AMUSE: Anytime Muon with Stable Gradient Evaluation

Jueun Kim[*]
KAIST
jueunkim@kaist.ac.kr

Baekrok Shin[*]
KAIST
br.shin@kaist.ac.kr

Jihun Yun
KRAFTON
jihuny@krafton.com

Beomhan Baek[†]
UCLA
beomhanbaek@ucla.edu

Minhak Song[†]
Stanford University
minhaks@stanford.edu

Chulhee Yun
KAIST
chulhee.yun@kaist.ac.kr

https://github.com/kjeiun/amuse

Abstract

Modern deep learning commonly relies on AdamW with prescribed learning rate schedules, but recent works challenge both components: Schedule-Free optimization removes explicit schedules via iterate averaging, and Muon improves the update geometry by orthogonalizing momentum for matrix parameters. Despite Muon's strong empirical performance, its underlying mechanism remains partially understood. We study Muon through the river-valley loss landscape, where useful training progress occurs along a flat, low-curvature bulk subspace (the river), while high-curvature dominant directions form steep valley walls that induce oscillations. We empirically show that while Muon's orthogonalization accelerates river progress by increasing the bulk component, it also amplifies dominant-direction noise, causing oscillatory trajectories. Building on this, we propose Anytime MUon with Stable gradient Evaluation (AMUSE), which integrates Muon's rapid bulk progress with the stabilizing effect of Schedule-Free averaging. AMUSE uses a time-varying interpolation coefficient that initially evaluates gradients near the fast Muon sequence for rapid adaptation, then gradually shifts toward the stable averaged sequence to suppress 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. For many modern architectures, especially Transformer-based language models, the standard recipe has long combined the Adam(W) optimizer (Kingma and Ba, 2015; Loshchilov and Hutter, 2019) with warmup and a prescribed learning rate decay schedule such as cosine decay (Loshchilov and Hutter, 2017).

Recent work challenges both components of this recipe. Schedule-Free AdamW (Defazio et al., 2024) removes the need for explicit learning rate schedules by evaluating gradients at interpolated points between the current and averaged iterates, while using the averaged iterate for inference. It has shown strong empirical performance relative to schedule-based baselines, including winning the Self-Tuning track of the 2024 AlgoPerf Challenge (Dahl et al., 2023; Kasimbeg et al., 2025).

[*]Authors contributed equally to this paper.
[†]Work done during internships at KRAFTON.

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

A complementary advance changes the underlying geometry of the update itself. Muon (Jordan et al., 2024) exploits matrix structure by applying momentum to matrix-valued parameters and orthogonalizing the resulting update direction. Muon and its variants have been reported to outperform AdamW in language model pretraining (Liu et al., 2025a), and have recently been adopted in large-scale foundation-model pretraining, including Kimi-K2/K3 (Kimi Team, 2025, 2026), GLM-5 (GLM-5 Team, 2026), DeepSeek-V4/V4.1-Flash (DeepSeek-AI, 2026a,b), MAI-Base-2 (Microsoft AI, 2026), and Qwen3.8-Flash-Next (Qwen Team, 2026). Despite this empirical success, *why orthogonalized momentum improves optimization* remains only partially understood.

We take this question as the first step toward optimizer design, and analyze Muon from the perspective of loss landscape geometry. Deep learning optimization takes place in high-dimensional parameter spaces whose loss landscapes are highly anisotropic. Empirically, the Hessian spectrum often contains a small number of large outlier eigenvalues and a much larger bulk of relatively small eigenvalues. This spectral structure induces a local decomposition of the parameter space into a low-dimensional, high-curvature *dominant subspace* and a high-dimensional, low-curvature *bulk subspace* (Ghorbani et al., 2019; Papyan, 2018, 2019, 2020; Sagun et al., 2016, 2017).

Recent work casts this anisotropy as the *river-valley landscape* (Song et al., 2025a; Wen et al., 2025): the dominant subspace forms steep valley walls, while the bulk subspace forms a relatively flat river. In this picture, training progress occurs mainly along the river, whereas dominant subspace components cause valley-wall oscillations. This suggests two core principles for optimizer design: *(i)* take large, consistent steps along the river and *(ii)* suppress unnecessary updates across the valley.

We find that Muon naturally promotes the first part of this principle. By orthogonalizing the momentum matrix, Muon prevents the update from being dominated by a few large singular components, producing a more balanced matrix-valued step. Empirically, Muon updates contain substantially larger bulk components than SGD or Adam. Moreover, orthogonalization itself explicitly increases this relative bulk component. This provides a geometric explanation for Muon's efficiency: *orthogonalized momentum accelerates progress along the river directions*.

However, this benefit comes with a tradeoff. Because orthogonalization is not selective, it can amplify noisy components as well as useful ones. In particular, small noise aligned with high-curvature dominant directions can induce oscillations across the valley walls after orthogonalization, leading to less stable trajectories (He et al., 2025; Qi et al., 2026; Shen et al., 2026; Zhang et al., 2026a).

Schedule-Free (SF) optimization offers a complementary stabilizing mechanism by evaluating gradients at an interpolation between the fast optimizer sequence and an averaged inference sequence. Song et al. (2025b) show that this mechanism can keep the inference trajectory close to the river without requiring learning rate decay. We find this particularly beneficial for Muon: evaluating gradients closer to the averaged sequence reduces the high-curvature component before orthogonalization, thereby suppressing oscillations between the valley walls and stabilizing the resulting trajectory.

Motivated by this complementarity, we propose **AMUSE** (Anytime MUon with Stable gradient Evaluation). AMUSE combines Muon's orthogonalized matrix updates with the SF formulation, using a *time-varying* interpolation coefficient. AMUSE initially evaluates gradients near the fast Muon sequence for rapid adaptation, then gradually shifts toward the averaged sequence, where gradients are less dominated by high-curvature components. Because this schedule is independent of the training horizon and requires no learning rate decay, AMUSE naturally supports *anytime* training.

We evaluate AMUSE across a diverse set of benchmarks spanning image classification, segmentation, and large language model pretraining. The results show that AMUSE consistently improves the performance-iteration Pareto frontier over (Schedule-Free) AdamW and Muon. Notably, AMUSE reaches Muon's final performance with substantially fewer training steps across diverse settings. In particular, it requires $1.51\times$, $1.34\times$, and $2.57\times$ fewer steps on 720M Llama pretraining, ImageNet with ResNet-50, and ImageNet with ViT fine-tuning, respectively.

2 Related Work

2.1 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 (Ghorbani et al.,

2019; Papyan, 2018, 2019, 2020; Sagun et al., 2016, 2017; Yao et al., 2020). 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 (Belloni et al., 2025; Cohen et al., 2025; Deng et al., 2026; Song et al., 2025a; Wang et al., 2025, 2024; Wen et al., 2025; Zhou et al., 2025).

Wen et al. (2025) 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. (2025) show that a similar geometry also arises in convolutional networks trained on image classification tasks. Song et al. (2025a) 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 (Liu et al., 2025b; Luo et al., 2026; Zhang et al., 2026b; Zhu et al., 2026).

2.2 Schedule-Free Optimizer

Defazio et al. (2024) introduce the schedule-free (SF) optimizer as an interpolation between Polyak–Ruppert (PR) averaging (Polyak and Juditsky, 1992; Ruppert, 1988) and primal averaging (Nesterov, 2009; Tao et al., 2018). Its update is given by

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

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 (Dahl et al., 2023).

Song et al. (2025b) 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.

2.3 Muon Optimizer

Muon (Jordan et al., 2024) 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:

$$\begin{aligned} \boldsymbol{M}_t &= \mu \boldsymbol{M}_{t-1} + (1-\mu)\boldsymbol{G}_t, \\ \boldsymbol{W}_{t+1} &= \boldsymbol{W}_t - \eta \mathcal{T}(\boldsymbol{M}_t), \end{aligned} \tag{2}$$

where η is the learning rate, μ is the momentum coefficient, and $\mathcal{T}$ denotes an orthogonalization operator, which is approximated using a Newton-Schulz iteration (Bernstein and Newhouse, 2024) 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. (2025) show that Muon can be advantageous when neural network Hessians exhibit low-rank or structured spectral geometry, while Ma et al. (2026) 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 (Kang et al., 2026), improving minority-group performance in imbalanced settings (Vasudeva et al., 2026). However, this same mechanism can also strengthen noisy components, potentially leading to less stable dynamics (Qi et al., 2026; Shen et al., 2026; Zhang et al., 2026a).

3 A River-Valley Analysis of Muon and Schedule-Free Dynamics

In this section, we analyze Muon and Schedule-Free dynamics in controlled deep learning settings where the leading Hessian eigenspaces can be computed directly. We first define the dominant and bulk subspaces used in our measurements. Section 3.1 then shows that Muon produces more bulk-oriented updates than SGD or AdamW, but that orthogonalization can also amplify small components in the dominant subspace, inducing oscillations across the valley walls. Section 3.2 shows that Schedule-Free averaging mitigates this effect by evaluating gradients at points with smaller dominant components before orthogonalization.

Dominant & Bulk Decomposition. Following Song et al. (2025a), we decompose parameter-space directions using the Hessian eigenspaces of the loss $\mathcal{L} : \mathbb{R}^d \to \mathbb{R}$.

Definition 3.1 (Dominant and bulk subspaces)**.** Let $\nabla^2\mathcal{L}(\theta) \in \mathbb{R}^{d\times d}$ have eigenvalues $\lambda_1 \geq \cdots \geq \lambda_d$ with corresponding eigenvectors $\boldsymbol{u}_1(\theta), \ldots, \boldsymbol{u}_d(\theta)$. The top-$k$ dominant subspace is $\mathcal{S}_k(\theta) := \mathrm{span}\{\boldsymbol{u}_1(\theta), \ldots, \boldsymbol{u}_k(\theta)\}$, and its orthogonal complement $\mathcal{S}_k^{\perp}(\theta)$ is the bulk subspace.

Definition 3.2 (Subspace projections)**.** We define the projection matrix onto $\mathcal{S}_k(\theta)$ as $\boldsymbol{P}_k(\theta) := \sum_{i=1}^{k} \boldsymbol{u}_i(\theta)\boldsymbol{u}_i(\theta)^\top$, and the projection matrix onto $\mathcal{S}_k^{\perp}(\theta)$ as $\boldsymbol{P}_k^{\perp}(\theta) := \boldsymbol{I} - \boldsymbol{P}_k(\theta)$. For any nonzero vector $\boldsymbol{v} \in \mathbb{R}^d$, we define

$$r^{\mathrm{dom}}(\boldsymbol{v};\theta) = \frac{\|\boldsymbol{P}_k(\theta)\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_2}, \qquad r^{\mathrm{bulk}}(\boldsymbol{v};\theta) = \frac{\|\boldsymbol{P}_k^{\perp}(\theta)\boldsymbol{v}\|_2}{\|\boldsymbol{v}\|_2}. \tag{3}$$

These quantities measure how strongly $\boldsymbol{v}$ is aligned with the dominant and bulk subspaces, respectively. Since $\boldsymbol{P}_k(\theta)$ and $\boldsymbol{P}_k^{\perp}(\theta)$ are orthogonal projections, the ratios satisfy $(r^{\mathrm{dom}})^2 + (r^{\mathrm{bulk}})^2 = 1$; hence, we report r^{dom} in most measurements. Following Song et al. (2025a), we set k to the number of classes, motivated by prior observations that classification Hessians exhibit exactly this many outlier eigenvalues (Papyan, 2018, 2019).

Before analyzing Muon's update decomposition, we verify that the dominant and bulk subspaces play distinct optimization roles in our controlled settings. Following Zhou et al. (2025), we independently scale the dominant and bulk components of each Muon update. As shown in Appendix B.1, amplifying the *bulk component* accelerates training, whereas amplifying the *dominant component* can destabilize optimization. This supports the river-valley interpretation in our setting and motivates the dominant/bulk update-ratio analysis below.

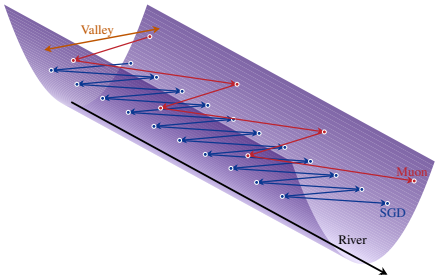
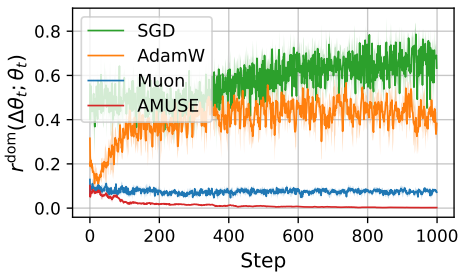


Figure 1: Illustration of a river-valley landscape. SGD oscillates across the valley walls and progresses slowly along the river, while Muon advances faster but remains oscillatory.

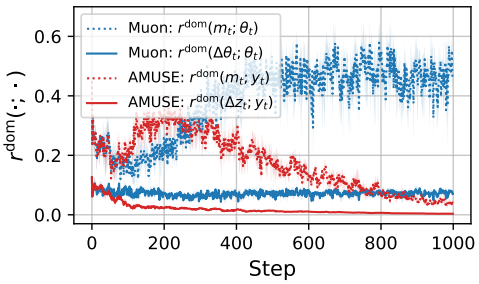
3.1 Orthogonalization Makes Muon Updates More Bulk-Oriented

Muon applies an orthogonalization operator $\mathcal{T}(\cdot)$ to the momentum matrix $\boldsymbol{M}_t$. Although this operation acts on the singular-vector geometry of each matrix, we measure its effect in the Hessian eigenspace by projecting the flattened update onto the dominant and bulk subspaces. For notation simplicity, we slightly abuse notation and write $\mathcal{T}(\boldsymbol{m}_t) := \mathrm{vec}(\mathcal{T}(\boldsymbol{M}_t))$.

To examine how orthogonalization redistributes optimizer updates across the Hessian eigenspace, we measure the dominant component ratio throughout training on a 5k-sample subset of MNIST using a



(a) Muon vs. SGD/AdamW



(b) Momentum $\boldsymbol{m}_t$ vs. Update $\Delta\theta_t$ $(= \mathcal{T}(\boldsymbol{m}_t))$

Figure 2: Comparison of dominant component ratios. Evaluated on a 5k MNIST subset using a 3-layer MLP.(**a**) Muon consistently produces smaller dominant updates than SGD/AdamW, and AMUSE further suppresses the dominant component. (**b**) 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. Additional experiments on other datasets and architectures are provided in Appendix B.3.

3-layer MLP. Experimental details are provided in Appendix B.2. Figure 2a compares the dominant ratios of optimizer updates. Muon produces much smaller dominant-subspace components than SGD or AdamW, indicating that its updates are less aligned with the high-curvature directions and more oriented toward the bulk subspace. The figure also includes AMUSE, introduced in Section 4, as a preview of the stabilized variant of Muon; its updates have even smaller dominant ratios.

Figure 2b isolates the role of orthogonalization by comparing the raw momentum $\boldsymbol{m}_t$ with the post-orthogonalized update $\Delta\theta_t(= \mathcal{T}(\boldsymbol{m}_t))$. For Muon, orthogonalization sharply reduces the dominant ratio relative to the raw momentum, showing that the orthogonalization step itself shifts the update away from dominant directions and toward the bulk subspace. This provides a geometric explanation for Muon's effectiveness: by increasing the relative bulk component, *orthogonalized momentum promotes faster progress along the river directions*, where long-term learning takes place.

However, this normalization process prevents dominant components from dying out. Even a small residual dominant component can induce non-negligible motion along high-curvature directions after normalization, leading to oscillations across the valley walls. This behavior resembles normalized-update dynamics, which can oscillate even on simple quadratic objectives (Arora et al., 2022); see Appendix A for an example.

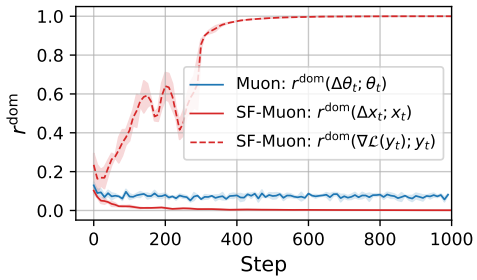
3.2 Schedule-Free Stabilizes Muon Trajectories

The previous subsection shows that Muon produces large bulk movement, but its orthogonalized updates may also amplify valley-wall oscillations. Prior work has shown that, with an appropriate choice of β, the averaged trajectory $\boldsymbol{x}_t$ of SF-AdamW follows the river (Song et al., 2025b), suggesting that *the averaging mechanism of the SF optimizer may mitigate this instability*. Motivated by this observation, we use Muon as the base optimizer under the SF optimizer (we refer to this variant as *SF-Muon*) and analyze the resulting trajectories.

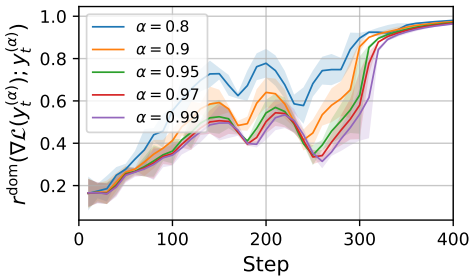
Taking Muon as the base optimizer, we obtain the following SF formulation for matrix-valued parameters:

$$\begin{aligned} \boldsymbol{Y}_t &= (1-\beta)\boldsymbol{Z}_t + \beta\boldsymbol{X}_t, \\ \boldsymbol{M}_t &= \mu\boldsymbol{M}_{t-1} + (1-\mu)\nabla\mathcal{L}(\boldsymbol{Y}_t), \\ \boldsymbol{Z}_{t+1} &= \boldsymbol{Z}_t - \eta\mathcal{T}(\boldsymbol{M}_t), \\ \boldsymbol{X}_{t+1} &= (1-c_{t+1})\,\boldsymbol{X}_t + c_{t+1}\boldsymbol{Z}_{t+1}. \end{aligned} \tag{4}$$

Here, $\boldsymbol{Z}_t$ is the base sequence updated by Muon, $\boldsymbol{Y}_t$ is the gradient-evaluation point, and $\boldsymbol{X}_t$ is the averaged sequence used as the final model parameter, with $c_{t+1} = 1/(t+1)$. The coefficient β determines where the gradient is evaluated between the fast-moving base sequence $\boldsymbol{Z}_t$ and the averaged sequence $\boldsymbol{X}_t$. Although Eq. (4) is written for a single Muon-updated matrix block, not all parameters are updated by Muon. Non-matrix parameters are updated using their corresponding SF base optimizer, such as SF-SGD or SF-AdamW.



(a) Comparison of dominant ratios



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

Figure 3: Comparison of dominant component ratios. Settings as in Figure 2. (**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**) 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. Additional datasets and model architectures show consistent trends; see Appendix B.3.

To understand how SF interacts with Muon, we measure subspace ratios in the full parameter space. Using the notation of Eq. (1), let $\boldsymbol{x}_t, \boldsymbol{y}_t, \boldsymbol{z}_t \in \mathbb{R}^d$ denote the averaged sequence, gradient-evaluation point, and base sequence, respectively, obtained by flattening and concatenating all trainable parameters. With $\Delta\boldsymbol{x}_t := \boldsymbol{x}_{t+1} - \boldsymbol{x}_t$, we compute $r^{\text{dom}}(\Delta\boldsymbol{x}_t;\boldsymbol{x}_t)$ as well as the ratio for the gradient evaluated at $\boldsymbol{y}_t$, $r^{\text{dom}}(\nabla\mathcal{L}(\boldsymbol{y}_t);\boldsymbol{y}_t)$.

The results in Figure 3a show that the update of $\boldsymbol{x}_t$ is predominantly bulk-oriented, indicating that the trajectory primarily follows the river direction. In contrast, the gradient evaluated at $\boldsymbol{y}_t$ still retains a substantial dominant component, which intensifies as training progresses. This suggests that while the averaging mechanism of SF stabilizes the $\boldsymbol{x}_t$ trajectory, the gradient passed to Muon may still be computed at a noisy valley-wall location. This raises a natural question:

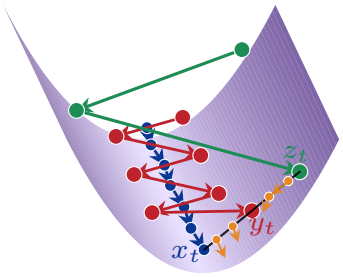


Figure 4: Schedule-free iterates in a river-valley landscape. Orange arrows indicate negative gradients computed at each $\boldsymbol{y}_t^{(\alpha)}$, exhibiting reduced dominant components when evaluated closer to the river.

Can we further reduce the dominant component before orthogonalization by evaluating gradients at a point closer to the river?

To test this hypothesis, we use $\boldsymbol{x}_t$ as a proxy for a point on the river and evaluate gradients at virtual interpolation points $\boldsymbol{y}_t^{(\alpha)} = (1-\alpha)\boldsymbol{z}_t + \alpha\boldsymbol{x}_t$ for varying $\alpha \in [0,1]$, while keeping the actual training trajectory fixed by using a fixed β during training (see Figure 4 for an illustration). At each $\boldsymbol{y}_t^{(\alpha)}$, we compute the gradient and measure its dominant ratio using Eq. (3).

Figure 3b shows that as α increases, the gradient becomes increasingly bulk-oriented. Thus, evaluating closer to $\boldsymbol{x}_t$ suppresses valley-wall contributions before the Muon orthogonalization step, yielding a more stable update. Consistent with this mechanism, AMUSE—the time-varying SF-Muon variant introduced in Section 4—suppresses dominant components both before and after orthogonalization (Figure 2b). In the language model experiment on 124M Llama pretraining (setup in Section 4), we observe a related effect at the trajectory level: increasing β yields larger averaged-sequence updates $\|\Delta\boldsymbol{x}_t\|$, indicating faster progress along the river direction (Figure 5, right).

This observation suggests evaluating gradients near the averaged sequence $\boldsymbol{x}_t$ by using a large fixed β. However, using a large constant β creates an early–late trade-off: early in training, $\boldsymbol{x}_t$ may lag behind the rapidly adapting base sequence $\boldsymbol{z}_t$, so evaluating too close to $\boldsymbol{x}_t$ can slow initial progress toward the river. Conversely, a smaller β preserves early adaptation but provides weaker late-stage stabilization.

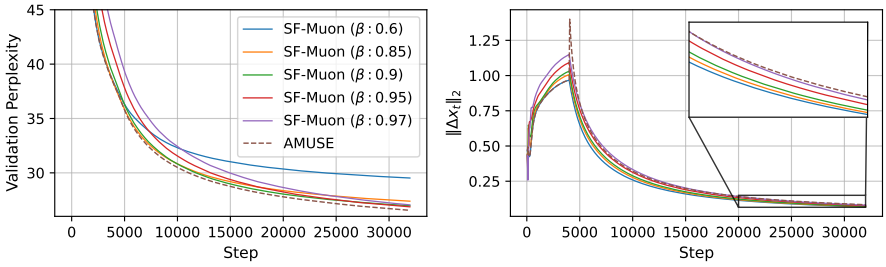


Figure 5: Comparison of fixed-β SF-Muon with different β values and AMUSE in the 124M Llama pretraining on FineWeb. Solid lines show fixed-β SF-Muon, and dashed lines show AMUSE with $\beta_{\text{init}} = 0.6$ and $\rho = 0.8$. We report validation perplexity **(left)** and the update norm $\|\Delta \boldsymbol{x}_t\|$ **(right)**.

4 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 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. (4), 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 $\beta_t = \beta_{\text{init}}$. For $t \geq T_0$, we set:

$$\beta_t = 1 - \left(\frac{c_{t+1}(1 - c_{T_0+1})}{c_{T_0+1}(1 - c_{t+1})} \right)^{\rho} (1 - \beta_{\text{init}}), \tag{5}$$

where $\rho \in [0, 1]$ controls how quickly β_t increases. Under the convention $c_{t+1} = 1/(t+1)$, Eq. (5) simplifies to

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

When $\rho = 0$, AMUSE reduces to fixed-β SF-Muon with $\beta_t = \beta_{\text{init}}$. Larger values of ρ move the gradient-evaluation point toward $\boldsymbol{x}_t$ more rapidly. We defer the detailed rationale and derivation of Eq. (5) to Appendix C.3.

Notably, *this time-varying β_t is independent of the total number of training steps*, gradually approaching 1 as training progresses. Furthermore, like the standard SF method, AMUSE does not require any learning rate schedule. Algorithm 1 in Appendix C.1 summarizes the complete AMUSE procedure.

To verify whether AMUSE behaves as intended, we compare it with fixed-β SF-Muon in Figure 5. We train a 124M Llama model on FineWeb, keeping all hyperparameters fixed except the interpolation coefficient: SF-Muon uses different constant β values, while AMUSE uses the time-varying β_t from Eq. (5). A small fixed β achieves better early performance but degrades late-stage performance, whereas a large fixed β improves performance in the later stages at the cost of higher early perplexity. AMUSE balances these effects by gradually increasing β_t, yielding a Pareto-optimal perplexity curve while maintaining a larger $\|\Delta \boldsymbol{x}_t\|$, which indicates faster progress along useful directions. Fixed-β SF-Muon variants also underperform AMUSE throughout training (Appendix F.7). We further evaluate the design of the β_t schedule in Eq. (5) by comparing it with tuned linear and cosine alternatives. The proposed schedule achieves lower validation perplexity (Table 25 in Appendix F.7).

We also compare AMUSE with post-hoc weight averaging methods, including Stochastic Weight Averaging (Izmailov et al., 2018), Latest Weight Averaging (Kaddour, 2022), and Pre-trained Model Average (Li et al., 2025a), in Appendix F.3. These methods average an existing trajectory, whereas AMUSE incorporates averaged iterates into gradient evaluation, modifying subsequent updates.

4.1 AMUSE Follows the River

Following Song et al. (2025b), we test whether AMUSE already follows the river using two post-hoc evaluations: Exponential Weight Averaging (EWA) and learning rate decay. If the optimizer still oscillates across the valley walls, applying either to the $\boldsymbol{x}_t$ trajectory should substantially reduce the loss by moving it closer to the river; if it already tracks the river, they should offer little benefit.

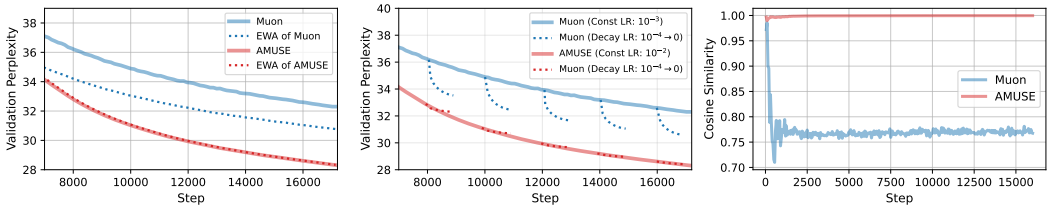


Figure 6: Comparison between constant learning rate Muon and AMUSE in the 124M Llama setting. **The left panel** shows the effect of EWA, where solid lines represent the original training trajectories and dotted lines represent their corresponding EWA trajectories. **The middle panel** shows the effect of learning rate decay, where after warmup we linearly decay the learning rate from 10^{-4} to 0 at selected iterations using the 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.

As shown in Figure 6, 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 $\boldsymbol{x}_t$.

5 Experiments

We evaluate AMUSE across a diverse set of benchmarks covering standard image classification, image segmentation, and large language model (LLM) pretraining. For the standard image domain experiments, we primarily follow the setup of Defazio et al. (2024). For LLM pretraining, we follow Semenov et al. (2026) and adopt their Llama-like decoder-only Transformer setup trained on FineWeb-100B (Penedo et al., 2024). In particular, we consider 124M, 720M, and 1.3B model sizes.

As baselines, we use SGD with momentum and SF-SGD for CIFAR-10, CIFAR-100, SVHN, ImageNet-1k, and ISIC 2018. AdamW and SF-AdamW are used for the remaining tasks. Across all settings, we additionally include Muon as a shared baseline. For SGD, AdamW and Muon, we employ cosine learning rate schedules across all experiments.

5.1 Image Domain Experiments

Experimental Setup. We evaluate AMUSE on image classification benchmarks on diverse datasets and architectures: Wide-ResNet-16-8 on CIFAR-10 (Krizhevsky et al., 2009; Zagoruyko and Komodakis, 2016), DenseNet on CIFAR-100 (Huang et al., 2017), ResNet-3-96 on SVHN (He et al., 2016; Netzer et al., 2011), and ResNet-50 on ImageNet-1k (Russakovsky et al., 2015). We further evaluate two additional settings: U-Net on ISIC 2018 for image segmentation (Codella et al., 2019; Ronneberger et al., 2015), and MAE fine-tuning with a ViT-B/16 on ImageNet (He et al., 2022). For existing baselines, we use the tuned hyperparameters from Defazio et al. (2024) when available. For Muon and AMUSE, whose settings are not provided, we fairly tune the learning rate, weight decay, and optimizer-specific hyperparameters; details are provided in Appendix D.1.

Experimental Results. As illustrated in Figure 7, AMUSE consistently achieves the best performance among the compared optimizers across all evaluated datasets, tasks, and architectures. It demonstrates *superior anytime performance*: while vanilla Muon requires the full training horizon to reach competitive performance, AMUSE exhibits rapid early progress and maintains the highest accuracy throughout training. Notably, AMUSE remains highly effective for MAE fine-tuning even when initialized from an AdamW-pretrained model. In contrast, vanilla Muon struggles in this setting, consistent with recent reports of optimizer mismatch when fine-tuning AdamW-pretrained models with Muon (Liu et al., 2025a,c; Qu et al., 2026). This suggests that AMUSE can be applied effectively to models pretrained with conventional optimizers.

5.2 Large Language Model Pretraining

Experimental Setup. We follow the Llama-style Transformer setup of Semenov et al. (2026), using tied input/output embeddings, SwiGLU, RMSNorm, and RoPE. We train 124M, 720M, and 1.3B models on FineWeb-100B (Penedo et al., 2024) with sequence length 512 and batch sizes 256, 1984,

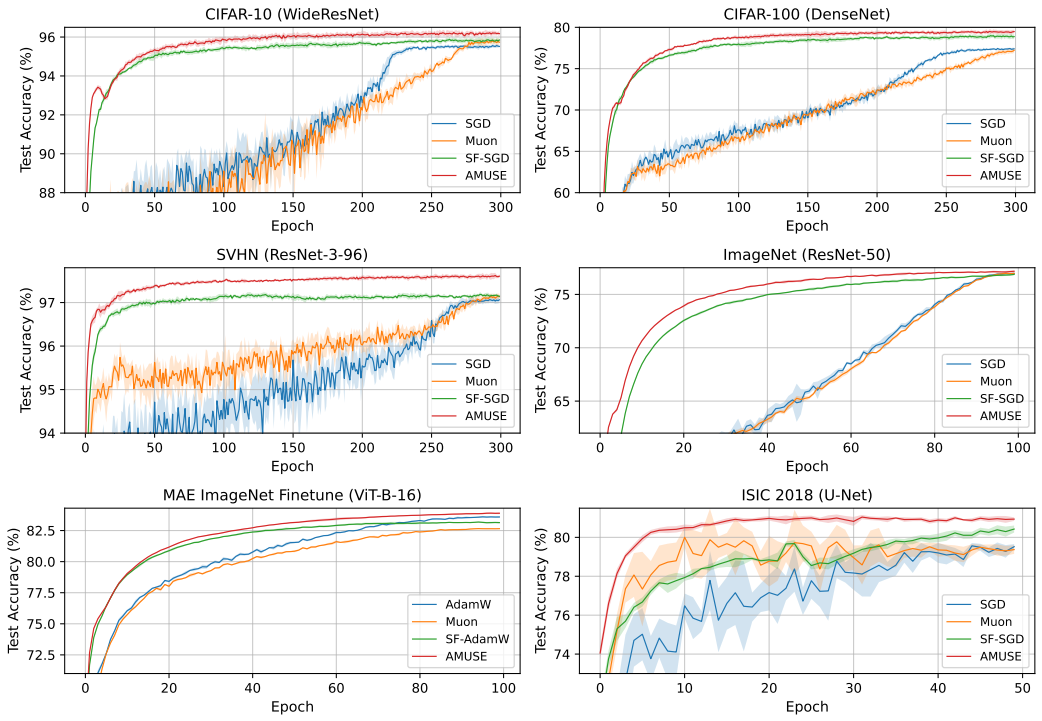


Figure 7: Test accuracy across image domain experiments. Averaged over five random seeds.

and 2048, respectively. The 124M and 720M models are trained for 16k iterations (≈2.1B and ≈16.3B tokens, respectively), while the 1.3B model is trained for 24.8k iterations (≈26B tokens). For baseline optimizers, we use the highly tuned hyperparameters from Semenov et al. (2026), including learning rate, weight decay, scheduler, warmup steps, and momentum. At the 124M scale, we additionally conduct an expanded hyperparameter search for Muon, starting from the reported best configuration. For AMUSE, we tune the corresponding hyperparameters under the same experimental setup, along with $\beta_{\text{init}} \in \{0.4, 0.6\}$ and $\rho \in \{0.6, 0.8\}$. See Appendix E.2 for more details.

We additionally evaluate the final 720M checkpoints after 16k pretraining iterations on ARC-Easy, ARC-Challenge, BoolQ, HellaSwag, PIQA, WinoGrande, and OpenBookQA in a zero-shot setting.

Experimental Results. The results in Figure 8 show that AMUSE consistently improves LLM pretraining. Across all Llama model scales, it achieves lower validation perplexity than all baselines throughout the entire training horizon. Extended training results across 124M and 720M model scales show that AMUSE remains effective beyond the standard training horizon (Figure 20 in Appendix F.4). We also compare AMUSE with other recent optimizers, demonstrating that it consistently improves the performance-iteration Pareto frontier (Figure 17 in Appendix F.1). While SF-AdamW struggles with the large batch sizes typically required for modern LLMs (Morwani et al., 2025), AMUSE scales effectively, maintaining strong performance on the 720M and 1.3B models trained with large batches. AMUSE's advantage persists across five random seeds at 124M; see Table 26 in Appendix F.8.

As shown in Table 1, AMUSE achieves the highest accuracy across all seven downstream tasks, with an average accuracy of 46.3% compared with 45.3% for Muon. These results indicate that AMUSE's improvements in validation perplexity also translate into improved downstream performance.

We further test whether adding exponential weight averaging or a decay phase to constant learning rate Muon recovers AMUSE's performance. As detailed in Appendix F.2, while these variants improve Muon, they still fall short of AMUSE. This suggests AMUSE inherently maintains a stable trajectory, rather than merely mimicking weight averaging or learning rate decay. AMUSE retains its performance advantage in wall-clock time, with runtime comparable to Muon (Appendix F.6).

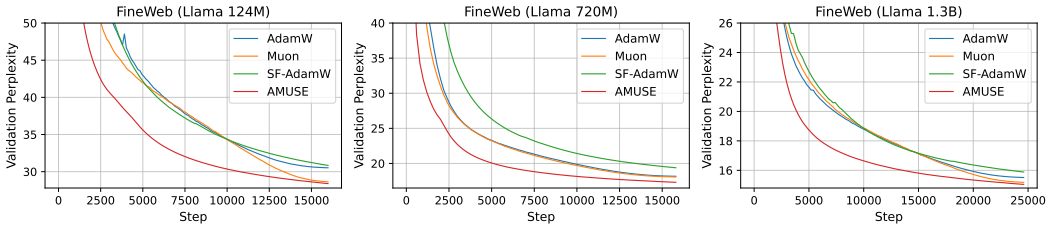


Figure 8: Validation perplexity on FineWeb pretraining across Llama model scales.

Table 1: Zero-shot downstream accuracy of 720M models after 16k pretraining iterations.

Optimizer	ARC-E	ARC-C	BoolQ	HS	PIQA	WG	OBQA	Avg.
AdamW	0.534	0.228	0.587	0.372	0.700	0.535	0.172	0.447
Muon	0.542	0.235	0.567	0.376	0.717	0.536	0.198	0.453
SF-Muon	0.542	0.224	0.576	0.384	0.706	0.511	0.206	0.450
AMUSE	**0.547**	**0.240**	**0.588**	**0.388**	**0.720**	**0.539**	**0.220**	**0.463**

5.3 Hyperparameter Sensitivity

In all AMUSE experiments, we fix the Muon momentum μ to 0.95 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 β_{init} and ρ in the Llama 124M setting in Appendix F.5, demonstrating its robustness.

5.4 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. For Muon updates, we use Nesterov momentum rather than the Polyak momentum formulation in Eqs. (2) and (4), following the practical Muon implementation (Jordan et al., 2024).

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. For other implementation details, see Appendix C.4.

6 Conclusion

We introduce AMUSE, an optimization algorithm designed to stabilize Muon using the Schedule-Free (SF) framework 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 the SF framework, a principled approach that controls where the gradient is evaluated and stabilizes training by keeping the optimization trajectory close to the river. With a carefully designed schedule for the interpolation parameter β_t, AMUSE outperforms conventional learning-rate-scheduled Adam(W) or Muon 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. Further evaluation at larger model scales and over longer training horizons is also needed.

Acknowledgments and Disclosure of Funding

This work was supported by an Institute of Information & Communications Technology Planning & Evaluation (IITP) grant (No. RS-2024-00457882 National AI Research Lab Project, 20%) funded by the Korean government (MSIT), two National Research Foundation of Korea (NRF) grants (No. RS-2023-00211352 Development of Efficient Stochastic Optimization Algorithms via Analyzing Without-replacement Sampling Methods, 40%; No. RS-2026-25612350 Development of an End-to-End Design and Validation Framework for Mathematically Principled Next-Generation AI Architectures and Learning Algorithms, 30%) funded by the Korean government (MSIT), and a KRAFTON research grant (10%).

References

Anuj Apte, Pranav Deshpande, Niraj Kumar, Shouvanik Chakrabarti, and Junhyung Lyle Kim. Anytime training with schedule-free spectral optimization. *arXiv preprint arXiv:2605.23061*, 2026.

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.

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.

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.

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.

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.

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.

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.

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.

DeepSeek-AI. DeepSeek-V4: Towards highly efficient million-token context intelligence, 2026a.

DeepSeek-AI. DeepSeek-V4.1-Flash: Pushing the limits of KV cache compression. *arXiv preprint arXiv:2609.19969*, 2026b. URL https://arxiv.org/abs/2609.19969.

Aaron Defazio. Schedulefree+: Scaling learning-rate-free & schedule-free learning to large language models. *arXiv preprint arXiv:2605.19095*, 2026.

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.

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.

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.

GLM-5 Team. GLM-5: from vibe coding to agentic engineering. *arXiv preprint arXiv:2602.15763*, 2026.

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.

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.

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.

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.

Pavel Izmailov, Dmitrii Podoprikhin, Timur Garipov, Dmitry Vetrov, and Andrew Gordon Wilson. Averaging weights leads to wider optima and better generalization. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence*, 2018.

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/.

Jean Kaddour. Stop wasting my time! saving days of imagenet and bert training with latest weight averaging. *arXiv preprint arXiv:2209.14981*, 2022.

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.

Priya Kasimbeg, Frank Schneider, Runa Eschenhagen, Juhan Bae, Chandramouli Shama Sastry, Mark Saroufim, BOYUAN FENG, Less Wright, Edward Z. Yang, Zachary Nado, Sourabh Medapati, Philipp Hennig, Michael Rabbat, and George E. Dahl. Accelerating neural network training: An analysis of the algoperf competition. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=CtM5xjRSfm.

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. URL https://arxiv.org/abs/2607.24653.

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

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

Yunshui Li, Yiyuan Ma, Shen Yan, et al. Model merging in pre-training of large language models. *arXiv preprint arXiv:2505.12082*, 2025a.

Zichong Li, Liming Liu, Chen Liang, Weizhu Chen, and Tuo Zhao. Normuon: Making muon more efficient and scalable. *arXiv preprint arXiv:2510.05491*, 2025b.

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*, 2025a.

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

Zehua Liu, Han Wu, Xiaojin Fu, Shuqi Liu, Xiongwei Han, Tao Zhong, and Mingxuan Yuan. Reg: A regularization optimizer for robust training dynamics. *arXiv preprint arXiv:2510.03691*, 2025c.

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.

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

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.

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

Microsoft AI. Frontier pre-training at Microsoft AI, October 2026. URL https://microsoft.ai/news/frontier-pre-training-at-microsoft-ai/.

Depen Morwani, Nikhil Vyas, Hanlin Zhang, and Sham Kakade. Connections between schedule-free optimizers, ademamix, and accelerated sgd variants. *arXiv preprint arXiv:2502.02431*, 2025.

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

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.

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

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.

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.

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.

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

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.

Xingyu Qu, Peigeng Huang, and Samuel Horvath. Can muon fine-tune adam-pretrained models? *arXiv preprint arXiv:2605.10468*, 2026.

Qwen Team. Qwen3.8-Flash-Next: A new architecture, towards ultimate cost-efficiency, August 2026. URL https://qwen.ai/blog?id=qwen3.8-flash-next.

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.

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

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.

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

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.

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

Tianqi Shen, Jinji Yang, Runze Shi, Jianhao Ma, Jiaye Teng, and Ziye Ma. Towards understanding the power and limits of the muon optimizer: A river-valley perspective, 2026. URL https://arxiv.org/abs/2606.21514.

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

Chongjie Si, Debing Zhang, and Wei Shen. Adamuon: Adaptive muon optimizer. *arXiv preprint arXiv:2507.11005*, 2025.

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

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*, 2025b. URL https://openreview.net/forum?id=CGx4XU9rCA.

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.

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.

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.

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.

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.

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

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.

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

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

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*, 2026b.

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.

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

1 Introduction **1**

2 Related Work **2**

2.1 River-Valley Loss Landscape 2

2.2 Schedule-Free Optimizer 3

2.3 Muon Optimizer 3

3 A River-Valley Analysis of Muon and Schedule-Free Dynamics **4**

3.1 Orthogonalization Makes Muon Updates More Bulk-Oriented 4

3.2 Schedule-Free Stabilizes Muon Trajectories 5

4 AMUSE: Anytime MUon with Stable gradient Evaluation **7**

4.1 AMUSE Follows the River 7

5 Experiments **8**

5.1 Image Domain Experiments 8

5.2 Large Language Model Pretraining 8

5.3 Hyperparameter Sensitivity 10

5.4 Implementation Details 10

6 Conclusion **10**

A A Quadratic Example of Bouncing in Matrix-Normalized Updates **18**

B Additional Results and Experimental Details for Section 3 **20**

B.1 Dominant Updates Can Cause Instability, while Bulk Updates Accelerate Training 20

B.2 Experimental Details 20

B.3 Additional Experimental Results 21

B.4 Extending the Dominant–Bulk Analysis to LLM Pretraining 25

C Details for AMUSE **27**

C.1 Algorithmic Description of AMUSE 27

C.2 Muon Momentum and Averaging After Orthogonalization 28

C.3 Derivation of the β_t Schedule 29

C.4 Implementation Details 30

C.5 Comparison with Concurrent Methods 31

D Image Domain Experiments **33**

D.1 Experimental Details 33

E Large Language Model Experiments **37**

E.1 Architecture Details 37

E.2 Hyperparameter Tuning 37

F Additional Experiments **40**

F.1 Comparison with Additional Optimizer Baselines 40

F.2 Comparison with Muon with EWA and WSD 40

F.3 Comparison with Post-hoc Weight-Averaging Methods 42

F.4 Longer Training Results 42

F.5 Hyperparameter Sensitivity 42

F.6 Wall-clock Time Comparison 44

F.7 Ablation Studies 45

F.8 Robustness Across Random Seeds 48

A A Quadratic Example of Bouncing in Matrix-Normalized Updates

We compare standard gradient descent (GD) with an idealized matrix-normalized update on a two-dimensional matrix quadratic objective. This example is not intended to model the full momentum dynamics of Muon, but rather isolates the magnitude-removing effect of polar normalization. We use an anisotropic setting to show that matrix normalization removes gradient magnitude information, which can cause bouncing of small nonzero components near the optimum.

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 symmetric positive definite matrix with an anisotropic spectrum:

$$\boldsymbol{A} = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top, \qquad \boldsymbol{\Lambda} = \begin{pmatrix} \lambda & 0 \\ 0 & 1 \end{pmatrix},$$

where $\boldsymbol{Q}$ is an orthogonal matrix, and $\lambda \gg 1$ controls the curvature gap between the first and second principal directions. We consider the quadratic objective

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

where the gradient is $\nabla f(\boldsymbol{W}) = \boldsymbol{A}\boldsymbol{W}$. To analyze the dynamics, we project the parameters into the eigenbasis of $\boldsymbol{A}$ by defining $\boldsymbol{V}_t = \boldsymbol{Q}^\top \boldsymbol{W}_t$. Consider a diagonal iterate in this eigenbasis:

$$\boldsymbol{V}_t = \begin{pmatrix} a_t & 0 \\ 0 & b_t \end{pmatrix}.$$

Then, $\boldsymbol{W}_t = \boldsymbol{Q}\boldsymbol{V}_t$, and the gradient is:

$$\nabla f(\boldsymbol{W}_t) = \boldsymbol{A}\boldsymbol{W}_t = (\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{Q}^\top)(\boldsymbol{Q}\boldsymbol{V}_t) = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{V}_t = \boldsymbol{Q}\begin{pmatrix} \lambda a_t & 0 \\ 0 & b_t \end{pmatrix}.$$

Gradient Descent Dynamics. GD with step size η gives $\boldsymbol{W}_{t+1}^{\mathrm{GD}} = \boldsymbol{W}_t - \eta\nabla f(\boldsymbol{W}_t)$. Projecting this update into the eigenbasis yields:

$$\boldsymbol{V}_{t+1}^{\mathrm{GD}} = \boldsymbol{V}_t - \eta\boldsymbol{\Lambda}\boldsymbol{V}_t.$$

For the first coordinate, we have $a_{t+1}^{\mathrm{GD}} = (1-\eta\lambda)a_t$.

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

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

The update magnitude in the first coordinate is $|\Delta a_t^{\mathrm{GD}}| = \eta\lambda|a_t|$, which goes to zero as $a_t \to 0$.

Matrix-Normalized Dynamics. 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:

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

We apply this to the gradient $\boldsymbol{G}_t = \nabla f(\boldsymbol{W}_t) = \boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{V}_t$. When $a_t \neq 0$ and $b_t \neq 0$, we have:

$$\boldsymbol{G}_t^\top\boldsymbol{G}_t = \boldsymbol{V}_t^\top\boldsymbol{\Lambda}^\top\boldsymbol{Q}^\top\boldsymbol{Q}\boldsymbol{\Lambda}\boldsymbol{V}_t = \begin{pmatrix} (\lambda a_t)^2 & 0 \\ 0 & b_t^2 \end{pmatrix}.$$

The inverse square root is $\operatorname{diag}(|\lambda a_t|^{-1}, |b_t|^{-1})$. Therefore,

$$\operatorname{Polar}(\boldsymbol{G}_t) = \boldsymbol{Q}\begin{pmatrix} \lambda a_t & 0 \\ 0 & b_t \end{pmatrix}\begin{pmatrix} |\lambda a_t|^{-1} & 0 \\ 0 & |b_t|^{-1} \end{pmatrix} = \boldsymbol{Q}\begin{pmatrix} \operatorname{sign}(a_t) & 0 \\ 0 & \operatorname{sign}(b_t) \end{pmatrix}.$$

The matrix-normalized update is $\boldsymbol{W}_{t+1}^{\mathrm{MN}} = \boldsymbol{W}_t - \eta\ \operatorname{Polar}(\nabla f(\boldsymbol{W}_t))$. Projecting this back into the eigenbasis ($\boldsymbol{V}_{t+1}^{\mathrm{MN}} = \boldsymbol{Q}^\top\boldsymbol{W}_{t+1}^{\mathrm{MN}}$) gives $a_{t+1}^{\mathrm{MN}} = a_t - \eta\ \operatorname{sign}(a_t)$.

The contrast is therefore clear. GD preserves the magnitude information of the gradient, while the matrix-normalized update removes it through polar decomposition. For every nonzero a_t in this full-rank regime, the matrix-normalized step magnitude remains constant:

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

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

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

so the first coordinate changes sign in one step. Moreover, if $0 < a_t < \eta$, then $a_{t+2}^{\mathrm{MN}} = a_t$, so the dynamics forms a two-cycle along this coordinate. Thus, unlike GD, the matrix-normalized update does not shrink as the component becomes arbitrarily small but nonzero. A small residual component can therefore receive a constant update and bounce across zero.

B Additional Results and Experimental Details for Section 3

In this section, we provide additional experimental details and results for the analysis in Section 3. We include experimental setups and additional results that were omitted from the main text due to space constraints.

B.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. (2025), 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 B.2; hyperparameters are reported in the corresponding figure captions.

We use the dominant and bulk subspaces defined in Definition 3.1. That is, 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.

Let $\boldsymbol{M}_t$ denote the matrix-valued Muon momentum and define the flattened post-orthogonalized update direction as $\boldsymbol{u}_t := \mathrm{vec}(\mathcal{T}(\boldsymbol{M}_t))$. Using the subspace projection matrices from Definition 3.2, we decompose $\boldsymbol{u}_t$ 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 9, 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.

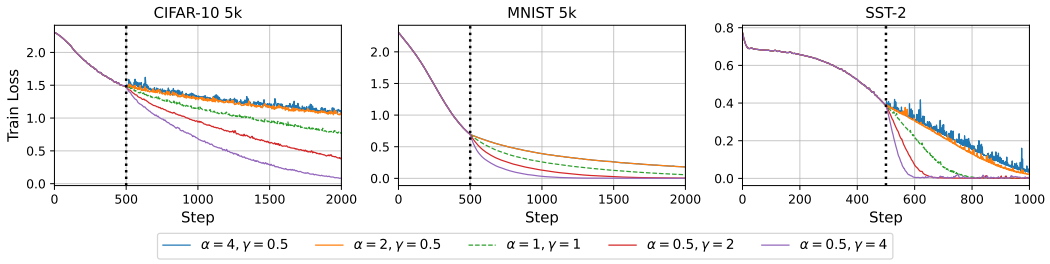


Figure 9: **Scaling Dominant (α) and Bulk (γ) components of Muon update.** We apply subspace-wise scaling to Muon updates starting at step 500. For CIFAR-10-5k and MNIST-5k, Muon is combined with SGD for non-Muon parameters, using momentum 0.9 and learning rate 5×10^{-4}. For SST-2-1k, Muon is combined with AdamW for non-Muon parameters, using Muon momentum 0.9, AdamW coefficients $(0.9, 0.99)$, and learning rate 1×10^{-4}.

B.2 Experimental Details

We follow the experimental setup of Cohen et al. (2021); Song et al. (2025a) for all experiments in Appendix B.3. For experiments in Appendix B.1, we use the same experimental setup except for introducing scaling factors following Zhou et al. (2025).

B.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. (2021); Song et al. (2025a). For convolutional image classification, we use a three-layer CNN with width 32 and ReLU activations, following Cohen et al. (2021); Song et al. (2025a). For text classification, we use a two-layer Transformer with hidden dimension 64 and 8 attention heads, following Damian et al. (2022); Song et al. (2025a).

B.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 3.

B.2.3 Experimental Setup

In all experiments in this section, including those in Section 3, 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 MNIST-5k and CIFAR-10-5k 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 2–4. Unless otherwise specified, each experiment uses the hyperparameters corresponding to its dataset. Specifically, we use the following settings:

- Figure 2, Figure 3, and Figure 10: MNIST hyperparameters in Table 2.
- Figure 10, Figure 11, and Figure 13: CIFAR-10 hyperparameters in Table 3.
- Figure 10, Figure 12, and Figure 14: SST-2 hyperparameters in Table 4.
- Figure 2: AMUSE with learning rate 0.01, $\beta_{\text{init}} = 0.8$, and $\rho = 0.6$.
- Figure 11: AMUSE with learning rate 0.02, $\beta_{\text{init}} = 0.8$, and $\rho = 0.6$.
- Figure 12: AMUSE with learning rate 0.02, $\beta_{\text{init}} = 0.8$, and $\rho = 0.6$.

Table 2: **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 3: **Hyperparameters for CIFAR-10-5k 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

B.3 Additional Experimental Results

In this section, we provide additional results for the analysis in Section 3. We organize the results into three parts: Hessian eigenspectra, Muon dominant-ratio behavior, and SF-Muon dynamics.

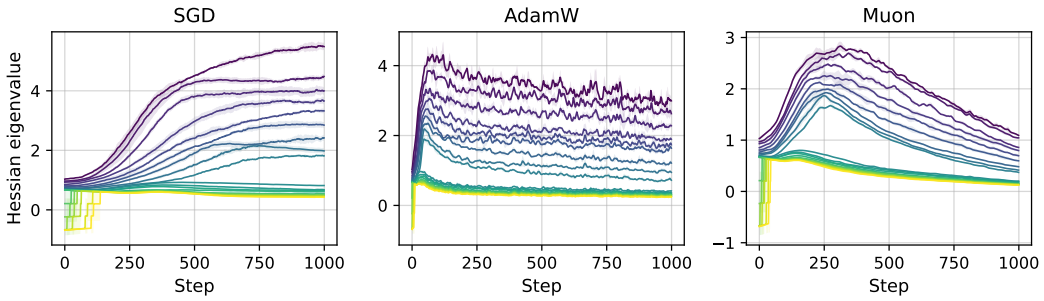
Table 4: **Hyperparameters for SST-2-1k Experiments.**

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-4	(–, 0.99)	0.9

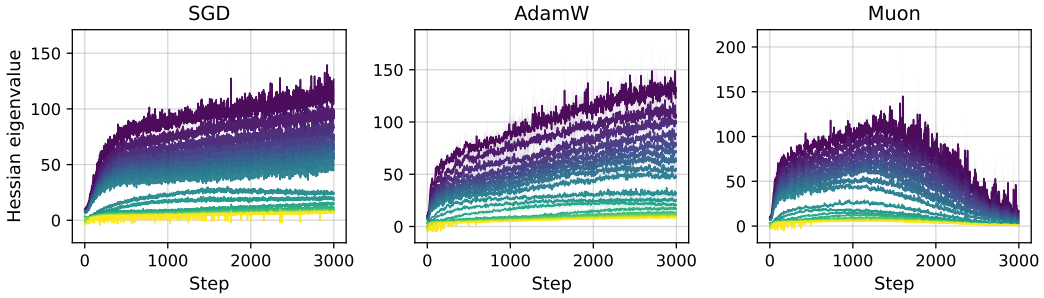
Hessian Eigenspectrum. We first examine the loss Hessian eigenspectrum across datasets and architectures. Figure 10 shows that the Hessian spectrum consistently contains a small number of large outlier eigenvalues, while the remaining eigenvalues are much smaller. This separation supports the dominant/bulk 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 11 and 12 show that Muon produces substantially smaller dominant-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 decreases the dominant component of the update. These results reinforce the explanation that Muon accelerates training by promoting movement in bulk directions, as mentioned in Section 3.1.

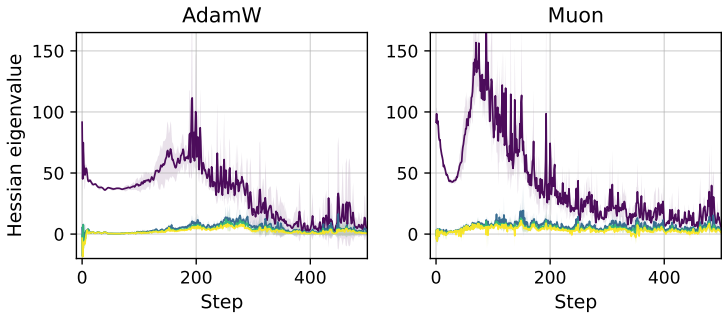
Schedule-Free Muon Dynamics. Finally, we repeat the SF-Muon analysis from the main text (Figure 3) on additional settings. Figures 13 and 14 compare the dominant components of Muon and SF-Muon, and also gradients at multiple virtual interpolation points $\boldsymbol{y}_t^{(\alpha)} = (1-\alpha)\boldsymbol{z}_t + \alpha\boldsymbol{x}_t$. In both settings, the averaged SF-Muon trajectory has a much smaller dominant component than vanilla Muon, and gradients evaluated closer to $\boldsymbol{x}_t$ become more bulk-oriented. These results further support the claim that evaluating gradients near the averaged iterate can suppress valley-wall contributions before the Muon orthogonalization step, as mentioned in Section 3.2.



(a) MNIST with MLP

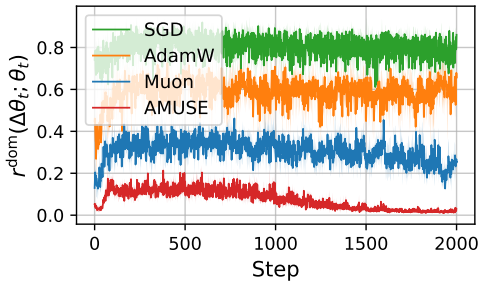


(b) CIFAR-10 with CNN

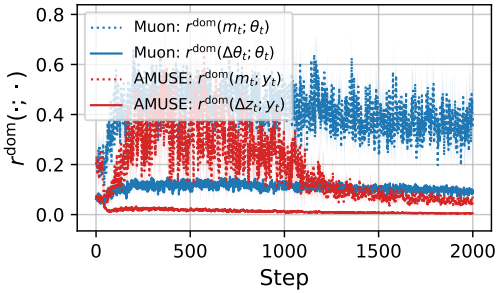


(c) SST-2 with Transformer

Figure 10: **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.

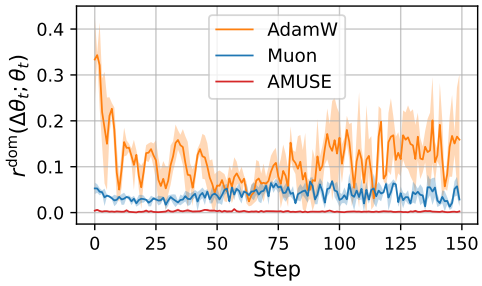


(a) Muon vs. SGD/AdamW

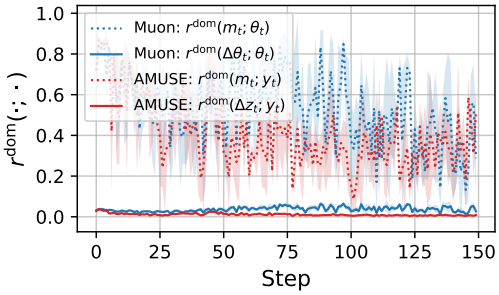


(b) Momentum $\boldsymbol{m}_t$ vs. Update $\Delta\theta_t(=\mathcal{T}(\boldsymbol{m}_t))$

Figure 11: **Comparison of dominant 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 smaller dominant component updates than SGD and AdamW, and AMUSE further suppresses the dominant component. (**b**) 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. Results are averaged over three runs.

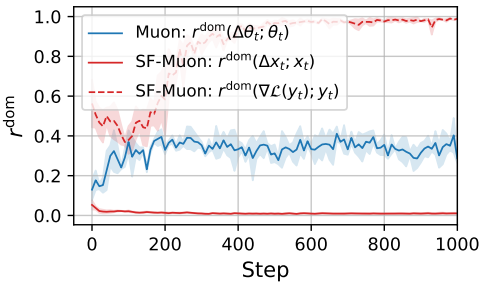


(a) Muon vs. AdamW

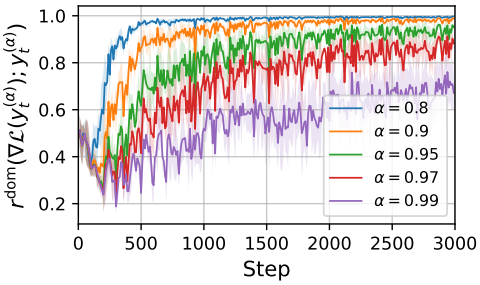


(b) Momentum $\boldsymbol{m}_t$ vs. Update $\Delta\theta_t(=\mathcal{T}(\boldsymbol{m}_t))$

Figure 12: **Comparison of dominant component ratios on SST-2 with Transformer.** Metrics are measured on SST-2 using a Transformer architecture. (**a**) Muon consistently produces substantially smaller dominant component updates than AdamW, and AMUSE further suppresses the dominant component. (**b**) 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. Results are averaged over three runs.

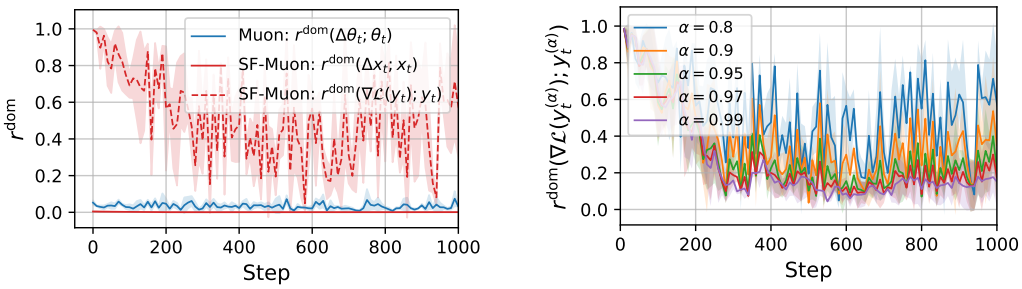


(a) Comparison of dominant ratios



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

Figure 13: **Comparison of dominant 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**) 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.



(a) Comparison of dominant ratios

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

Figure 14: **Comparison of dominant 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**) 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.

B.4 Extending the Dominant–Bulk Analysis to LLM Pretraining

We further extend dominant–bulk analysis in Section 3 to 124M Llama pretraining setup. At each checkpoint, we compute the gradient and Hessian on the same mini-batch of 256 sequences used for the parameter update. We construct the dominant subspace using the top 30 Hessian eigenvectors and measure the dominant ratio following Definition 3.1.

Dominant Ratios of Muon Gradients and Updates. We first compare the dominant ratios of the gradient $\nabla L(\theta_t)$ and the full parameter update $\Delta_t^{\text{full}} = \theta_{t+1} - \theta_t$ along a Muon training trajectory. As shown in Table 5, the gradient exhibits a high dominant ratio. In contrast, the full optimizer update maintains a dominant ratio of approximately 0.01 throughout training. This is consistent with our small-scale observations that Muon orthogonalization promotes bulk-oriented updates.

Table 5: Dominant component ratios of the gradient and full optimizer update along a Muon trajectory during 124M Llama pretraining. The dominant subspace is constructed using the top 30 Hessian eigenvectors.

Method	0K	2K	4K	6K	8K	10K	12K	14K	16K
Muon $\nabla L(\theta_t)$	0.43	0.81	0.76	0.78	0.74	0.76	0.84	0.73	0.74
Muon Δ_t^{full}	0.01	0.01	0.01	0.00	0.01	0.01	0.01	0.01	0.01

Gradient-Evaluation Point in AMUSE. We next compare the dominant ratios of gradients along a separately trained Muon trajectory and at the fast iterate z_t and gradient-evaluation point y_t along an AMUSE trajectory. As shown in Table 6, the gradients along the Muon trajectory and at z_t exhibit relatively high dominant ratios. In contrast, the gradients at y_t consistently exhibit lower dominant ratios than those at z_t after initialization, decreasing overall from 0.79 at 2K steps to 0.31 at 16K steps. These results suggest that moving the gradient-evaluation point toward the averaged iterate x_t suppresses the dominant gradient component before Muon orthogonalization.

Table 6: Dominant component ratios of gradients along Muon and AMUSE trajectories during 124M Llama pretraining. For AMUSE, gradients are evaluated at z_t and y_t.

Method	0K	2K	4K	6K	8K	10K	12K	14K	16K
Muon $\nabla L(\theta_t)$	0.43	0.81	0.76	0.78	0.74	0.76	0.84	0.73	0.74
AMUSE $\nabla L(z_t)$	0.43	0.85	0.93	0.91	0.94	0.89	0.69	0.66	0.69
AMUSE $\nabla L(y_t)$	0.43	**0.79**	**0.70**	**0.52**	**0.39**	**0.41**	**0.47**	**0.37**	**0.31**

Together, these results extend our small-scale observations to LLM pretraining: Muon produces updates concentrated in the bulk subspace, while AMUSE further reduces the dominant gradient component through its time-varying gradient-evaluation point.

C Details for AMUSE

C.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_{init}, AMUSE ρ, Muon momentum μ, AdamW second-moment coefficient β_2, numerical constant ϵ, Muon scaling factor s_{NS}, 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**
  $\eta_t \leftarrow \eta \cdot \min(1, t/T_0)$
  $c_{t+1} \leftarrow \frac{\eta_t^2}{\sum_{i=1}^{t} \eta_i^2}$
  if $t = T_0$ **then**
    $c_{\text{warm}} \leftarrow c_{t+1}$
  end if
  if $t \le T_0$ **then**
    $\beta_t \leftarrow \beta_{\text{init}}$
  else
    $\beta_t \leftarrow 1 - \left(\frac{c_{t+1}(1-c_{\text{warm}})}{c_{\text{warm}}(1-c_{t+1})}\right)^{\rho} (1 - \beta_{\text{init}})$ ▷ Eq. (5)
  end if
  $\boldsymbol{Y}_t \leftarrow (1-\beta_t)\boldsymbol{Z}_t + \beta_t \boldsymbol{X}_t$ ▷ Gradient evaluation point
  $\boldsymbol{G}_t \leftarrow \nabla \mathcal{L}(\boldsymbol{Y}_t)$
  if the parameter group uses Muon **then**
    $\boldsymbol{M}_t \leftarrow \mu \boldsymbol{M}_{t-1} + (1-\mu)\boldsymbol{G}_t$
    $\widehat{\boldsymbol{M}}_t \leftarrow \mu \boldsymbol{M}_t + (1-\mu)\boldsymbol{G}_t$ ▷ Nesterov momentum
    $\boldsymbol{O}_t \leftarrow s_{\text{NS}} \cdot \text{Newton-Schulz}(\widehat{\boldsymbol{M}}_t)$
    $\boldsymbol{Z}_{t+1} \leftarrow (1-\eta_t\lambda)\boldsymbol{Z}_t - \eta_t \boldsymbol{O}_t$
  else if the parameter group uses AdamW **then**
    $\boldsymbol{V}_t \leftarrow \beta_2 \boldsymbol{V}_{t-1} + (1-\beta_2)(\boldsymbol{G}_t \odot \boldsymbol{G}_t)$
    $\widehat{\boldsymbol{V}}_t \leftarrow \boldsymbol{V}_t / (1-\beta_2^t)$
    $\boldsymbol{U}_t \leftarrow \boldsymbol{G}_t \oslash \left(\sqrt{\widehat{\boldsymbol{V}}_t} + \epsilon\right)$
    $\boldsymbol{Z}_{t+1} \leftarrow (1-\eta_t\lambda)\boldsymbol{Z}_t - \eta_t \boldsymbol{U}_t$
  else if the parameter group uses SGD **then**
    $\boldsymbol{Z}_{t+1} \leftarrow (1-\eta_t\lambda)\boldsymbol{Z}_t - \eta_t \boldsymbol{G}_t$
  end if
  $\boldsymbol{X}_{t+1} \leftarrow (1-c_{t+1})\boldsymbol{X}_t + c_{t+1}\boldsymbol{Z}_{t+1}$
  $\boldsymbol{X}_{t+1} \leftarrow (1-c_{t+1})\boldsymbol{X}_t + c_{t+1}\boldsymbol{Z}_{t+1}$ ▷ Update averaged iterate
end for
return $\boldsymbol{X}_{T+1}$

C.2 Muon Momentum and Averaging After Orthogonalization

In this section, we justify why AMUSE retains the Muon momentum buffer $\boldsymbol{M}_t$. Defazio et al. (2024) observe that Schedule-Free SGD and AdamW do not require an explicit first-moment momentum buffer, since the displacement of the averaged sequence itself induces a momentum-like update. In particular, for uniform averaging and constant β, Theorem 4 in Defazio et al. (2024) gives

$$\Delta \boldsymbol{X}_t = \frac{t-1}{t+1}\Delta \boldsymbol{X}_{t-1} + \frac{1}{t+1}\Delta \boldsymbol{Z}_t,$$
$$\Delta \boldsymbol{Y}_t = \beta \Delta \boldsymbol{X}_t + (1-\beta)\Delta \boldsymbol{Z}_t,$$

where $\Delta \boldsymbol{X}_t := \boldsymbol{X}_{t+1} - \boldsymbol{X}_t$, $\Delta \boldsymbol{Y}_t := \boldsymbol{Y}_{t+1} - \boldsymbol{Y}_t$, and $\Delta \boldsymbol{Z}_t := \boldsymbol{Z}_{t+1} - \boldsymbol{Z}_t$. Thus, $\Delta \boldsymbol{X}_t$ acts as an implicit momentum term that stabilizes the movement of the gradient-evaluation sequence $\boldsymbol{Y}_t$.

For AMUSE, however, this implicit Schedule-Free momentum does not replace the Muon momentum buffer. Muon's momentum $\boldsymbol{M}_t$ serves a distinct role: it accumulates recent gradients before the orthogonalization step, thereby determining the direction to which the Muon update is applied. Therefore, AMUSE retains $\boldsymbol{M}_t$ while using the Schedule-Free interpolation to stabilize the point at which gradients are evaluated.

We derive the averaged-sequence update using the Nesterov direction $\widehat{\boldsymbol{M}}_t := \mu \boldsymbol{M}_t + (1-\mu)\nabla \mathcal{L}(\boldsymbol{Y}_t)$ from Algorithm 1:

$$\boldsymbol{Z}_{t+1} = \boldsymbol{Z}_t - \eta \mathcal{T}(\widehat{\boldsymbol{M}}_t), \qquad \boldsymbol{X}_{t+1} = (1-c_{t+1})\,\boldsymbol{X}_t + c_{t+1}\boldsymbol{Z}_{t+1},$$

with initialization $\boldsymbol{Z}_1 = \boldsymbol{X}_1$. Since $c_{t+1} = 1/(t+1)$, $\boldsymbol{X}_t$ is the uniform average of the $\boldsymbol{Z}_t$ iterates, i.e., $\boldsymbol{X}_t = \frac{1}{t}\sum_{i=1}^{t} \boldsymbol{Z}_i$.

Each base iterate can be written as $\boldsymbol{Z}_i = \boldsymbol{Z}_1 - \eta \sum_{j=1}^{i-1} \mathcal{T}(\widehat{\boldsymbol{M}}_j)$. Substituting this expression into the uniform average gives

$$\begin{aligned}
\boldsymbol{X}_t &= \frac{1}{t}\sum_{i=1}^{t}\left(\boldsymbol{Z}_1 - \eta \sum_{j=1}^{i-1} \mathcal{T}(\widehat{\boldsymbol{M}}_j)\right) \\
&= \boldsymbol{Z}_1 - \frac{\eta}{t}\sum_{i=1}^{t}\sum_{j=1}^{i-1} \mathcal{T}(\widehat{\boldsymbol{M}}_j) \\
&= \boldsymbol{Z}_1 - \frac{\eta}{t}\sum_{j=1}^{t-1}(t-j)\mathcal{T}(\widehat{\boldsymbol{M}}_j).
\end{aligned}$$

We now derive the averaged-sequence update $\Delta \boldsymbol{X}_t$ using the expression above:

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

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

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

For $j = t$, only the first sum contributes, giving $-\frac{\eta}{t+1} = -\frac{\eta t}{t(t+1)}$. Combining both cases, we obtain

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

Therefore, the averaged sequence in AMUSE averages a trajectory after orthogonalized Muon updates have already been applied, and $\Delta \boldsymbol{X}_t$ becomes a weighted average of previous orthogonalized updates. This is distinct from Muon momentum: the momentum buffer $\boldsymbol{M}_t$ smooths gradients before orthogonalization, whereas the averaged sequence $\boldsymbol{X}_t$ averages the trajectory induced by the already orthogonalized updates. Empirically, we also find that removing the Muon momentum buffer substantially degrades performance compared to the full AMUSE and SF-Muon update, as shown in Figure 27 in Appendix F.7.

C.3 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. (2025b), 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. To see this, first consider the fixed-β case. Recall the updates from Eq. (1):

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

From the definition of $\boldsymbol{y}_{t+1}$, we have

$$\boldsymbol{z}_{t+1} = \frac{\boldsymbol{y}_{t+1} - \beta\boldsymbol{x}_{t+1}}{1-\beta}.$$

Substituting this expression into the update of $\boldsymbol{x}_{t+1}$ gives

$$\boldsymbol{x}_{t+1} = (1-c_{t+1})\boldsymbol{x}_t + c_{t+1}\frac{\boldsymbol{y}_{t+1} - \beta\boldsymbol{x}_{t+1}}{1-\beta}.$$

Rearranging the terms involving $\boldsymbol{x}_{t+1}$ yields

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

Equivalently,

$$\boldsymbol{x}_{t+1} = \frac{(1-c_{t+1})(1-\beta)}{(1-c_{t+1})(1-\beta)+c_{t+1}}\boldsymbol{x}_t + \frac{c_{t+1}}{(1-c_{t+1})(1-\beta)+c_{t+1}}\boldsymbol{y}_{t+1}.$$

Defining

$$\omega_{t+1} := \frac{c_{t+1}}{(1-c_{t+1})(1-\beta)+c_{t+1}},$$

we obtain the recursion

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

Thus, $\boldsymbol{x}_{t+1}$ is an implicit weighted average of the evaluation points $\boldsymbol{y}_s$, with time-varying averaging coefficient ω_{t+1}. If $c_{t+1} = 1/(t+1)$, the effective weight ω_{t+1} simplifies to

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

This expression suggests that, when designing a time-varying schedule, replacing β by β_t makes the effective averaging coefficient controlled by $t(1-\beta_t)$. If β_t increases too rapidly toward 1, the term $1-\beta_t$ vanishes, causing $t(1-\beta_t)$ to decrease and thereby increasing the effective weight. An increasing effective weight implies that the implicit averaging window actually *shrinks*. From an optimization perspective, shrinking the averaging window at later stages of training is undesirable, since it reduces the stabilizing effect of the averaged sequence. Therefore, the schedule for β_t should be chosen so that $t(1-\beta_t)$ does not decay too rapidly, preventing the implicit averaging window from collapsing.

Based on this observation, we define the upper boundary by considering the fastest schedule that still maintains the implicit averaging window after a given step T_0. Using the β_t notation, the effective averaging coefficient is controlled by

$$\widetilde{\omega}_{t+1} := \frac{c_{t+1}}{(1-c_{t+1})(1-\beta_t)+c_{t+1}}.$$

Since the window size is inversely related to $\widetilde{\omega}_{t+1}$, keeping the window size fixed amounts to keeping $\widetilde{\omega}_{t+1}$ constant. Setting $\widetilde{\omega}_{t+1} = \widetilde{\omega}_{T_0+1}$ for $t \geq T_0$, we obtain

$$\begin{aligned}\widetilde{\omega}_{t+1} &= \frac{c_{t+1}}{(1-c_{t+1})(1-\beta_t)+c_{t+1}}\\ &= \frac{c_{T_0+1}}{(1-c_{T_0+1})(1-\beta_{T_0})+c_{T_0+1}} = \widetilde{\omega}_{T_0+1}.\end{aligned}$$

Solving this equality for β_t gives the constant-window boundary

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

Building on this boundary condition, we introduce an interpolation parameter ρ to control how quickly β_t approaches this boundary. Taking $\beta_{T_0} = \beta_{\text{init}}$, we define, for $t \geq T_0$,

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

This schedule interpolates between the standard fixed-β schedule and the constant-window boundary: when $\rho = 0$, we recover $\beta_t = \beta_{\text{init}}$, while when $\rho = 1$, we recover the boundary that keeps $\widetilde{\omega}_t$ constant. By restricting $\rho \in [0, 1]$, we ensure that the implicit averaging window is allowed to grow for $0 \leq \rho < 1$ and remains constant at the boundary when $\rho = 1$. Thus, the schedule prevents the averaging window from shrinking while still allowing β_t to increase over training.

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 averaged 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:

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

This weight α_t directly dictates the size and shape of the effective averaging window. In Figure 15, we plot the histogram of these weights α_t (initialized with $\beta_{\text{init}} = 0.8$) 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 grow approximately linearly 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 16.

C.4 Implementation Details

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. (2024). 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.

Batch Normalization. Averaging methods, including Schedule-Free (Defazio et al., 2024), 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 (Wu and Johnson, 2021). 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.

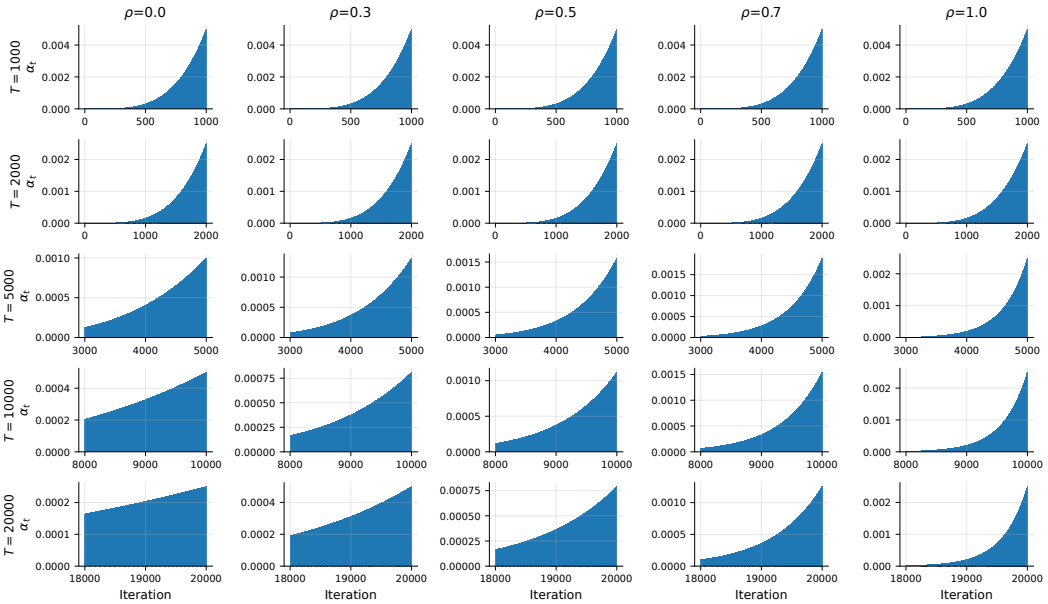


Figure 15: **Effect of ρ on effective weights α_t.** Histograms of α_t across varying sequence lengths T. For $T > T_0 = 2000$, lower ρ values allow the effective averaging window to continuously grow, whereas $\rho = 1$ strictly fixes the window to its size at T_0.

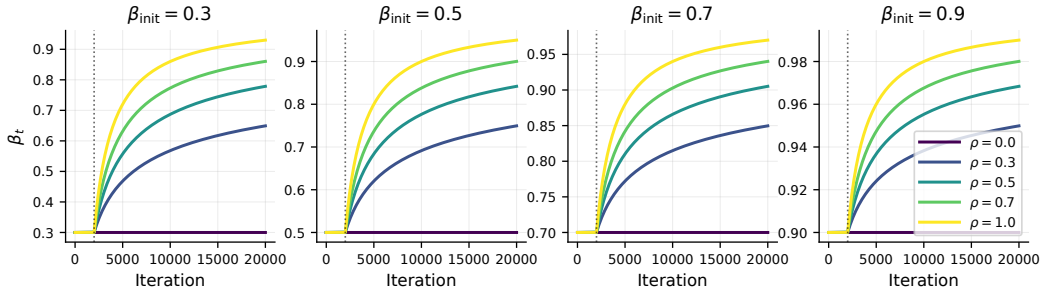


Figure 16: **Evolution of β_t controlled by ρ.** After the initial warmup phase ($T_0 = 2000$, dotted vertical line), ρ smoothly interpolates between a constant β baseline ($\rho = 0$) and the strict constant-window upper bound ($\rho = 1$).

Weight Decay. Following Defazio et al. (2024), 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 and 4 B200 GPUs with data-parallel training.

C.5 Comparison with Concurrent Methods

We discuss two concurrent works that are closely related to AMUSE: SF-NorMuon, which combines Schedule-Free optimization with normalized Muon-style updates, and ScheduleFree+, which extends Schedule-Free AdamW to large-scale language model pretraining.

SF-NorMuon. SF-NorMuon (Apte et al., 2026) extends NorMuon (Li et al., 2025b) to the Schedule-Free framework. They provide convergence guarantees for schedule-free spectral optimization and

analyze the role of weight decay in long-horizon training. Their analysis suggests that applying weight decay to the fast iterate $\boldsymbol{z}_t$ improves stability. This finding is consistent with AMUSE, which also applies decoupled weight decay directly to $\boldsymbol{z}_t$ (Algorithm 1). A key difference is that SF-NorMuon considers a fixed interpolation coefficient β throughout training, whereas AMUSE considers a time-varying coefficient β_t.

ScheduleFree+. Defazio (2026) proposes ScheduleFree+, an extension of Schedule-Free AdamW designed for large-scale language model pretraining. Among other changes, ScheduleFree+ reintroduces Adam's inner first-moment momentum and incorporates fully decoupled AdamC weight decay and a Polyak-style adaptive step-size rule. For long-duration training, it also anneals the Schedule-Free interpolation coefficient β_t. The closest connection to our method lies in the design of this coefficient: both methods increase β_t over training, thereby shifting the gradient evaluation point $y_t = (1-\beta_t)z_t + \beta_t x_t$ from the fast iterate z_t toward the averaged iterate x_t.

ScheduleFree+ implements this transition by log-linearly interpolating the residual coefficient $1-\beta_t$ between fixed initial and final endpoints:

$$1-\beta_t = (1-\beta_{\text{initial}})^{1-\tau_t}(1-\beta_{\text{final}})^{\tau_t}, \qquad \tau_t = \min\left\{\frac{t}{T_{\text{anneal}}}, 1\right\}.$$

In its long-duration experiments, the annealing horizon spans the full training run, whereas the shorter-duration experiments use a fixed β. In contrast, following the schedule design in Appendix C.3, our method uses the power-law form

$$1-\beta_t = (1-\beta_{\text{init}})\,(T_0/t)^{\rho},$$

where T_0 is a warmup iteration and $\rho > 0$ controls the rate at which the gradient evaluation point shifts toward the averaged iterate. Unlike the endpoint-based rule of ScheduleFree+, our schedule does not require specifying either a terminal value β_{final} or a total annealing horizon.

We further examine related SF-AdamW variants in Appendix F.7, where we study the effect of applying our β_t schedule both with and without Adam's inner first-moment accumulation.

D Image Domain Experiments

D.1 Experimental Details

For the image-domain experiments, we mostly follow the experimental setup of Defazio et al. (2024). We additionally include Muon with cosine annealing as a new baseline and add the ISIC 2018 image segmentation task.

We follow the standard Muon hybrid setup (Jordan et al., 2024): Muon is applied to hidden matrix-valued layers, while SGD with momentum is used for normalization layers, biases, and classifier heads. For MAE fine-tuning in the image-domain experiments, we instead adopt the Muon+AdamW hybrid setup commonly used for Transformer models (Jordan et al., 2024; Liu et al., 2025a). Specifically, 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. (2024) for SGD, SF-SGD, AdamW, and SF-AdamW. For the added baseline Muon, we tune the learning rate, weight decay, and Muon momentum 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 optimizer as for Muon, reducing the number of tuned hyperparameters. In the image-domain experiments, except for MAE fine-tuning, AMUSE uses SGD without momentum for the non-Muon parameters, which corresponds to an SF-SGD update. For MAE fine-tuning, AMUSE instead uses AdamW with no first-moment momentum and fixes the second-moment coefficient to $\beta_2 = 0.999$, which corresponds to an SF-AdamW update. Thus, beyond the base learning rate and weight decay, AMUSE only tunes the AMUSE-specific parameters β_{init} and ρ.

CIFAR-10. We train a Wide-ResNet-16-8 architecture (Zagoruyko and Komodakis, 2016) on the CIFAR-10 image classification (Krizhevsky et al., 2009) dataset. We use a batch size of 128 and train for 300 epochs.

Table 7: **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_{init}	–	0.8
AMUSE ρ	–	0.3
LR decay scheduler	cosine	–

CIFAR-100. We train a DenseNet architecture (Huang et al., 2017) on the CIFAR-100 image classification dataset. We use a batch size of 64 and train for 300 epochs.

Table 8: **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	2e-4	2e-4
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	1e-3	0.5
Weight decay	5e-3	2e-3
Muon momentum μ	0.9	0.95
SGD momentum β	0.9	–
AMUSE β_{init}	–	0.7
AMUSE ρ	–	0.4
LR decay scheduler	cosine	–

SVHN. We train a ResNet-3-96 architecture (He et al., 2016) on the Street View House Numbers (SVHN) dataset (Netzer et al., 2011). We use a batch size of 32 and train for 300 epochs.

Table 9: **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	1e-4	2e-4
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	0.05
SGD LR	1e-3	0.05
Weight decay	2e-3	0.01
Muon momentum μ	0.9	0.95
SGD momentum β	0.9	–
AMUSE β_{init}	–	0.8
AMUSE ρ	–	0.2
LR decay scheduler	cosine	–

ImageNet-1k. We train a ResNet-50 architecture (He et al., 2016) on the ILSVRC 2012 (Russakovsky et al., 2015) ImageNet-1k classification dataset. We use a batch size of 256 and train for 100 epochs.

Table 10: **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	1e-4	5e-5
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	1e-5	0.5
Weight decay	1e-3	5e-4
Muon momentum μ	0.9	0.95
SGD momentum β	0.9	–
AMUSE β_{init}	–	0.5
AMUSE ρ	–	0.8
LR decay scheduler	cosine	–

ISIC 2018. We train a base U-Net architecture (Ronneberger et al., 2015) on the ISIC 2018 dataset (Codella et al., 2019) 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[3].

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 11: **Hyperparameters for ISIC 2018.**

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	5e-4	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	1e-3	0.2
Weight decay	1e-3	0.05
Muon momentum μ	0.9	0.95
SGD momentum β	0.9	–
AMUSE β_{init}	–	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 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_{init} for AMUSE. We use smaller β_{init} values 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[4].

[3] https://github.com/FengheTan9/U-Bench
[4] https://github.com/facebookresearch/mae

Table 12: **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	2e-4	2e-3
Weight decay	0.01	0.05
Muon momentum	0.95	0.95
AdamW β_1	0.9	–
AMUSE β_{init}	–	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	

E Large Language Model Experiments

E.1 Architecture Details

Our experimental setup follows Semenov et al. (2026), using their public codebase.[5] We use three decoder-only Llama-style models with approximately 124M, 720M, and 1.3B 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 13 summarizes the model-specific dimensions.

Table 13: **Model configurations for LLM experiments.**

Model size	Layers	Attention heads	Hidden size
124M	12	12	768
720M	12	16	2048
1.3B	24	32	2048

E.2 Hyperparameter Tuning

For AdamW and SF-AdamW, we use the best-performing hyperparameter configurations reported by Semenov et al. (2026).

For Muon, we conduct an expanded hyperparameter search of 60 configurations, starting from the best configuration reported by Semenov et al. (2026). The search varies learning rate, warmup steps, weight decay, Muon momentum, and the AdamW second-moment coefficient β_2, covering both the learning-rate regime near the reported configuration and substantially larger learning rates. For fixed-β SF-Muon, we evaluate 37 configurations in the 124M setting. We use the same hyperparameter grids as AMUSE and test fixed interpolation coefficients $\beta \in \{0.6, 0.8, 0.9, 0.95, 0.98\}$.

For AMUSE, we first tune the learning rate and then explore selected combinations of weight decay, warmup steps, and AMUSE-specific hyperparameters. Specifically, we sweep the initial interpolation value $\beta_{\text{init}} \in \{0.4, 0.6\}$ and $\rho \in \{0.6, 0.8\}$, with $\rho = 0.8$ consistently performing well across our experiments. We do not additionally tune the base-optimizer momentum parameters, fixing the Muon momentum to $\mu = 0.95$ and AdamW's second-moment coefficient for non-Muon parameters to $\beta_2 = 0.999$. In total, we evaluate 34 AMUSE configurations, compared with 60 additional configurations for Muon.

Since Semenov et al. (2026) report experiments only up to the 720M scale, we additionally evaluate a 1.3B model to test whether the observed results persist at a larger scale. For this setting, we initialize each optimizer from its corresponding 720M hyperparameter configuration and perform a learning-rate sweep for all methods, including AMUSE. See Tables 14–17 for the selected hyperparameters, and Table 18 for the AMUSE hyperparameter search grid.

[5] https://github.com/epfml/llm-optimizer-benchmark

E.2.1 Hyperparameters for 124M parameters model

Table 14: **Selected hyperparameters at the 124M scale.**

AdamW and Muon (D-Muon)

Hyperparameter	AdamW	Muon
Learning rate	0.001	0.01
Batch size	256	256
Sequence length	512	512
Warmup steps	2000	2000
Weight decay	0.1	0.05
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	4000
Weight decay	0.1	0.05
Gradient clipping	0.5	0.5
AMUSE β_{init}	–	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	–	–

E.2.2 Hyperparameters for 720M parameters model

Table 15: **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_{init}	–	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	–	–

E.2.3 Hyperparameters for 1.3B parameters model

Table 16: **Selected hyperparameters for AdamW and Muon (D-Muon) at the 1.3B scale.** We initialize from the 720M configuration and sweep the learning rate. Bold values indicate the selected learning rate.

Hyperparameter	AdamW	Muon
Learning rate	0.0005, **0.001**, 0.002	0.0005, 0.001, **0.002**
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

Table 17: **Selected hyperparameters for SF-AdamW and AMUSE at the 1.3B scale.** We initialize from the 720M configuration and sweep the learning rate. Bold values indicate the selected learning rate.

Hyperparameter	SF-AdamW	AMUSE
Learning rate	0.0005, 0.001, **0.002**	0.005, **0.01**, 0.02
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_{init}	–	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	–	–

E.2.4 Search grid table for AMUSE

Table 18: **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	$2000,\ \mathbf{4000}$
	Weight decay	$0.01,\ \mathbf{0.05},\ 0.1$
	Gradient clipping	$\mathbf{0.5}$
	AMUSE β_{init}	$0.4,\ \mathbf{0.6}$
	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_{init}	$\mathbf{0.4},\ 0.6$
	AMUSE ρ	$\mathbf{0.8}$
1.3B	Learning rate	$0.005, \mathbf{0.01}, 0.02$
	Batch size	2048
	Sequence length	512
	Warmup steps	$\mathbf{2000}$
	Weight decay	$\mathbf{0.1}$
	Gradient clipping	$\mathbf{0.1}$
	AMUSE β_{init}	$\mathbf{0.4}$
	AMUSE ρ	$\mathbf{0.8}$
Fixed across sweeps: AdamW $\beta_2 = 0.999$, Muon momentum 0.95, no LR decay.		

F Additional Experiments

F.1 Comparison with Additional Optimizer Baselines

We first compare AMUSE against a broad set of optimizer baselines for pretraining a 124M-parameter Llama model. In addition to AdamW, Muon, and SF-AdamW, we evaluate AdEMAMix, ADOPT, Signum, Prodigy, MARS, Lion, Sophia, and SOAP using the tuned hyperparameters reported by Semenov et al. (2026). As shown in Figure 17, AMUSE consistently achieves the lowest validation perplexity throughout training. AMUSE also attains the best final validation perplexity, indicating that its advantage is not limited to the main baselines considered in the paper.

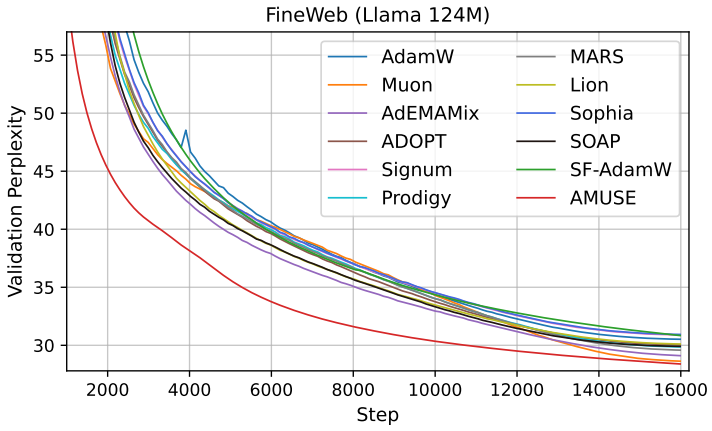


Figure 17: **Comparison with Other Baselines on Llama 124M model.**

Recent Muon Variants. We also compare AMUSE with two recent Muon variants: AdaMuon (Si et al., 2025), an adaptive variant of Muon, and Mousse (Zhang et al., 2026b), a curvature-aware variant. Both methods are evaluated on the same 124M Llama pretraining setup.

For Mousse, we evaluate 45 hyperparameter configurations, varying learning rate, weight decay, warmup steps, and Shampoo momentum, while keeping the curvature exponent fixed. For AdaMuon, we tune learning rate, weight decay, warmup steps, the auxiliary AdamW second-moment coefficient, and the shared momentum/second-moment coefficient.

Table 19 compares validation perplexity, wall-clock time, optimizer-state memory, and median throughput. While both AdaMuon and Mousse achieve competitive performance, AMUSE attains the lowest validation perplexity. Moreover, AMUSE achieves comparable runtime and throughput to AdaMuon, while Mousse incurs substantially higher computational and memory overhead.

Table 19: Comparison with recent Muon variants on 124M Llama pretraining on one NVIDIA B200 GPU. Lower validation perplexity is better.

Method	Val. PPL ↓	Time	State (GiB)	Throughput (k tok/s)
AdaMuon	28.70	1h 54m	**0.76**	376.8
Mousse	28.56	2h 48m	2.52	366.4
AMUSE	**28.48**	**1h 53m**	0.92	**377.8**

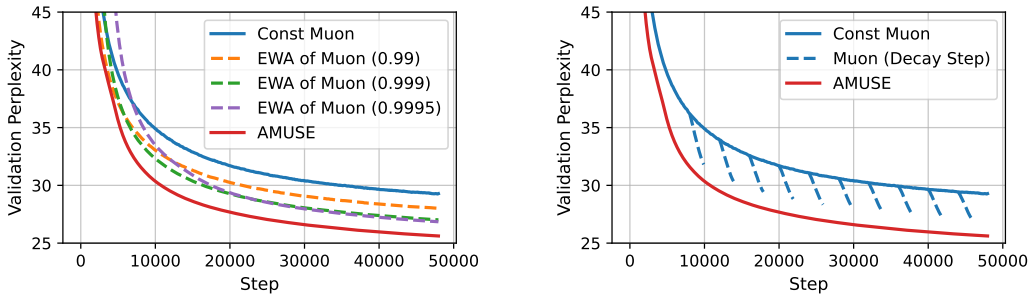
F.2 Comparison with Muon with EWA and WSD

We further compare AMUSE against two extensions of constant step size Muon with warmup: exponential weight averaging (EWA) and warmup-stable-decay (WSD) scheduling. These baselines test two different alternatives to AMUSE. EWA examines whether AMUSE's gains can be recovered by applying an external averaging mechanism to a Muon trajectory, whereas WSD examines whether they can be matched by adding an explicit learning rate decay phase.

For the 124M Llama experiment, we first tune constant step size Muon over learning rate, warmup steps, and weight decay, while keeping the momentum parameters identical to those used in the cosine-decay Muon baseline. The best constant step size Muon configuration uses learning rate 10^{-3}, weight decay 0.05, and $1,000$ warmup steps. With these tuned hyperparameters, we evaluate EWA with multiple averaging coefficients $(0.99, 0.999, 0.9995)$. Separately, to evaluate the effect of WSD, we start from multiple Muon checkpoints and apply a 2k-iteration decay phase.

As shown in Figure 18, both strategies improve constant step size Muon. Stronger EWA coefficients lead to better validation perplexity, suggesting that external averaging helps stabilize the final model. Similarly, applying a short decay phase from Muon checkpoints substantially lowers perplexity, showing that part of Muon's degradation can be mitigated by an explicit terminal decay. However, none of the EWA variants or WSD decay runs matches AMUSE. Moreover, AMUSE consistently achieves better perplexity across the evaluated training trajectory. These results suggest that AMUSE does not merely imitate post-hoc averaging or learning rate decay. Instead, by stabilizing the gradient-evaluation trajectory during training, AMUSE preserves Muon's fast progress while maintaining better optimization stability.

For the image-domain experiments, we also evaluate Muon with EWA on ImageNet, using the same grid of EWA coefficients as in the Llama experiments. We start from the best hyperparameter configuration of the cosine-scheduled Muon baseline, but train Muon with a constant learning rate, and then apply EWA to the resulting trajectory. As shown in Figure 19, Muon with EWA does not outperform AMUSE for any averaging coefficient. Together with the Llama results, this suggests that AMUSE does not simply behave like an external weight averaging mechanism. Instead, AMUSE appears to maintain a faster and more stable trajectory during training.



(a) Muon with multiple EWA coefficients

(b) Muon with WSD decay

Figure 18: **Comparison with Muon+EWA and Muon+WSD on 124M Llama pretraining.** (**a**): We apply exponential weight averaging (EWA) with multiple averaging coefficients to the tuned constant-LR Muon trajectory. (**b**): We apply 2k-step decay phase starting from multiple Muon checkpoints. Both EWA and WSD improve constant-LR Muon, but none of these variants matches AMUSE.

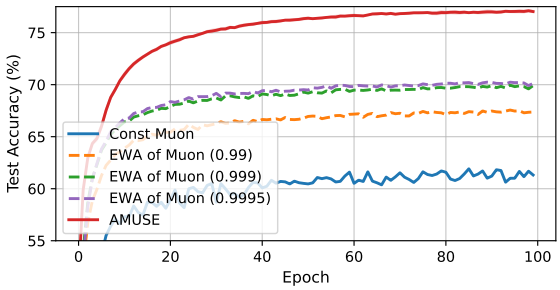


Figure 19: **Comparison with Muon+EWA on ImageNet.** AMUSE achieves the best performance, outperforming Muon with EWA across all averaging coefficients.

Table 20: Comparison with post-hoc weight-averaging methods on 124M Llama pretraining. Lower is better.

Method	8K	10K	13K	16K
Muon baseline	36.021	34.872	33.716	32.928
Muon + SWA	33.169	31.952	30.773	29.956
Muon + LAWA	33.572	32.087	30.784	29.950
Muon + PMA-EMA	33.754	32.142	30.765	29.910
Muon + PMA-WMA	33.410	31.981	30.719	29.906
AMUSE	**31.716**	**30.447**	**29.262**	**28.484**

F.3 Comparison with Post-hoc Weight-Averaging Methods

To examine whether AMUSE's improvements can be reproduced by post-hoc weight averaging, we compare AMUSE with established methods, including Stochastic Weight Averaging (SWA) (Izmailov et al., 2018), Latest Weight Averaging (LAWA) (Kaddour, 2022), and Pre-trained Model Average (PMA) (Li et al., 2025a), on 124M Llama pretraining.

Experimental setup. We apply these methods to tuned constant-learning-rate Muon trajectories. We save checkpoints every 500 iterations for three promising learning rates and tune the averaging hyperparameters for each method, including the averaging start point, window size, checkpoint stride, and exponential decay rate, as applicable. For PMA-EMA, we additionally tune the exponential decay parameter α.

Results. Table 20 reports the best results at different training steps. All weight-averaging methods substantially improve upon the Muon baseline. Nevertheless, AMUSE consistently achieves the best performance across all stopping points.

Beyond the performance difference, AMUSE operates online without requiring checkpoint storage or post-hoc selection of averaging configurations. In contrast, LAWA and the windowed PMA variants require access to multiple saved checkpoints in our evaluation. This highlights the distinction between post-hoc averaging of an existing trajectory and AMUSE's online modification of the gradient-evaluation point.

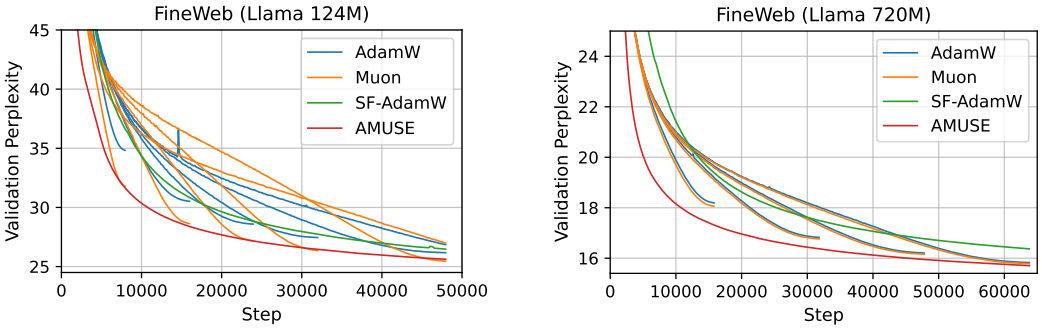
F.4 Longer Training Results

We further train AMUSE and the main baselines on 124M and 720M Llama models for extended horizons, corresponding to approximately $2.5\times$ and $4.5\times$ the Chinchilla token budget, respectively. Following Semenov et al. (2026), we retain the hyperparameter configurations tuned for $16{,}000$ iterations and evaluate performance at different stopping points.

As shown in Figure 20, AMUSE achieves the lowest validation perplexity at most evaluated stopping points for the 124M model and consistently outperforms all baselines across the evaluated training budgets for the 720M model. Notably, unlike cosine-scheduled baselines whose learning-rate schedules must be specified for a predefined training horizon, AMUSE requires only a single run with no explicit decay schedule. This *single AMUSE trajectory* performs strongly across different training horizons, suggesting that its averaged trajectory remains effective beyond the standard training budget.

F.5 Hyperparameter Sensitivity

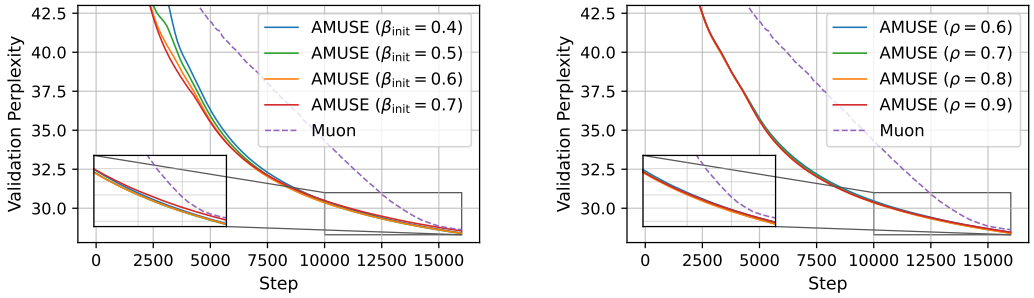
Language Model Pretraining (FineWeb with 124M Llama). Figure 21 and Table 21 show the sensitivity of AMUSE to β_{init} and ρ in the 124M Llama setting. AMUSE shows only mild variation in validation perplexity across the tested ranges.



(a) 124M Longer-horizon Training

(b) 720M Longer-horizon Training

Figure 20: **Longer-horizon language model pretraining on FineWeb.** We train AMUSE and the main baselines on 124M and 720M Llama models for approximately $2.5\times$ and $4.5\times$ the Chinchilla token budget, respectively. Unlike cosine-scheduled baselines that require a predefined training horizon, a single AMUSE trajectory performs strongly across training horizons, highlighting its anytime training capability.



(a) β_{init} Sweep Results

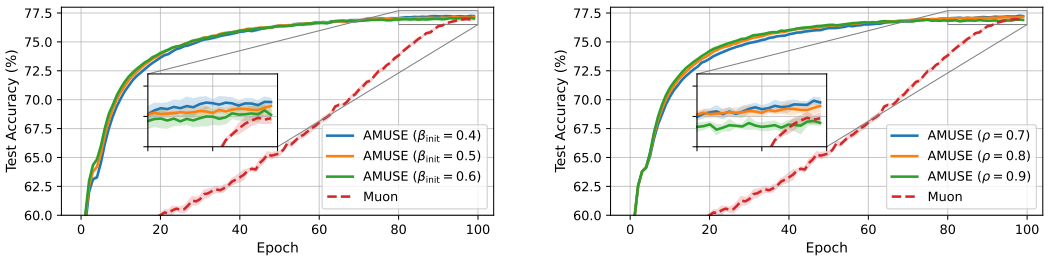
(b) ρ Sweep Results

Figure 21: **Hyperparameter sensitivity varying β_{init} and ρ on Llama-124M training.** AMUSE remains robust to both β_{init} and ρ, exhibiting stable learning curves across the tested configurations.

Table 21: **Hyperparameter sensitivity of AMUSE on 124M Llama pretraining.**

Sweep	Hyperparameter value	Validation perplexity
β_{init} sweep, $\rho = 0.8$	0.4	28.50
	0.5	28.47
	0.6	28.48
	0.7	28.64
ρ sweep, $\beta_{\text{init}} = 0.6$	0.6	28.55
	0.7	28.50
	0.8	28.48
	0.9	28.56
Muon with cosine decay	–	28.63

Image Classification (ImageNet with ResNet-50). Figure 22 and Table 22 show analogous results in the image-domain setting, using ImageNet with ResNet-50. AMUSE is similarly robust to both β_{init} and ρ: its learning curves remain stable across the tested ranges, and nearly all configurations outperform the tuned Muon baseline with cosine learning-rate decay.



(a) β_{init} Sweep Results

(b) ρ Sweep Results

Figure 22: **Hyperparameter sensitivity varying β_{init} and ρ on ImageNet training.** AMUSE exhibits stable performance across a range of hyperparameter choices. Results are averaged over five runs.

Table 22: **Hyperparameter sensitivity of AMUSE using ImageNet with ResNet-50.**

Sweep	Hyperparameter value	Test Accuracy (%)
β_{init} sweep, $\rho = 0.8$	0.4	77.24
	0.5	77.17
	0.6	77.03
ρ sweep, $\beta_{\text{init}} = 0.5$	0.7	77.23
	0.8	77.17
	0.9	76.89
Muon with cosine decay	–	76.97

F.6 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 23: **Wall-clock time comparison on 124M Llama pretraining.** We report average iteration time on four NVIDIA A6000 GPUs 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	194469
SF-AdamW	0.684	1.015	191626
Muon	0.706	1.047	185654
AMUSE	0.719	1.067	182298

Figure 23 compares validation perplexity and test accuracy against wall-clock time for 124M and 720M Llama pretraining and ResNet-50 training on ImageNet. AMUSE retains its performance advantage with a total runtime comparable to Muon. We obtain the time coordinates by scaling logged training progress by each method's total runtime, assuming a constant step or epoch cost; SF-Muon uses the same runtime as AMUSE.

Time to target. Table 24 reports the training progress and wall-clock time needed to reach fixed performance targets, using the measured average training throughput over 16K LLM iterations or 100 ImageNet epochs. For ImageNet SF-Muon, we use the first recorded evaluation of the five-seed mean curve in Figure 23 that exceeds the target, counting completed training epochs and using AMUSE's

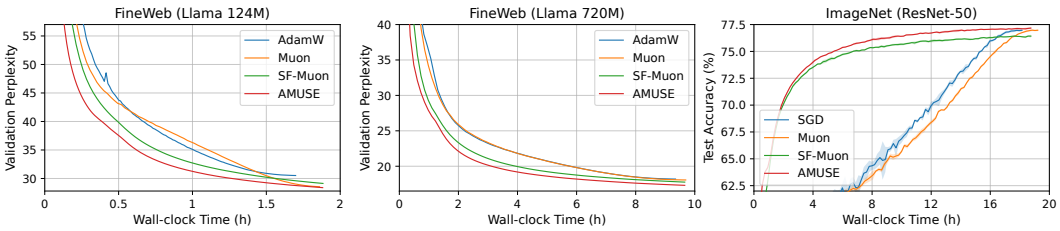


Figure 23: Validation perplexity and test accuracy versus wall-clock time. Curves end at the last recorded evaluation.

total runtime. Timing measurements use one NVIDIA B200 GPU for 124M Llama and ImageNet, and four B200 GPUs with data parallelism for 720M Llama.

Table 24: Training progress and wall-clock time needed to first reach fixed performance targets. Times are rounded to the nearest minute.

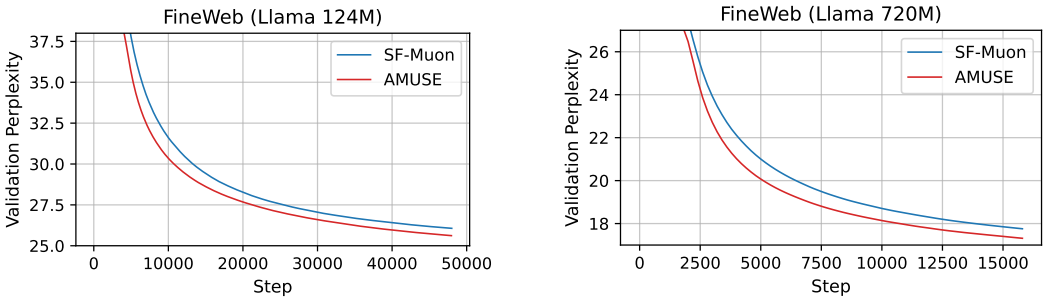
Setting and target	Optimizer	Steps / epochs	Time to target
Llama 124M Validation PPL < 33	AdamW	11,270	1h 12m
	Muon	10,810	1h 16m
	SF-Muon	8,050	57m
	AMUSE	**6,670**	**47m**
Llama 720M Validation PPL < 19	AdamW	12,000	7h 06m
	Muon	11,600	7h 07m
	SF-Muon	9,000	5h 34m
	AMUSE	**7,000**	**4h 17m**
ImageNet (ResNet-50) Test accuracy $> 75\%$	SGD	84	15h 15m
	Muon	85	16h 20m
	SF-Muon	36	6h 45m
	AMUSE	**28**	**5h 15m**

AMUSE reaches every target in the least wall-clock time, improving time to target by approximately $1.21\times$, $1.30\times$, and $1.29\times$ over the next-fastest method for 124M Llama, 720M Llama, and ImageNet, respectively.

F.7 Ablation Studies

Comparison with SF-Muon. We compare AMUSE with fixed-β SF-Muon. SF-Muon is tuned over the same hyperparameter grid, including multiple fixed β values. As shown in Figures 24a and 24b, AMUSE consistently achieves lower validation perplexity at both Llama model scales. On image classification benchmarks, Figure 25 shows that AMUSE achieves higher test accuracy, while fixed-β SF-Muon tends to saturate earlier in training.

These results suggest that a fixed interpolation coefficient cannot effectively balance rapid early-stage progress with consistent improvement later in training. In contrast, by gradually increasing β_t, AMUSE enables faster progress in the early stages while maintaining stable improvements later in training.



(a) FineWeb Llama 124M

(b) FineWeb Llama 720M

Figure 24: **AMUSE and fixed-β SF-Muon on 124M/720M Llama pretraining.** We compare AMUSE with SF-Muon using the tuned fixed interpolation value, $\beta = 0.95$. AMUSE achieves lower validation perplexity at both model scales.

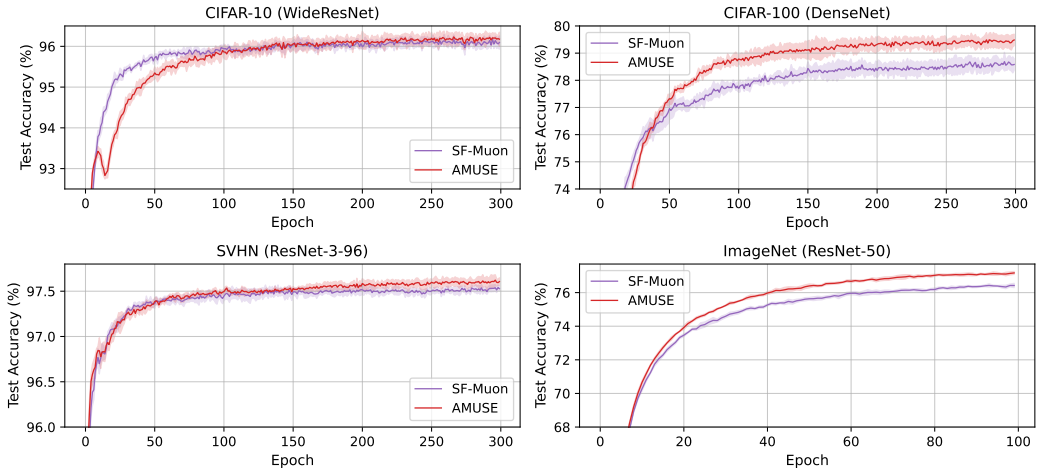


Figure 25: **AMUSE and fixed-β SF-Muon on Image domain experiments.** We compare AMUSE with SF-Muon in image classification benchmarks.

Evaluating the Proposed β_t Schedule. To evaluate the design of the β_t schedule in Eq. (5), we compare it with tuned linear and cosine alternatives. This ablation tests whether simply increasing β_t is sufficient to explain AMUSE's gains. Table 25 shows that the proposed schedule yields the lowest validation perplexity at 16K iterations. The alternatives require the final iteration T through normalized training progress, whereas Eq. (5) is independent of the training horizon.

Table 25: Comparison of the β_t schedule in Eq. (5) with tuned linear and cosine alternatives. Lower validation perplexity is better.

β_t schedule	Validation perplexity at 16K
Linear	29.18
Cosine	29.90
AMUSE (Eq. (5))	**28.48**

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, $\beta_{\text{init}} = 0.6$ 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 26, 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.

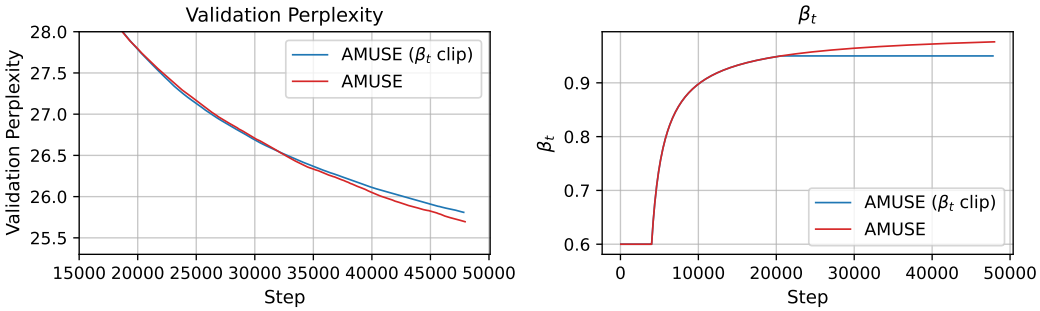
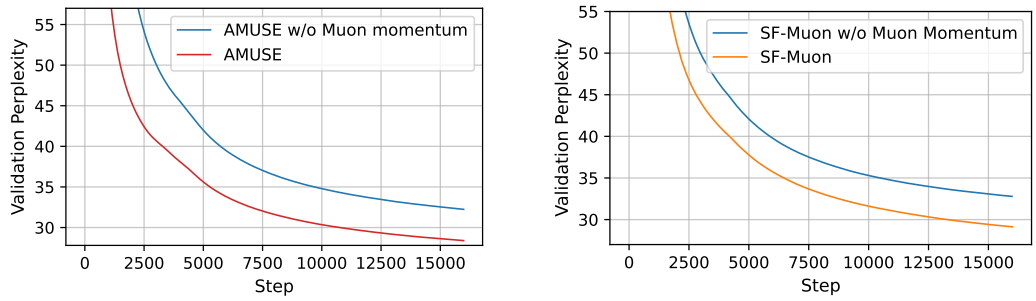


Figure 26: **Effect of stopping the increase of β_t in AMUSE.** (**Left**) Validation perplexity on 124M Llama pretraining. The clipped variant underperforms AMUSE, with the gap widening later in training. (**Right**) Evolution of β_t. While both methods reach similar large values, AMUSE continues to increase β_t, whereas the clipped variant saturates to 0.95.

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 27, 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 27: **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.

Applying β_t Scheduling and Momentum to SF-AdamW. We investigate whether the modified β_t schedule introduced in Eq. (5) can also improve the performance of SF-AdamW. As shown in Figure 28, applying only the β_t schedule does not improve performance over the original SF-AdamW. In contrast, when combined with AdamW's first-moment ($\beta_1 = 0.9$), the β_t schedule consistently improves performance throughout training, yielding lower validation perplexity in both the early and late stages.

Interestingly, neither the β_t schedule nor first-moment alone improves performance over the original SF-AdamW. In contrast, applying both together consistently improves validation perplexity throughout training, indicating that the two components are effective only when used in combination.

However, incorporating first-moment in AdamW into SF-AdamW requires maintaining an additional momentum buffer, resulting in higher memory consumption than either AdamW or AMUSE.

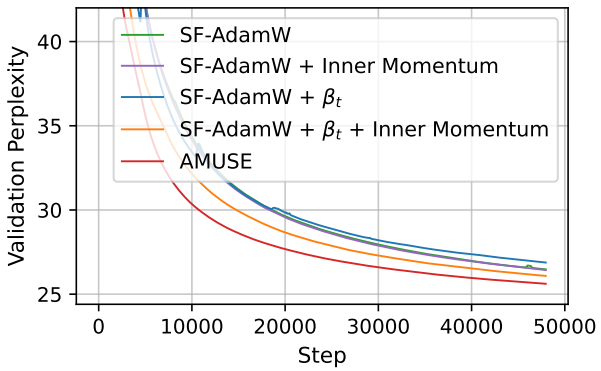


Figure 28: **Effect of β_t scheduling and momentum in SF-AdamW.** We evaluate whether the modified β_t schedule used in AMUSE can improve SF-AdamW in 124M Llama pretraining. Applying only the β_t schedule ($\beta_{\text{init}} = 0.6$, $\rho = 0.6$) degrades performance relative to the original SF-AdamW. However, when combined with AdamW's first-moment ($\beta_1 = 0.9$), the β_t schedule improves validation perplexity throughout training, benefiting both the early and late training stages. Hyperparameters were tuned separately for the schedule variant.

F.8 Robustness Across Random Seeds

We repeat 124M Llama pretraining for 16K iterations with five random seeds for each method. All hyperparameters are fixed across seeds, with no per-seed retuning, and each run is evaluated on the full evaluation set. Table 26 reports the mean and standard deviation across seeds. AMUSE outperforms both baselines in all five matched-seed runs, reducing mean validation perplexity by 0.164 relative to Muon and by 0.557 relative to SF-Muon.

Table 26: Validation perplexity of 124M Llama models after 16K pretraining iterations, evaluated on the full evaluation set. Values are mean $\pm$ standard deviation across five random seeds; lower is better.

Method	Validation PPL ↓
SF-Muon	29.073 ± 0.041
Muon	28.680 ± 0.042
AMUSE	$\mathbf{28.516 \pm 0.041}$

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer:[Yes]

   Justification: The abstract and introduction clearly outline our main contributions, including the proposal of the AMUSE optimizer and its motivation, which are fully supported by our empirical findings.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss the limitations of our approach in the Implementation Details (Section 5.4) and Conclusion (Section 6). Specifically, we address the introduction of the new hyperparameter ρ and the computational overhead of maintaining the averaged iterates.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: This paper does not contain formal theorems or lemmas requiring mathematical proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: We provide all necessary details for reproducibility, including model architectures, dataset specifications, and the exact hyperparameter configurations in Section 5.4 and Appendix D and E. The core algorithm is also summarized in Algorithm 1.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in

some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes]

   Justification: Our implementation and usage instructions are publicly available at https://github.com/kjeiun/amuse.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://neurips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes]

   Justification: We specify all experimental configurations, including dataset details (e.g., FineWeb-100B), model architectures, and comprehensive hyperparameter tables for all baselines in Appendix D and E.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: Most image-domain results are averaged over multiple independent runs and reported with standard deviations or shaded error bands. We additionally report mean and standard deviation across five random seeds for 124M LLM pretraining (Appendix F.8). Other LLM pretraining results are reported as single runs.

   Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: Detailed information regarding the hardware resources used for our experiments, including the specific GPU types and memory is provided in Appendix C.4.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: Our research adheres to the NeurIPS Code of Ethics. The study focuses on optimization algorithms and does not involve human subjects, sensitive private data, or any ethically controversial practices.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

Answer: [N/A]

Justification: This work focuses on foundational optimization algorithms and does not have a direct societal impact. As a general-purpose mathematical tool for training neural networks, any potential impact would depend entirely on the specific application it is used for.

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

Answer: [N/A]

Justification: This paper proposes a general-purpose optimization algorithm and does not release high-risk assets such as large-scale pre-trained models or sensitive datasets that would require specific safeguards.

Guidelines:

- The answer [N/A] means that the paper poses no such risks.
- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [Yes]

Justification: We properly cite the original creators of the datasets and baseline models used in our study. All assets are used in compliance with their respective open-source licenses.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes]

Justification: We introduce the AMUSE optimizer as a new asset. A complete description of the algorithm is provided in the main text, and documented implementation code is included in the supplemental material for reproducibility.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A]

Justification: This research does not involve crowdsourcing nor research with human subjects.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

Answer: [N/A]

Justification: This research does not involve human subjects, and therefore does not require IRB approval or equivalent ethical reviews.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

Answer: [N/A]

Justification: LLMs were not used as a component of the core methodology or for generating original research results. Their usage was limited to standard writing and editing assistance, which does not require declaration according to the NeurIPS guidelines.

Guidelines:

- The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.