Replacing Normalization in Deep Transformers with Lipschitz and Manifold Geometry

Rahul Bir
UC Berkeley
rahul.bir@berkeley.edu

Micah Mok
UC Berkeley
micahmok@berkeley.edu

Devan Perkash
UC Berkeley
devanp@berkeley.edu

Ethan Stone
UC Berkeley
ethanstone@berkeley.edu

Abstract

Transformers typically rely on normalization layers to stabilize training and prevent activation scale from compounding across depth. Recent work has explored intrinsically stable alternatives, including architectural modifications that enforce Lipschitz bounds on transformer blocks, as well as geometry-aware optimizers like Muon and Manifold Muon that adapt parameter updates to the geometry of each layer and constrain weight matrices to specific manifolds during optimization. In this paper, we examine whether these techniques can effectively replace normalization in deep autoregressive Transformers. Using a controlled in-context learning least-squares task, we evaluate models trained without RMS normalization and compare the stability and convergence properties of AdamW, Muon, and Manifold Muon under Lipschitz architectural modifications. Our findings show that Lipschitz constraints alone are sufficient for stable norm-free training at small scale, while Manifold Muon exhibits superior robustness under extreme learning-rate perturbations, suggesting potential benefits for large-scale regimes where stability is the primary bottleneck.

1 Introduction

Stability remains one of the central challenges in training deep Transformers. As signals propagate through many residual layers, activation scales can grow unpredictably, leading to sensitivity to hyperparameters and unstable training. In practice, normalization layers such as RMSNorm are used to bandage over these activation scales, and their presence has become deeply embedded in the standard Transformer architecture.

Recent work in geometry-aware optimization and Lipschitz-motivated architectural design suggests that explicit normalization layers may not be fundamentally required for stable training. Constraining weight matrices to structured manifolds and modifying attention and MLP blocks to have bounded sensitivity provide intrinsic mechanisms for limiting activation growth. These strategies aim to stabilize deep Transformers through their underlying geometry and architecture, rather than relying on normalization as an external corrective mechanism.

In this work, using an in-context learning least-squares regression task, we investigate whether a deep Transformer can be trained stably without any normalization layers. We examine how Lipschitz-inspired architectural modifications affect stability, and we evaluate how the training dynamics of geometry-aware optimizers Manifold Muon and Muon and the explicit manifold weight constraints of Manifold Muon compare to AdamW.

Our study is guided by two questions:

(1) Can intrinsic architectural constraints and manifold-aware optimization reliably replace explicit normalization layers in deep Transformers?

(2) Do these mechanisms improve training stability across hyperparameter regimes?

We find that Lipschitz-inspired architectural constraints alone are sufficient for stable norm-free training under AdamW and Muon at small scale, while Manifold Muon—despite slower convergence—exhibits strong robustness under extreme hyperparameters. These results provide evidence that explicit normalization is not strictly required for stability in controlled settings and suggest that geometric constraints may offer a viable stabilization mechanism for larger models. Finally, we observe that geometry-aware optimizers such as Muon and Manifold Muon display training dynamics reminiscent of grokking as seen by A. Power et al. [11], with extended plateaus followed by delayed improvement, hinting at potential benefits for generalization.

2 Related Work

Recent work on intrinsically stable deep learning has developed a unified framework connecting architectural Lipschitz constraints with geometry-aware optimization. The Modular Norm framework introduced a way to bound the sensitivity of deep networks by analyzing how individual modules compose, motivating architectural interventions such as residual reparameterization and attention scaling that later appeared in Lipschitz Transformers. Modular Duality extended this perspective by formalizing how gradients should be mapped from dual space back into parameter space under the appropriate geometry, laying the foundation for geometry-aligned optimizers. Building on this theory, optimizers such as Muon and Manifold Muon adapt parameter updates according to operator-norm geometry or by constraining weights to lie on matrix manifolds, respectively. Together, these approaches represent a coherent research direction aimed at achieving intrinsic training stability without relying on explicit normalization layers.

2.1 Geometry-Aware Optimization

Muon (2024). The Muon optimizer proposes a distinct mechanism for stabilizing training by controlling the induced rms–rms norm of updates and shaping update geometry through normalized, often orthogonalized, gradient directions. By regulating the magnitude and orientation of parameter updates, Muon aims to prevent updates from disproportionately amplifying certain directions in weight space. This approach offers a conceptual alternative to architectural stabilizers by attempting to maintain favorable geometric properties of weights throughout training.

Scalable Optimization in the Modular Norm (2024). This work treats a neural network as a sequence of modules, where each module has its own norm and sensitivity that depend on its position within the network and how its outputs feed into subsequent modules. By composing these position-dependent module norms, the authors define the modular norm, a single norm on updates to the entire network that reflects how perturbations propagate through all downstream computations. This global perspective allows them to derive depth-aware scaling rules that keep the overall network update well-conditioned, preventing modules at different depths from receiving disproportionately large or small updates. The result is an optimization framework that stabilizes training purely through the structure and composition of the network's modules.

Modular Duality in Deep Learning (2025). This work develops a framework in which each layer of a neural network is associated with a primal space, a dual space, and a geometry-specific duality map that converts gradients from the dual space back into valid parameter updates in the primal space. The authors show that standard optimization implicitly assumes a Euclidean duality map, which can be inappropriate for layers whose natural geometry is non-Euclidean. By identifying the correct duality map for each module—effectively a "type check" on gradient flow—they derive update rules that are consistent with the module's underlying geometry. This perspective leads directly to the Muon optimizer, which applies geometry-aware updates by using the appropriate duality map for each layer.

Training Transformers with Enforced Lipschitz Bounds (2025). This work investigates whether Transformers can be trained effectively when their Lipschitz constants are explicitly bounded throughout training. The authors study the tradeoff between enforcing small Lipschitz bounds and maintaining model performance, showing how stricter control of operator norms affects expressivity and stability. To enforce these bounds adaptively, they introduce two weight-norm constraint methods-spectral soft cap and spectral hammer—which limit the singular values of linear layers without fully projecting onto a spectral norm ball.

Manifold Muon / Modular Manifolds (2025). Manifold Muon extends this approach by constraining optimization to the Stiefel manifold, thereby maintaining orthonormal columns in weight matrices throughout training. The work introduces a dual ascent procedure to enforce these manifold constraints efficiently. By preserving orthonormality, the method seeks to avoid conditioning pathologies that otherwise accumulate in deep models. While the authors discuss how such geometric constraints may reduce the need for some architectural stabilizers, this idea remains primarily theoretical, with limited empirical validation in settings where stabilizers are intentionally removed. Our work contributes empirical evidence in this direction.

Fantastic Pretraining Optimizers & Where to Find Them (2025). This study evaluates optimizer performance across large-scale pretraining regimes and finds that matrix-based or geometry-inspired optimizers exhibit diminishing relative advantages as model size increases. Although not focused on stability specifically, the findings underscore that optimizer behavior interacts strongly with model scale and architecture. Because our experiments evaluate geometry-aware optimizers in a controlled small-scale setting, this work provides useful context for understanding how optimizer structure may interact with model structure when stabilizers are limited or absent.

2.2 Transformer Architectural Components

GLU Variants Improve Transformer (2021). This work introduces gated linear units and demonstrates that GLU-style activations can improve the expressivity and optimization behavior of transformer feedforward networks. The key idea is to modulate one transformed pathway with another through elementwise gating, providing smoother control over activation magnitudes than non-gated activations such as ReLU or GELU. GLU and its variants (e.g., SwiGLU) are now widely adopted in modern transformers, including the LLaMA family. Their gating mechanism has been shown to improve gradient flow and parameter efficiency, which indirectly contributes to more stable optimization. Because our architectures adopt SwiGLU feedforward layers, this work underlies one of the core activation choices in our experimental setup.

RoFormer / RoPE (2023). Rotary Position Embedding (RoPE) introduces a method of encoding positional information by applying complex-plane rotations to query and key vectors in attention mechanisms. Unlike absolute positional encodings added to embeddings, RoPE incorporates positional information directly into the vector geometry, preserving the inner-product structure of attention while enabling relative position handling. This formulation allows attention patterns to naturally incorporate directional relationships without modifying activation magnitudes or relying on large positional embeddings. RoPE has become a standard design choice in contemporary LLMs due to its simplicity and strong empirical performance, and is used in our architecture as well.

Llama Family Overview (2024). This survey summarizes architectural patterns used in the LLaMA models, highlighting components such as pre-layer normalization, SwiGLU activations, RoPE position embeddings, and grouped-query attention. Importantly, the LLaMA models employ pre-layer normalization, a widely adopted technique for improving optimization stability in deep transformers by normalizing activations before each sublayer rather than after. This design is closely tied to stable and efficient training. Because our Type1 architecture includes these standard stabilizing components, while Type2 removes them entirely, the LLaMA configuration provides a modern reference point for what “stabilized” transformer training looks like in practice.

3 Background

In this section, we go over background information necessary for the rest of the paper.

3.1 Manifold Muon

The Manifold Muon optimizer, as first proposed by Jeremy Bernstein, constrains the weight matrices $\mathbf{W}$ to have orthonormal columns, existing on the Stiefel manifold $V_{k,n}$, and performs each step by minimizing the loss approximation subject to orthonormality and a step-size constraint. This is solved by introducing a Lagrange multiplier $\mathbf{\Lambda}$ (here found via Online Dual Ascent) which defines a spectrally restricted descent direction $\mathbf{A}$. Finally, the new weights are projected back onto the manifold using a Retraction (the matrix sign function) to preserve the constraints, which ultimately introduce architectural stability by enforcing spectral norm bounds.

The algorithm of a manifold muon optimizer step is outlined below:

1) Start with the gradient $\mathbf{G}$, the current orthonormal weights $\mathbf{W}$, and Lagrange Multiplier $\mathbf{\Lambda}$.

2) Direction Calculation (Dual Problem Solution): Solve for $\mathbf{\Lambda}$. Use $\mathbf{\Lambda}$ to calculate the descent direction $\mathbf{A}$, by taking the matrix sign function of $\mathbf{G} + 2\mathbf{W\Lambda}$.

$$\mathbf{A} = \text{msign}(\mathbf{G} + 2\mathbf{W\Lambda})$$

3) Primal Update and Retraction: Calculate the intermediate weights $\mathbf{W}_{\text{temp}} = \mathbf{W} - \eta\mathbf{A}$. Then, Retract $\mathbf{W}_{\text{temp}}$ back onto the Stiefel manifold using the matrix sign function, ensuring the new weights $\mathbf{W}_{t+1}$ maintain the orthonormality constraint $\mathbf{W}_{t+1}^T\mathbf{W}_{t+1} = \mathbf{I}$.

$$\mathbf{W}_{t+1} = \text{msign}(\mathbf{W}_{\text{temp}})$$

The full algorithm can be found in Appendix 8.1

3.2 Lipschitz bounds.

A function f is said to be L-Lipschitz if its output cannot change faster than L times the change in its input. Formally, under a norm $\|\cdot\|$, the function f is L-Lipschitz if

$$\|f(x) - f(y)\| \leq L\,\|x - y\| \qquad \text{for all } x, y.$$

The constant L measures the worst-case sensitivity of the function; small Lipschitz constants prevent the amplification of perturbations, while large ones allow outputs to change rapidly with small input differences. In deep networks, uncontrolled growth of Lipschitz constants across layers can lead to exploding activations and instability.

In this work, we focus on Lipschitz bounds with respect to the $\|\cdot\|_{\infty,\text{RMS}}$ norm, which measures the maximum RMS activation magnitude across tokens. For simple linear layers in our network the token-wise RMS norm is simply a scaled version of the standard ℓ_2 norm. Consequently, if a weight matrix is constrained to lie on the Stiefel manifold (having orthonormal columns), it preserves the ℓ_2 norm of each token vector and therefore preserves its RMS norm as well. Taking the maximum across tokens implies that such token-wise linear projections are exactly 1-Lipschitz under the $\|\cdot\|_{\infty\text{RMS}}$ norm.

We emphasize that this argument applies only to the token-wise linear components of the network. Operations such as attention, which mix information across tokens, require separate Lipschitz analyses that can be found in Large et al. [7]

4 Methods

This section describes the architecture, dataset, training setting, and optimizers. Our code can be found at https://github.com/RahulBirCodes/bluey.git.

4.1 Task and Dataset

The goal for the model is to solve least squares regression problems of where the dimension of the X and Y are $D = 5$.

For each batch, we generate $X \in \mathbb{R}^{B\times T\times 5}$, where B is the batch size and T is the number of (x, y) pairs per sequence. For every batch element b we sample $X_b \in \mathbb{R}^{T\times 5}$, where each row

$x_{b,t} \in \mathbb{R}^5$ is sampled with i.i.d. standard normal entries. We resample X_b until its condition number is below a fixed threshold $\kappa_{\max} = 10^3$, ensuring that each least-squares problem is numerically well-conditioned. We also sample a matrix $W_b \in \mathbb{R}^{5\times 5}$ with entries $\sim \mathcal{N}(0, 1/5)$ and define $y_{b,t} = W_b x_{b,t}$ such that $Y = XW$ for $Y \in \mathbb{R}^{B\times T\times 5}$ and $W \in \mathbb{R}^{B\times 5\times 5}$.

Each training sequence is then constructed by interleaving x- and y-tokens for a given batch element. For a sequence with T pairs we create $2T$ positions and assign the (x_t, y_t) pairs either in the order $x_1, y_1, x_2, y_2, \ldots, x_T, y_T$ or in the order $y_1, x_1, y_2, x_2, \ldots, y_T, x_T$, with the choice made independently and uniformly at random for each sequence. This random swapping of the (x, y) order for each pair removes trivial positional symmetries.

Each token is a vector in $\mathbb{R}^{13}$. The layout of a single token is

$$[\text{x\_flag}, \text{y\_flag}, x_1, \ldots, x_5, y_1, \ldots, y_5, 1]$$

where:

- x_flag $\in \{0, 1\}$ is 1 if the token encodes an x-input and 0 otherwise,
- y_flag $\in \{0, 1\}$ is 1 if the token encodes a y-target and 0 otherwise,
- the next five coordinates store $x \in \mathbb{R}^5$ when x_flag $= 1$ and are 0 otherwise,
- the following five coordinates store $y \in \mathbb{R}^5$ when y_flag $= 1$ and are 0 otherwise,
- the final coordinate is always 1 and acts as an explicit bias feature.

Thus the batch of input tokens has shape tokens $\in \mathbb{R}^{B\times 2T\times 13}$.

During training we add small Gaussian noise to the input x-coordinates to encourage robustness. For each x-token we sample $\varepsilon_{b,t} \sim \mathcal{N}(0, 0.01^2 I_5)$ and replace the stored $x_{b,t}$ by $x_{b,t} + \varepsilon_{b,t}$ in the token. The underlying targets Y and the y-coordinates used for supervision remain noise-free; the noise is purely an input perturbation for the model.

The model is trained over the full sequence, but we only use its outputs at positions where the input token is an x-token. This relationship is summarized as follows:

Given $x_{b,t} \in \mathbb{R}^5$, the model predicts $\hat{y} = f_\theta(x)$, and the training objective is the mean-squared error

$$\mathcal{L}(\theta) = \|f_\theta(x) - y_{b,t}\|_2^2.$$

For every epoch, the matrices X, Y, and W, as well as the sequence orderings and the input noise, are freshly resampled.

4.2 Model Architecture

We construct a modern autoregressive, decoder transformer network with:

- SwiGLU feedforward blocks,
- RoPE positional encodings.

Our model is a 15-layer Transformer with a hidden size of 256. Each layer consists of a Multi-Head Attention module with 8 heads, followed by an MLP with hidden size 256. No explicit normalization is used in any part of the network. The MLP employs a SwiGLU activation.

We use a custom initialization for the embedding matrix and randomly sample each entry i.i.d from $\sim \mathcal{N}(0, 1/7)$ to ensure that our initial embeddings are RMS norm 1. A proof of this can be found in Appendix 8.2.

4.3 Lipschitz-Inspired Architectural Modifications

Following the Lipschitz Transformer design of Newhouse et al. [14], we incorporate several architectural modifications that bound the sensitivity of each block under the $\|\cdot\|_{\infty,\mathrm{RMS}}$ norm (the maximum RMS norm across tokens). These changes aim to prevent uncontrolled amplification of activations in the absence of normalization layers.

Convex Residual Reparameterization. The standard residual update $x + \mathrm{block}(x)$ can double the Lipschitz constant at every layer, even when the block itself is 1-Lipschitz. To mitigate this compounding effect, we adopt the convex residual parameterization introduced by Large et al. [7]:

$$x \leftarrow \frac{N-1}{N}x \; + \; \frac{1}{N}\,\mathrm{block}(x),$$

where N is the total number of layers. If the block is 1-Lipschitz under the $\|\cdot\|_{\infty,\mathrm{RMS}}$ norm, then the reparameterized residual connection is also 1-Lipschitz. While this bound is not strict in deep networks once activation norms deviate from 1, the transformation substantially limits Lipschitz growth across depth and empirically stabilizes training in the absence of normalization.

Attention Scaling. We also follow the Lipschitz attention formulation proposed by Large et al. In contrast to the standard Transformer's $1/\sqrt{d}$ scaling, functional attention becomes 1-Lipschitz under the $\|\cdot\|_{\infty,\mathrm{RMS}}$ norm when the dot-product is scaled by $1/d$:

$$\mathrm{softmax}\left(\frac{QK^\top}{d}\right)V.$$

To ensure unit sensitivity when attention is viewed as a function of the triplet (Q, K, V), Large et al. multiply the output by an additional factor of $1/3$. We adopt the same $1/3$ scaling in our implementation.

SwiGLU Scaling (Empirical Lipschitz Control). Rather than deriving an explicit Lipschitz bound for the SwiGLU activation and computing the exact scaling needed to make it 1-Lipschitz, we instead introduce a simple empirical scaling factor of $\frac{1}{3}$ that worked well in our experiments.

Orthogonal Initialization. To avoid initial numerical instability that can propagate catastrophically through a deep norm-free network, we initialize all linear weight matrices (except the embedding and unembedding layers) with orthonormal matrices.

4.4 Optimizers and Respective Hyperparameter Values Compared

We evaluate three optimizers, each with slightly different sets of hyperparameters

Table 1: Hyperparameter search ranges for each optimizer.

Optimizer	Learning rate	(β_1, β_2) - Fixed	Weight decay	Batch size
AdamW	$\{10^{-4}, 10^{-3}, 10^{-2}\}$	$(0.9,\ 0.999)$	$\{0.0,\ 0.01,\ 0.1\}$	$\{64,\ 256\}$
Muon	$\{10^{-4}, 10^{-3}, 10^{-2}\}$	$(0.95,\ 0.999)$	$\{0.0,\ 0.01,\ 0.1\}$	$\{64,\ 256\}$
Manifold Muon	$\{0.1,\ 0.2\}$	$(0.95,\ 0.999)$	0.01	256

Rationale Behind Hyperparameter Decisions. For AdamW, we use a standard grid of learning rates, weight decays, and batch sizes commonly adopted in Transformer training, following the recommendations of Loshchilov and Hutter [10]. These settings provide a familiar and well-studied baseline for adaptive optimization.

For Muon, we reuse the same grid to maintain a controlled comparison with AdamW. Prior work on Muon [1] and its large-scale extensions [9] recommends fixing the momentum parameter at $\beta_1 = 0.95$ and using a fixed number of Newton–Schulz steps; we follow these conventions.

For Manifold Muon, we acknowledge that we do not conduct a full hyperparameter sweep. Due to compute constraints, and motivated by the Modular Manifolds framework [2]—which emphasizes the optimizer's stability at relatively large learning rates—we restrict our exploration to a small set of higher learning rates rather than performing an exhaustive search. Additionally, because Manifold Muon often exhibits prolonged plateau phases resembling grokking, we train it for substantially longer (20k steps) than AdamW and Muon (3k steps), which typically converge or plateau much

Table 2: Best Optimizer Hyperparameters under Lipschitz Modifications and No Normalization

Optimizer	Learning rate	Batch size	Weight Decay	Loss
AdamW	1e-3	256	0.01	0.16656
Muon	1e-3	256	0.01	0.75127
Manifold Muon	0.1	256	N/A	1.96396

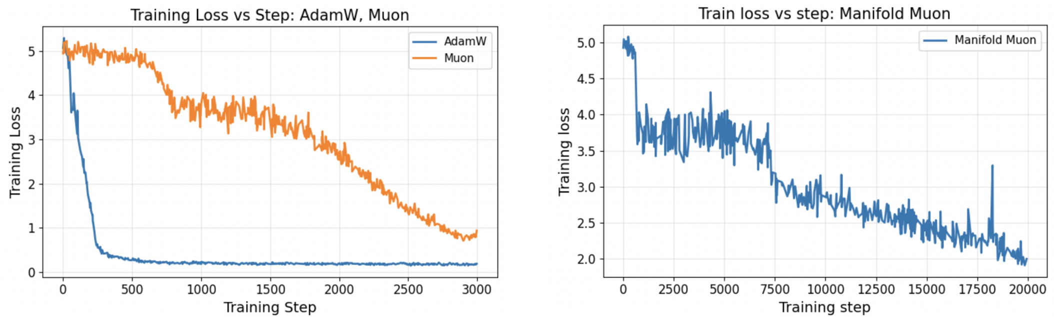


Figure 1: Optimal AdamW/Muon Optimizers Outperform Manifold Muon, as Seen in Table 2

earlier in our experiments. While this results in unequal training horizons across optimizers, it ensures that each method is evaluated in a regime where its characteristic behavior is observable, enabling a fair qualitative comparison of stability and convergence dynamics.

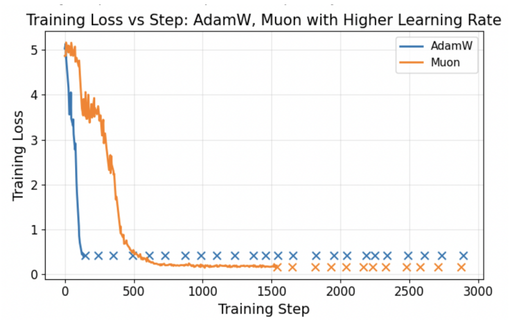
5 Results

Under carefully chosen hyperparameters, we are able to train a deep autoregressive Transformer without any normalization layers or logit capping, relying solely on the intrinsic Lipschitz architectural constraints introduced in Section 4. AdamW and Muon train stably and converge rapidly in this norm-free setting, indicating that Lipschitz modifications alone are sufficient to prevent activation blow-up at this scale (Figure 1). In contrast, Manifold Muon performs suboptimally on our small algorithmic task: although it eventually converges, it does so significantly slower than the unconstrained optimizers (Figure 1).

Although AdamW exhibits small activation RMS norms in our experiments, we attribute this primarily to its rapid convergence: the model reaches a low-loss regime so quickly that activations have little opportunity to accumulate scale. In contrast, Muon—whose loss decreases much more slowly—shows significantly larger activation norms, suggesting that norms can grow substantially during extended optimization when weights are not explicitly constrained (Figure 3). This pattern indicates that while explicit manifold constraints may be unnecessary for small models trained to convergence quickly, they could become important at larger scales or longer training horizons, where activation growth is more likely to compound. Consistent with this view, Manifold Muon displays exceptional resilience to large learning rates, remaining stable in settings where AdamW and Muon diverge immediately. Although its convergence is slower, this robustness may make it valuable in regimes where stability is the primary bottleneck.

5.1 The ”Norm-Free” Barrier

We observe that removing RMSNorm from the network layers leads to immediate training instability, regardless of whether AdamW, Muon, or Manifold Muon is used. In the absence of normalization, training runs diverge catastrophically, with the loss reaching NaN immediately on the very first loss calculation. This failure mode arises because the residual stream is no longer scale-controlled: without normalization, activation variance compounds across depth, and even small deviations in scale can grow exponentially, eventually causing numerical overflow in the forward pass. We therefore confirm that enforcing these per-block Lipschitz constraints is a necessary condition for achieving stable training in the norm-free setting.



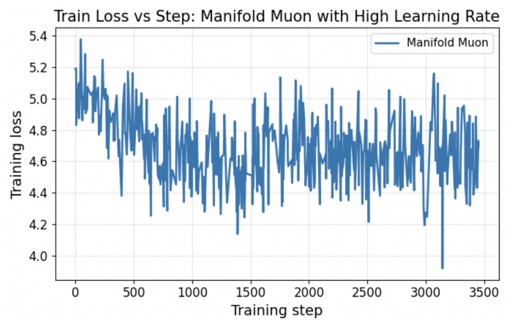
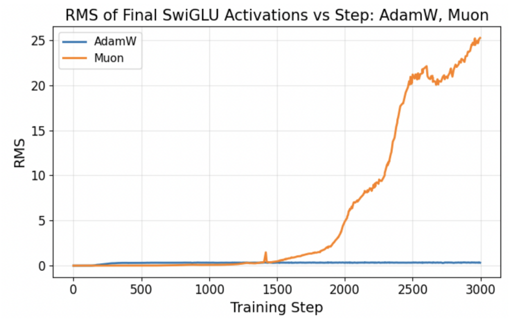


Figure 2: AdamW/Muon are Unstable at Learning Rate 0.01, but Manifold Muon Remains Stable even at Learning Rate 1



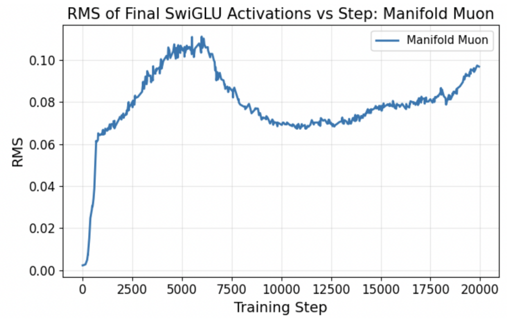


Figure 3: RMS Norms on Activations after the Final SwiGLU Layer. Muon has significantly larger activation norms than both AdamW and Manifold Muon

5.2 Stable Training Across Learning-Rates with Manifold Constraint

Although Manifold Muon converges more slowly, its training dynamics remain stable in the norm-free setting even when the learning rate is increased by large factors. In contrast, both vanilla AdamW and unconstrained Muon exhibit rapid activation blow-up under the same conditions, reflecting the absence of any structural control on their weight matrices. The stability of Manifold Muon suggests that enforcing manifold constraints yields robustness across wider hyperparameter regimes and reduces the need for delicate tuning—an appealing property for scaling to larger models, where training stability often becomes the primary bottleneck. As shown in Figure 1, all optimizers admit a set of well-behaved hyperparameters under careful tuning; however, when we scale each optimizer's best-performing learning rate by a factor of 10 (Figure 2), almost all runs diverge immediately. The sole exception is Manifold Muon, which continues to train stably even under this aggressive perturbation. This illustrates that geometric constraints impart significantly higher robustness compared to unconstrained optimizers.

5.3 Geometry Aware Optimizers and Grokking

During early experiments, we observed an initially puzzling behavior: Manifold Muon runs often plateaued for long durations, remaining nearly flat across most of the training horizon, while AdamW and Muon continued to decrease in loss. These dynamics closely resembled the multi-phase grokking patterns documented by Power et al. [11] and by Liu et al. [12]. Motivated by this similarity, we extended Manifold Muon training to 20k steps and found that, after a prolonged plateau, the loss eventually began a delayed descent. We speculate that, given a sufficiently long training schedule, Manifold Muon would converge to a regime comparable to AdamW and Muon.

A plausible explanation is that the Stiefel manifold constraint in Manifold Muon acts like very strong weight decay: by restricting weights to lie on the manifold, the optimizer limits the model's ability

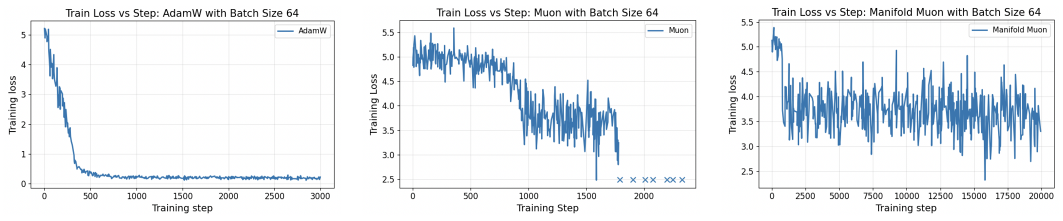


Figure 4: AdamW, Muon, and Manifold Muon with Smaller Batch Sizes (64)

to memorize via unconstrained parameter growth, forcing it to search for more efficient internal circuits before improving. This constraint may also underlie Manifold Muon's heightened sensitivity to batch size, discussed further below.

Interestingly, Muon exhibits a similar—though faster and less pronounced—plateau-and-descent pattern. This suggests that geometry-aware update rules more broadly, even without a hard manifold constraint, bias the optimization trajectory toward smoother, more generalized solutions, leading to grokking-like behavior.

5.4 Batch Size Interactions with Geometry Aware Optimizers

We observed that batch size plays a large role in the behavior of geometry-aware optimizers such as Muon and Manifold Muon. For Manifold Muon, reducing the batch size causes the loss to plateau for nearly the entire run, whereas larger batch sizes produce steady improvement (Figure 4). Muon shows a similar effect: small-batch runs plateau or even diverge, while larger batches converge reliably. In contrast, AdamW remains largely insensitive to batch size within the ranges we tested (Figure 4).

We do not have a definitive explanation for this phenomenon. However, one possibility is that the orthogonalized updates used in both Muon and Manifold Muon may take disproportionately large steps in noisy gradient directions when the batch size is small, reducing the effective learning signal and causing optimization to stall. A more detailed investigation of this effect is left for future work.

6 Ablation

To disentangle the contribution of architectural constraints versus optimization geometry, we performed a series of ablations isolating the effects of attention and MLP scaling, manifold gain parameters, and input augmentation.

6.1 The Necessity of Attention and MLP Lipschitz Scaling

During early experiments with higher Manifold Muon learning rates, we observed a recurring failure mode: activation RMS norms remained stable for many steps and then abruptly exploded at the grokking transition, causing immediate divergence to NaN. Upon investigating, we discovered that we had omitted the 1/3 scaling on the attention output introduced by Large et al. [7], as well as the analogous scaling required on the output of the SwiGLU block (Shazeer et al. [3]). Reintroducing both scalings restored stability.

Importantly, keeping only the attention scaling while removing the MLP scaling was insufficient—activations still diverged. Both forms of architectural scaling were necessary for stable training. This supports our central hypothesis: manifold constraints on the optimization trajectory are not enough on their own; they must be paired with architectural modifications that ensure each block remains Lipschitz-bounded.

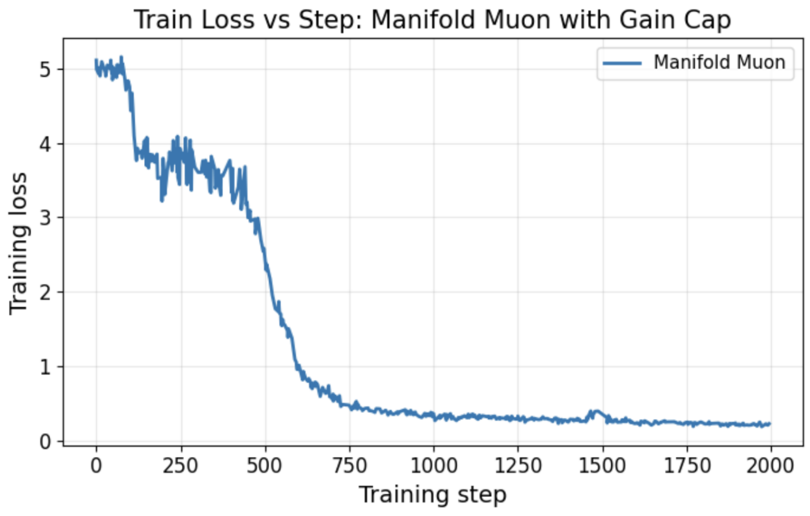


Figure 5: Manifold Muon Converges Faster with a Gain Cap of 5, to a Loss of 0.2 after 2,000 Steps

6.2 Adding a Diagonal Gain to our Stiefel Constraint

In early Manifold Muon runs, we noticed extended plateaus and suspected that the model lacked sufficient expressivity under the strict Stiefel-manifold constraint. Following Newhouse et al. [13], we RMS-normalized the embedding matrix to enforce unit-norm inputs, but we did not apply a similar normalization to the unembedding due to the regression nature of our task.

We hypothesized that forcing all singular values of every linear map to remain equal (a consequence of remaining on the Stiefel manifold) restricted the model's ability to create directional amplification. To relax this, we introduced a trainable 1D gain vector, expanded into a diagonal matrix in the forward pass and multiplied with each Stiefel-constrained linear map. The gains were capped by a tunable hyperparameter.

This modification accelerated convergence (Figure 5), but at a cost: Manifold Muon became significantly more sensitive to hyperparameters, losing its characteristic stability in high–learning-rate regimes. Moreover, the loss plateau persisted unless we also removed the RMS normalization we had been applying to the embedding matrix on each optimization step. These findings suggest a nuanced trade-off: diagonal gains improve expressivity but weaken the stability benefits of strict manifold optimization. In our final results, we remove RMS normalization on the embedding matrix.

6.3 Input Augmentation and Noise

To investigate whether limited expressivity contributed to Manifold Muon's slower convergence, we tested two additional modifications. First, we appended a constant 1 to each input vector, giving the model access to an implicit bias term despite our no-bias architecture. Second, we injected small noise into the input in the hope that, during the retraction step of Manifold Muon, any "noise" component of the update would be preferentially discarded, preserving meaningful signal and potentially accelerating training.

Although our main experiments were run with both augmentations enabled, neither modification produced a meaningful change in convergence speed in later experiments.

7 Conclusion

We set out to test whether geometry-aware optimization and intrinsic Lipschitz constraints can replace explicit normalization layers in deep Transformers. In a controlled in-context least-squares setting, we demonstrated that a 15-layer autoregressive Transformer can be trained stably without any normalization or logit capping, so long as each block is modified to be approximately 1-Lipschitz under simple architectural scalings. Under these conditions, standard optimizers such as AdamW and Muon train reliably and reach low loss, indicating that explicit weight normalization is not strictly necessary for stability at this scale.

Our results also highlight a complementary role for manifold-based optimization. Although Manifold Muon converges more slowly, it exhibits remarkable robustness to aggressive learning-rate increases that cause AdamW and Muon to diverge immediately. This suggests that constraining weight geometry can enlarge the basin of stable hyperparameters, trading off convergence speed for substantially greater resilience in the norm-free regime.

Taken together, our findings support a view in which architectural Lipschitz constraints supply the primary mechanism preventing activation blow-up, while manifold-constrained optimization provides additional robustness to hyperparameter choices and long training horizons by explicitly enforcing these constraints throughout training. Notably, the “grokking”-like behavior observed under Manifold Muon may hint at improved generalization properties, though we leave a systematic investigation of this effect to future work. These results point to a promising pathway toward scalable norm-free Transformers and motivate further investigation at larger model sizes, where training stability is often the dominant bottleneck.

References

[1] Keller Jordan. *Muon: Orthogonal, geometry-aware updates for stable training*. 2024.

[2] Thinking Machines Lab. *Modular Manifolds / Manifold Muon*. 2025.

[3] Noam Shazeer et al. *GLU Variants Improve Transformer*. arXiv:2002.05202, 2021.

[4] Jianlin Su et al. *RoFormer: Enhanced Transformer with Rotary Position Embedding*. arXiv:2104.09864, 2023.

[5] “The Llama Family of Models: Model Architecture, Size, and Scaling Laws”. 2024.

[6] Laker Newhouse. *Duality, Weight Decay, and Metrized Deep Learning*. 2025.

[7] Tim Large, Jeremy Bernstein et al. *Scalable Optimization in the Modular Norm*. arXiv:2405.14813, 2024.

[8] Kaiyue Wen et al. *Fantastic Pretraining Optimizers & Where to Find Them*. arXiv:2509.02046, 2025.

[9] Jingyuan Liu et al. *Muon is Scalable For LLM Training*. arXiv:2502.16982, 2025.

[10] Ilya Loshchilov and Frank Hutter. *Decoupled Weight Decay Regularization*. arXiv:1711.05101, 2017.

[11] A. Power et al. *Grokking: Generalization Beyond Overfitting on Small Algorithmic Datasets*. 2022.

[12] J. Liu et al. *OmniGrok: Grokking Beyond Algorithmic Data*. 2024.

[13] Laker Newhouse, R. Preston Hess, Franz Cesista et al. *Training Transformers with Enforced Lipschitz Bounds*. arXiv:2507.13338, 2025.

[14] Jeremy Bernstein and Laker Newhouse. *Modular Duality in Deep Learning*. arXiv:2410.21265, 2024.

8 Appendix

8.1 Manifold Muon Algorithm

A manifold is a subspace of a higher-dimensional space that maintains certain geometric properties, but at a close enough proximity, appears like a flat plane. The Manifold Muon optimizer constrains all of the columns in the weight matrices $\mathbf{W} \in \mathbb{R}^{n \times k}$ to be on the Stiefel manifold $V_{k,n}$— a subspace that requires all columns to be orthonormal and where all the singular values are equal to 1. This is formalized by the equation:

$$\mathbf{W}^T\mathbf{W} = \mathbf{I}_k$$

where $\mathbf{I}_k$ is the $k \times k$ identity matrix.

The Manifold Muon optimizer updates the weight matrices by solving a dual constrain optimization problem: finding the update U that minimizes the loss approximation and has spectral norm less than η, and also being tangent to the manifold. The objective is defined as:

$$\min_{\mathbf{U}} \left(\mathrm{Tr}(\mathbf{A}^T\mathbf{G}) + \frac{1}{2\eta}||\mathbf{A}||_F^2 \right) \quad \text{subject to } \mathbf{A} \in T_{\mathbf{W}}V_{k,n}$$

Here, $\mathbf{G} = \nabla_{\mathbf{W}}\mathcal{L}$ is the Euclidean gradient of the loss $\mathcal{L}$ with respect to $\mathbf{W}$, and the constraint $\mathbf{A} \in T_{\mathbf{W}}V_{k,n}$ ensures the update is tangent to the manifold.

In order to solve the optimization problem, the authors of the original Modular Manifold essay [Bernstein et al.] introduce the matrix Λ, the Lagrange multipliers subject to the tangent space constraint. By forming the Lagrangian and setting its derivative with respect to A to zero, the optimal update $\mathbf{A}$ (the tangent update) is found to be:

$$\mathbf{A} = -\eta(\mathbf{G} - \mathbf{W}\mathbf{\Lambda})$$

The term $\mathbf{W}\mathbf{\Lambda}$ effectively projects the Euclidean gradient $\mathbf{G}$ onto the tangent space, yielding the Riemannian gradient component of the update. Finding the optimal $\mathbf{\Lambda}$ matrix involves solving a second equation, which can be computationally expensive.

The matrix update A is constrained to be orthonormal via the matrix sign function. Once an optimal update direction is found, the new W matrix is calculated by adding the update matrix A multiplied by the learning rate η. The following result is then retracted back onto the manifold again via the matrix sign function.

Instead of calculating and solving for Λ in every manifold muon call, we implement an online version of the manifold muon optimizer by storing the candidate Λ matrix after one call of the manifold optimizer step, and re-using that Λ matrix and improving it in the next optimizer step with the following equation: $\mathbf{\Lambda} = \mathbf{\Lambda} - \alpha\mathbf{H}$.

8.2 Embedding Initialization and Activation Scaling

Our input sequence is composed of two binary flags signalling x or y, 5 i.i.d unit Gaussian normals, and then an extra dimension to add an explicit bias feature. Thus, the total number of entries in any given input is 5 + 1 + 1. These relationships are formalized below:

$$u = [\texttt{x\_flag}, \texttt{y\_flag}, x_1, \ldots, x_{xy-size}, y_1, \ldots, y_{xy-size}, 1],$$

With s being the number of nonzero entries:

$$s = D + 1 + 1 = 5 + 1 + 1 = 7$$

For example, for an x-token we have

$$\texttt{x\_flag} = 1, \quad \texttt{y\_flag} = 0, \quad x_j \sim \mathcal{N}(0,1), \quad y_j = 0,$$

and for a y-token we have

$$\texttt{x\_flag} = 0, \quad \texttt{y\_flag} = 1, \quad y_j \sim \mathcal{N}(0,1), \quad x_j = 0,$$

The final coordinate is always 1 and acts as an explicit bias feature.

We implement our linear embedding without bias:

$$h = W_{\text{emb}}u, \qquad W_{\text{emb}} \in \mathbb{R}^{d_{\text{model}} \times d_{\text{in}}},$$

where $h \in \mathbb{R}^{d_{\text{model}}}$ is the embedded token representation. We initialize each row of W_{emb} with i.i.d. Gaussian entries,

$$(W_{\text{emb}})_{ij} \sim \mathcal{N}(0, \sigma^2),$$

and choose σ so that the *RMS norm* of h is approximately 1 at initialization.

To determine the variance of each of the entries of W, we calculate the RMS norm for each entry of the activation vector, h_k. Thus, we have:

$$h_k = \sum_{j=1}^{d_{\text{in}}} (W_{\text{emb}})_{kj} u_j.$$

Assuming $\mathbb{E}[u_j] = 0$ for the non-constant coordinates, $\mathbb{E}[u_j^2] \approx 1$ for the s active coordinates and 0 for the others, and independence between W_{emb} and u, the variance of h_k is

$$\mathrm{Var}(h_k) = \sum_{j=1}^{d_{\text{in}}} \mathbb{E}\big[(W_{\text{emb}})_{kj}^2\big]\, \mathbb{E}[u_j^2] \approx s\sigma^2.$$

Thus to achieve RMS norm 1 for each coordinate at initialization,

$$\sqrt{\mathbb{E}[h_k^2]} = 1 \quad \Longleftrightarrow \quad \mathrm{Var}(h_k) = 1,$$

we achieve

$$s\sigma^2 = 1 \quad \Longrightarrow \quad \sigma = \frac{1}{\sqrt{s}} = \frac{1}{\sqrt{D+1+1}}.$$

In code, we implement this as

```
self.embedding = nn.Linear(input_dim, hidden_size, bias=False)
nn.init.normal_(self.embedding.weight,
                mean=0.0,
                std=(xy_size + 1 + (1 if add_fake_dim else 0))**-0.5)
```