	Search grid
124M	Learning rate	$0.002,\ 0.005,\ \mathbf{0.01},\ 0.02$
	Batch size	256
	Sequence length	512
	Warmup steps	$2000,\ 4000,\ \mathbf{6000}$
	Weight decay	$0.01,\ \mathbf{0.05},\ 0.1$
	Gradient clipping	$\mathbf{0.5}$
	AMUSE β_1	$0.4,\ \mathbf{0.6}$
	AMUSE ρ	$0.6,\ \mathbf{0.8}$
720M	Learning rate	$0.002,\ 0.005,\ \mathbf{0.01}, 0.02$
	Batch size	1984
	Sequence length	512
	Warmup steps	$\mathbf{2000}$
	Weight decay	$\mathbf{0.1}$
	Gradient clipping	$\mathbf{0.1}$
	AMUSE β_1	$\mathbf{0.4},\ 0.6$
	AMUSE ρ	$\mathbf{0.8}$
1B	Learning rate	$\mathbf{0.01}$
	Batch size	2048
	Sequence length	512
	Warmup steps	$\mathbf{2000}$
	Weight decay	$\mathbf{0.1}$
	Gradient clipping	$\mathbf{0.1}$
	AMUSE β_1	$\mathbf{0.4}$
	AMUSE ρ	$\mathbf{0.8}$
Fixed across sweeps: AdamW $\beta_2 = 0.999$, Muon momentum 0.95, no LR decay.		

Appendix H. Additional Experiments

H.1. Comparison with Additional Optimizer Baselines

We first compare AMUSE with other strong optimizer baselines for 124M LLaMA pretraining. In addition to AdamW, Muon, and SF-AdamW, we include AdEMAMix, ADOPT, Signum, Prodigy, MARS, Lion, Sophia, and SOAP, using the tuned hyperparameters reported by Semenov et al. [39]. As shown in Figure 16, AMUSE achieves the best validation perplexity among these optimizers, indicating that its advantage is not limited to the main baselines considered in the paper.

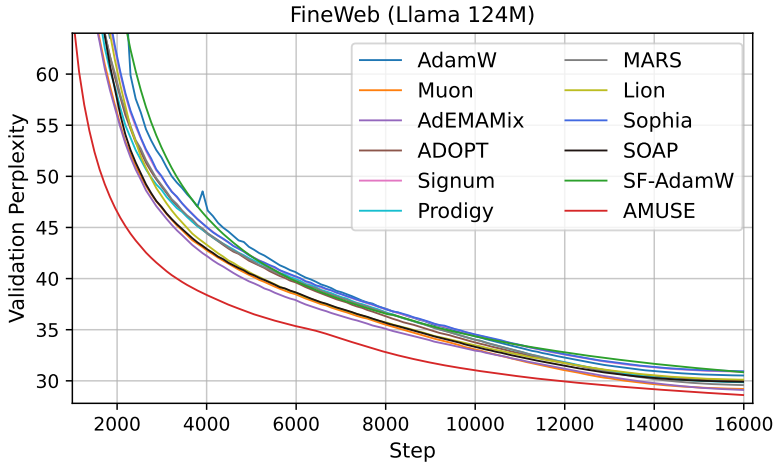


Figure 16: **Comparison with Other Baselines on Llama 124M model.**

H.2. Comparison with Muon with EWA and WSD

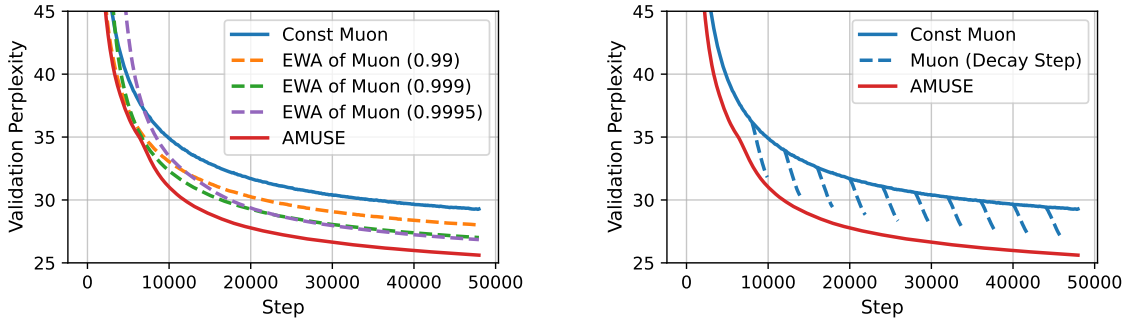
We further compare AMUSE with Muon equipped with exponential weight averaging (EWA) and warmup-stable-decay (WSD). These baselines test whether the gains of AMUSE can be matched by adding an external averaging mechanism or an explicit decay phase to constant-LR Muon.

For the 124M LLaMA experiment, we tune constant-LR Muon over learning rate, warmup steps, and weight decay, while keeping the momentum parameters the same as in the cosine-decay Muon baseline. The best constant-LR Muon setting uses learning rate 10^{-3}, weight decay 0.05, and 1000 warmup steps. We apply EWA with multiple averaging coefficients $(\alpha = 0.99, 0.999, 0.9995)$ to this tuned constant-LR Muon trajectory, and also run WSD-style decay phases for 2k iterations starting from multiple Muon checkpoints.

As shown in Figure 17, both EWA and WSD improve constant-LR Muon. EWA gives better validation perplexity with stronger averaging, and applying a 2k-step decay phase from Muon checkpoints substantially lowers perplexity. However, none of the EWA variants or WSD decay runs matches the performance of AMUSE, and AMUSE remains better throughout training. This suggests that AMUSE does not merely imitate post-hoc averaging or learning-rate decay, but instead maintains a more stable trajectory during training while preserving fast progress.

For ImageNet, we additionally evaluate Muon with EWA using the same EWA coefficient grid. In this experiment, we use the optimal hyperparameters of the cosine-scheduled Muon baseline and apply EWA. As shown in Figure 18, Muon+EWA does not outperform AMUSE for any averaging

coefficient. This suggests that AMUSE does not merely behave like an external weight average or a learning-rate decay, but instead maintains a more stable trajectory during training.



(a) Muon with multiple EWA coefficients

(b) Muon with WSD decay

Figure 17: **Comparison with Muon+EWA and Muon+WSD on 124M LLaMA pretraining.** Left: we apply exponential weight averaging (EWA) with multiple averaging coefficients to the tuned constant-LR Muon trajectory. Right: we apply 2k-step WSD-style decay phases starting from multiple Muon checkpoints. Both EWA and WSD improve constant-LR Muon, but none of these variants matches AMUSE.

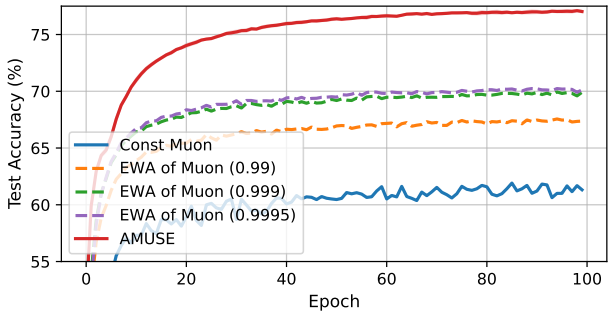


Figure 18: **Comparison with Muon with EWA on ImageNet.**

H.3. Longer Training Result

We further train AMUSE and the main baselines for a longer horizon, corresponding to the $3\times$ Chinchilla token budget. As shown in Figure 19, AMUSE continues to achieve the lowest validation perplexity on the 124M LLaMA model without additional hyperparameter tuning. Notably, AMUSE maintains its advantage even without an explicit learning-rate decay schedule, suggesting that its averaged trajectory remains effective beyond the standard training horizon.

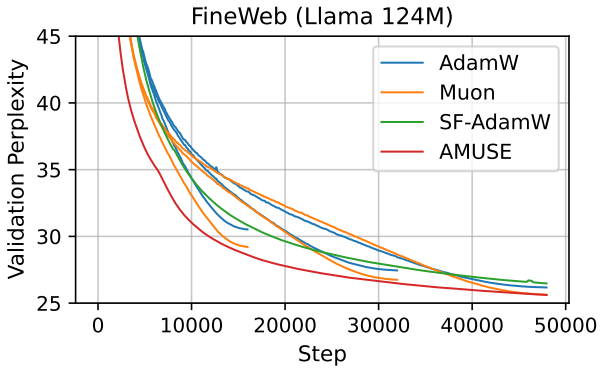


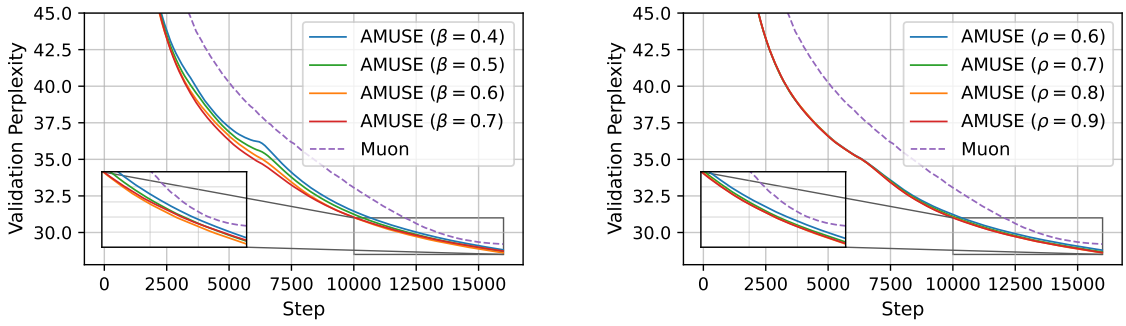
Figure 19: **Longer-horizon training results on 124M LLaMA pretraining.** We train AMUSE and the main baselines for the $3\times$ Chinchilla token budget. AMUSE achieves the lowest validation perplexity throughout the extended training horizon, even without using an explicit learning-rate decay schedule.

H.4. Hyperparameter Sensitivity.

Figure 20 and Table 15 show the sensitivity of AMUSE to β_1 and ρ in the 124M LLaMA setting. AMUSE shows only mild variation in validation perplexity across the tested ranges. Moreover, every swept configuration achieves lower perplexity than the tuned Muon baseline with cosine learning-rate decay.

Table 15: **Hyperparameter sensitivity of AMUSE on 124M LLaMA pretraining.**

Sweep	Hyperparameter value	Validation perplexity
β_1 sweep, $\rho = 0.8$	0.4	28.81
	0.5	28.70
	0.6	28.61
	0.7	28.73
ρ sweep, $\beta_1 = 0.6$	0.6	28.79
	0.7	28.68
	0.8	28.61
	0.9	28.63
Muon with cosine decay	–	29.21



(a) β_1 Sweep Results

(b) ρ Sweep Results

Figure 20: **Hyperparameter sensitivity varying β_1 and ρ on 124M LLaMA pretraining.**

H.5. Wall-clock Time Comparison

We compare the per-iteration wall-clock time of different optimizers in the 124M LLaMA pretraining setting. Following common practice in optimizer runtime comparisons, we report the average training iteration time, including forward pass, backward pass, and optimizer update. All optimizers are evaluated with the same model, sequence length, global batch size, precision, hardware, and distributed training configuration. AMUSE has a per-iteration cost similar to Muon, while SF-AdamW has a per-iteration cost similar to AdamW. The main runtime difference between these two groups comes from the Newton–Schulz iterations used to orthogonalize Muon updates, which are also used by AMUSE for Muon-updated layers.

Table 16: **Wall-clock time comparison on 124M LLaMA pretraining.** We report average iteration time after discarding warmup iterations. Each iteration processes 256×512 tokens. Relative time is normalized by AdamW.

Optimizer	Iter. time (s)	Relative time	Throughput (tokens/s)
AdamW	0.674	1.000	194,469
SF-AdamW	0.684	1.015	191,626
Muon	0.706	1.047	185,654
AMUSE	0.719	1.067	182,298

H.6. Ablation Studies

Comparison with SF-Muon. We compare AMUSE with fixed-β SF-Muon on 124M LLaMA pretraining. SF-Muon is tuned over the same hyperparameter grid, including multiple fixed β values. As shown in Figures 21 and 22, fixed-β SF-Muon does not outperform AMUSE in either setting. In particular, its improvement becomes limited in the later stage of training, suggesting that a fixed gradient-evaluation point cannot maintain the same early-speed and late-stability trade-off as AMUSE. By gradually increasing β_t, AMUSE continues to make progress later in training while preserving stable optimization.

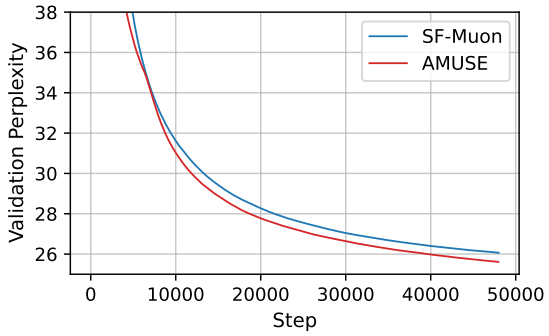


Figure 21: **Comparison between AMUSE and fixed-β SF-Muon on 124M LLaMA pretraining.** We compare AMUSE with SF-Muon using the best fixed interpolation value, $\beta = 0.95$. Fixed-β SF-Muon makes progress early in training but its improvement becomes limited in the later stage. In contrast, AMUSE continues to improve by gradually moving the gradient-evaluation point toward the averaged trajectory.

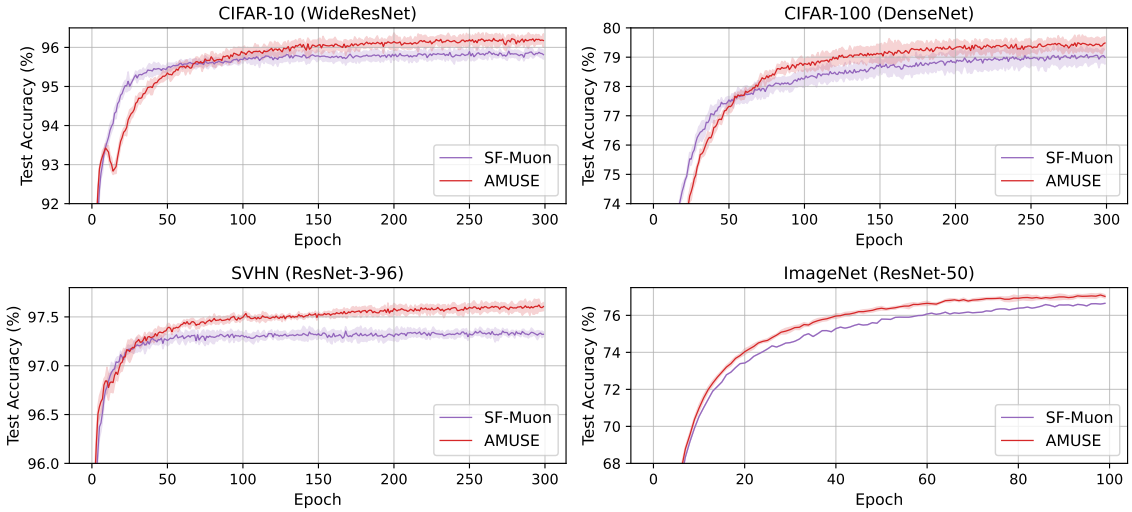
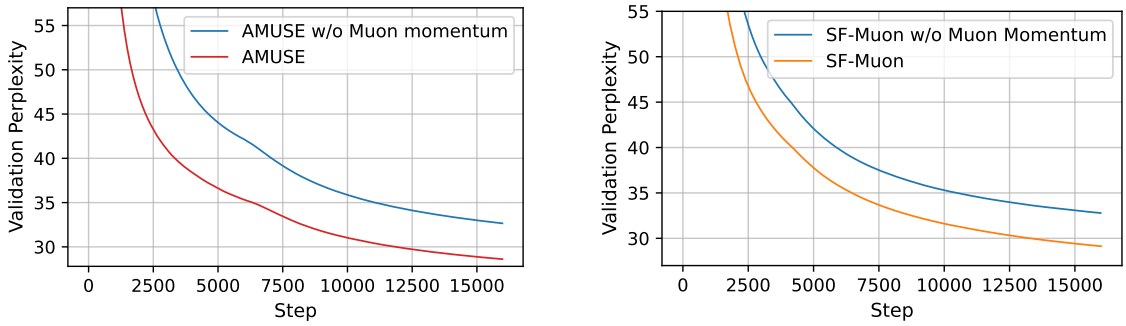


Figure 22: **Comparison between AMUSE and fixed-β SF-Muon on Image domain experiments.** We compare AMUSE with SF-Muon in image classification benchmarks. Runs are averaged over five random seeds, except that ImageNet with SF-Muon is reported with a single seed due to computational constraints

AMUSE without Muon Momentum. To examine the role of Muon momentum in AMUSE and SF-Muon, we remove the momentum buffer used before the orthogonalization step. As shown in Figure 23, removing Muon momentum causes a large performance drop for both methods. This indicates that momentum is crucial for Muon-based optimization: it smooths stochastic gradient noise before orthogonalization and accumulates a more reliable update direction, including useful bulk components. Without momentum, the orthogonalization step can amplify noisy components, resulting in less stable and less effective training.



(a) AMUSE w/o Muon momentum

(b) SF-Muon w/o Muon momentum

Figure 23: **Momentum ablation on 124M LLaMA pretraining.** We compare each method with and without Muon momentum, keeping all other hyperparameters fixed. Removing Muon momentum substantially degrades both AMUSE and SF-Muon, showing that momentum is essential for stable Muon-based optimization.

Can We Stop Increasing β_t After It Becomes Large? We further test whether AMUSE's improvement comes only from reaching a specific large interpolation value. To do so, we use the best hyperparameter for AMUSE, $\beta_1 = 0.6$ and $\rho = 0.8$, but cap β_t at the best fixed value used by SF-Muon, $\beta = 0.95$. That is, after β_t reaches 0.95, we keep it fixed instead of allowing it to continue increasing toward 1.

As shown in Figure 24, the clipped schedule performs worse than AMUSE, and the gap increases later in training. This indicates that AMUSE benefits not only from using a large β_t, but also from continuously shifting the evaluation point toward the averaged trajectory.

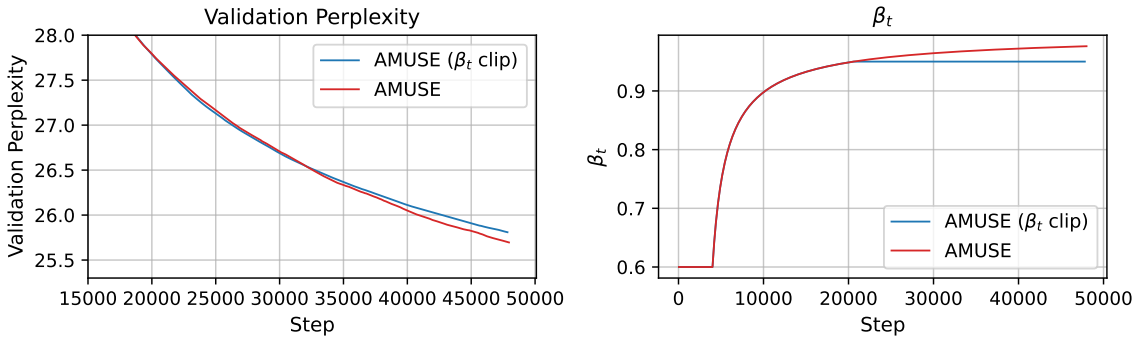


Figure 24: **Effect of stopping the increase of β_t in AMUSE.** (**Left**) Validation perplexity on 124M LLaMA pretraining. The clipped variant underperforms AMUSE, with the gap widening later in training. (**Right**) Evolution of β_t. While both methods reach similar large values, AMUSE continues to increase β_t, whereas the clipped variant saturates to 0.95.