arXiv:2608.02502v1 [cs.AI] 3 Aug 2026

# CMuon: Accelerating and Stabilizing Diffusion Transformer Training via Chunked Momentum Orthogonalization

Chuyan Chen[1] Peng Sun[2,3*] Kun Yuan[1*†]

[1]Peking University [2]Westlake University [3]Zhejiang University
chuyanchen@stu.pku.edu.cn sunpeng@westlake.edu.cn kunyuan@pku.edu.cn

**Abstract.** Diffusion Transformers (DiTs) have achieved state-of-the-art (SOTA) performance in visual generative modeling, yet their training remains computationally prohibitive. While the recently proposed Momentum Orthogonalization (Muon) optimizer offers a promising alternative to AdamW, its direct application to DiTs yields suboptimal late-stage convergence. In this paper, we identify the root cause of this bottleneck: standard DiT architectures fuse functionally distinct weights (e.g., within AdaLN and QKV layers) into unified tensors for computational efficiency. Applying Muon to these fused tensors inadvertently induces implicit subspace coupling, which distorts update directions and degrades global optimization. To address this, we introduce Chunked Muon (CMuon), a simple yet highly effective strategy that partitions these matrices into independent sub-components prior to orthogonalization. Extensive experiments demonstrate that a 675M-parameter DiT trained with CMuon achieves a FID of 1.18 on ImageNet 256 in just 200 epochs. This represents more than a 2x training speedup over AdamW, while effectively overcoming the late-stage convergence plateaus of vanilla Muon.

**Keywords:** Optimization · Generative Model · Machine Learning

## 1 Introduction

Training large-scale neural networks with AdamW [10, 16] has long been the standard practice. However, the recent introduction of the Muon optimizer [9] is beginning to shift this paradigm. By orthogonalizing momentum matrices, Muon accelerates convergence and improves memory efficiency compared to AdamW. It has been rapidly adopted within the large language model (LLM) and deep learning communities [14, 32, 37], demonstrating significant speedups in large-scale pre-training [14, 32]. Building on this success, several advanced variants—such as Scion [21], Gluon [22], and PolarExpress [1]—have further generalized its capabilities. Consequently, optimization based on Linear Minimization Oracles

* These authors jointly supervised this work.
† Corresponding author.

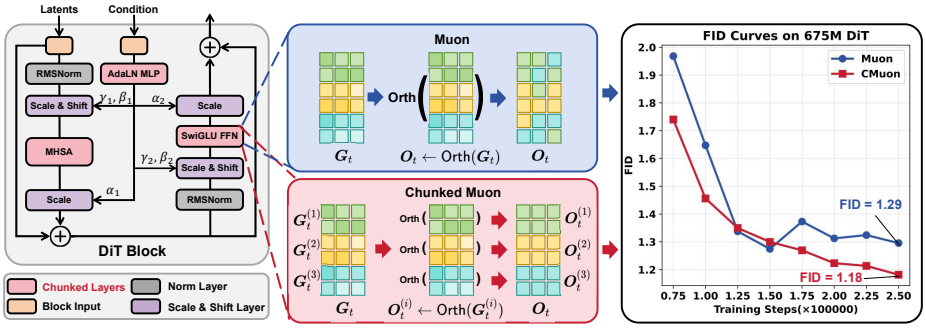



**Fig. 1: Overview and empirical validation of CMuon.** *Left:* CMuon applies momentum orthogonalization in a decoupled manner by chunking selected weight matrices in Diffusion Transformers (DiTs), replacing a single global orthogonalization with per-chunk orthogonalization. This design mitigates cross-block interference, improving optimization stability and efficiency relative to vanilla Muon. *Mid:* Illustration of our proposed CMuon. *Right:* On ImageNet at $256 \times 256$, CMuon yields a substantial performance gain over vanilla Muon when training a 675M-parameter DiT model.

(LMOs) is rapidly emerging as a powerful new standard for training foundation models.

In the realm of visual generative modeling, recent works have pioneered the application of Muon to Diffusion Transformers (DiTs) [20], such as in pixel MeanFlow [17] and HunyuanVideo [37]. While these initial efforts report an approximate $2\times$ acceleration in convergence speed during early training epochs, empirical observations also reveal a notable plateau effect: although Muon exhibits a sharply decreasing loss initially, the final Fréchet Inception Distance (FID) often merely matches that of AdamW [43]. In fully trained regimes, the definitive advantage of Muon remains ambiguous; it is unclear whether the optimizer can strictly surpass the generative quality of AdamW or merely reach the same sub-optimal baseline faster.

In this paper, we demonstrate that applying Muon naively to DiTs yields suboptimal results, failing to provide a sustained, definitive speedup over AdamW. We identify the root cause of this bottleneck as the architectural implementation of specific DiT components, namely the AdaLN, QKV, and FFN layers. In standard implementations, functionally distinct weights with independent initializations are concatenated into unified tensors for computational efficiency. Because vanilla Muon applies its Newton-Schulz orthogonalization iteratively to the entire concatenated matrix collectively, it inadvertently induces implicit coupling among these functionally disjoint sub-matrices. This parameter coupling distorts the update directions and ultimately degrades global optimization efficiency. To address this, we propose **Chunked Muon (CMuon)**, a simple yet highly effective algorithmic modification. By chunking these concatenated matrices back into their original functional sub-components and orthogonalizing their momentums independently, CMuon fundamentally circumvents the subspace

interference issue. This approach dramatically boosts convergence speed up to 2× over AdamW and, crucially, maintains this trajectory advantage throughout the later stages of training.

Furthermore, we conduct comprehensive ablation studies on key optimization hyperparameters, including matrix scaling factors, the selection of layers reserved for AdamW (e.g., embeddings and final projection layers), and the strategy for learning rate scaling. Under the optimal configuration, our 675M-parameter DiT model [20, 30], equipped with the VA-VAE latent space [40], achieves a SOTA FID of 1.18 on ImageNet $256 \times 256$ at 200 epochs (with a batch size of 1024). Notably, we attain this performance solely via the CMuon optimizer, without relying on auxiliary training acceleration techniques such as Representation Alignment (REPA) [41]. **In summary, our contributions are threefold:**

(a) We reveal that naively applying Muon to DiTs yields suboptimal convergence, and we trace this bottleneck to the unintended orthogonalization coupling of functionally disjoint parameters concatenated within AdaLN and QKV layers.
(b) To alleviate this limitation, we introduce CMuon, a simple and effective algorithm that chunks these unified tensors prior to orthogonalization. This eliminates implicit cross-subspace coupling, requires only minor code modifications in the optimizer, and introduces negligible computational overhead.
(c) Extensive experiments demonstrate that CMuon significantly accelerates the FID reduction rate compared to both AdamW and vanilla Muon. For a 675M DiT model, we achieve a 2× training speedup—an advantage that persists throughout the entire training process. Furthermore, our ablation studies verify the robustness of CMuon, showing consistent improvements over vanilla Muon across various hyperparameter configurations.

## 2 Related Works

### 2.1 Diffusion Transformers for Image Generation

The integration of transformer architectures into diffusion models [20, 34] has redefined the frontier of visual generative modeling, significantly elevating the upper bound of synthesis quality and architectural scalability. This breakthrough has spurred the development of highly efficient variants [18,35,40]. To manage the computational demands of high-resolution image generation, modern approaches predominantly adopt the latent diffusion paradigm [11, 23, 39]. By utilizing advanced Variational Autoencoders (VAEs)—such as SD-VAE [23], VA-VAE [40], and REPAE [12]—images are compressed into continuous, lower-dimensional latent spaces. Performing diffusion or flow-based modeling within this latent manifold drastically reduces computational overhead while preserving, and often enhancing, reconstruction fidelity.

Formally, the generative process is cast as progressively denoising a sample from a tractable prior distribution along predefined trajectories [24]. This dynamics is typically modeled via probability-flow ODEs or flow matching frameworks [13,15,30], with latent trajectories efficiently integrated using DDIM-style

discretizations or specialized ODE solvers [27]. Today, this VAE-coupled diffusion framework serves as the de facto standard for open-domain image synthesis [11, 27], forming the backbone of contemporary SOTA open-source models [2, 3, 33, 38]. Consequently, developing optimization strategies to train these massive Diffusion Transformers rapidly and stably has emerged as a fundamentally critical challenge.

### 2.2 Momentum Orthogonalization for Optimization

The introduction of the Momentum Orthogonalization (Muon) optimizer [9] has catalyzed a paradigm shift toward optimization strategies based on Linear Minimization Oracles (LMOs). A vibrant line of follow-up research has proposed algorithmic extensions, including Scion [21], Gluon [22], and PolarExpress [1], which further refine orthogonalization operators and enhance numerical stability. The empirical superiority of this optimizer family has been systematically validated in comprehensive benchmarking efforts; notably, recent studies [26, 36] demonstrate Muon's advantages across diverse architectures scaling up to one billion parameters. Furthermore, its scalability and efficiency have been rigorously proven in large-scale industrial settings [14, 32]. Given that Muon has rapidly become an indispensable tool for Large Language Model (LLM) pre-training, extending its success to other foundation models presents a highly promising research trajectory.

### 2.3 Muon for DiT Training

Although initially popularized for accelerating LLM pre-training, Muon has recently emerged as a highly compelling alternative to AdamW for training Diffusion Transformers. The heavy reliance of DiTs on dense projection matrices—specifically within self-attention and multi-layer perceptron blocks—makes them structurally primed for Muon's 2D-matrix orthogonalization operations. Recent optimization benchmarks [25] have empirically validated this synergy, demonstrating that Muon achieves significantly lower final losses and accelerated convergence when learning denoising flow trajectories.

The practical scalability of Muon in visual generative modeling is most prominently showcased by recent large-scale foundation models. For instance, in the 8.3-billion-parameter HunyuanVideo 1.5 [37], Muon was deployed across a multi-stage progressive training regimen spanning text-to-image, text-to-video, and image-to-video tasks. Empirical observations indicate that Muon attains a lower training loss than AdamW in only half the optimization steps, drastically accelerating convergence while preserving high-fidelity motion coherence. Similarly, recent pixel MeanFlow [17] highlights that Muon exhibits significantly faster convergence than AdamW when pre-training few-step pixel-space DiTs from scratch. However, despite these early-stage accelerations, applying Muon naively to DiTs often yields suboptimal late-stage convergence—a bottleneck rooted in architectural parameter coupling, which our proposed CMuon directly addresses.

## 3 Methodology

We study how to stably and efficiently train Diffusion Transformers [20] with Muon-style momentum orthogonalization. While Muon can accelerate convergence and reduce optimizer-state overhead, a practical DiT implementation often fuses multiple parameter blocks (e.g., QKV projections or AdaLN projections) into a larger 2D matrix for efficiency. This seemingly innocuous concatenation can be problematic: heterogeneous blocks may exhibit substantially different gradient statistics, yet Muon constructs a shared preconditioner for the fused matrix, implicitly coupling otherwise unrelated subspaces. We formalize this phenomenon as *subspace interference* in Sec. 3.3 and show it can distort the descent geometry. To mitigate it, we introduce **Chunked Muon (CMuon)** in Algorithm 1

### 3.1 Preliminaries on Diffusion Models

Diffusion-based generative models map a tractable prior distribution to a complex empirical data distribution by simulating a dynamic transport process. While classical diffusion formulations learn to reverse a stochastic noise corruption process [7, 28], this work adopts the continuous-time Flow Matching (FM) framework [13]. FM provides a generalized, simulation-free objective for training continuous normalizing flows and has proven highly effective for high-fidelity image generation. Specifically, let $\mathbf{x} \sim p_{\text{data}}$ denote a target data sample and $\mathbf{z} \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$ represent standard Gaussian noise. Flow matching defines a deterministic, probability-flow interpolation path between the data and noise distributions. A standard choice is a simple linear interpolation from data at $t = 0$ to noise at $t = 1$:

$$\mathbf{x}_t \;=\; (1-t)\,\mathbf{x} \;+\; t\,\mathbf{z}, \qquad t \in [0,1]\,. \tag{1}$$

The time derivative of this path yields a constant-velocity target vector field, $\mathbf{z}-\mathbf{x}$. Given an optional conditioning signal $\mathbf{c}$ (e.g., text embeddings), we train a neural network $\boldsymbol{F}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, \mathbf{c})$ to regress this target vector field via a straightforward squared-error objective:

$$\mathcal{L}(\boldsymbol{\theta}) \;=\; \mathbb{E}_{\mathbf{x},\mathbf{z},\mathbf{c},t}\Big[\big\|\boldsymbol{F}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, \mathbf{c}) - (\mathbf{z}-\mathbf{x})\big\|_2^2\Big]\,. \tag{2}$$

During inference, generation is framed as an initial value problem (IVP). Starting from a noise sample $\mathbf{x}_1 \sim \mathcal{N}(\mathbf{0}, \boldsymbol{I})$, we generate a clean data sample $\mathbf{x}_0$ by integrating the learned vector field backward in time (from $t = 1$ to $t = 0$) using the corresponding probability-flow ordinary differential equation (ODE):

$$\frac{\mathrm{d}\mathbf{x}_t}{\mathrm{d}t} \;=\; \boldsymbol{F}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, \mathbf{c})\,. \tag{3}$$

### 3.2 The Muon Optimizer

Under the diffusion training setup above, we optimize model parameters $\boldsymbol{\theta}$ by minimizing the flow-matching objective in Eq. (2):

$$\boldsymbol{\theta}^\star \;=\; \arg\min_{\boldsymbol{\theta}} \; \mathcal{L}(\boldsymbol{\theta}), \qquad \mathcal{L}(\boldsymbol{\theta}) = \mathbb{E}_{\mathbf{x},\mathbf{z},\mathbf{c},t}\Big[\big\|\boldsymbol{F}_{\boldsymbol{\theta}}(\mathbf{x}_t, t, \mathbf{c}) - (\mathbf{z}-\mathbf{x})\big\|_2^2\Big]. \quad (4)$$

Standard optimizers view the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ as a collection of vectors. In contrast, Muon [9] operates matrix-wise: for each 2D weight tensor (e.g., a linear projection) $\boldsymbol{W} \in \mathbb{R}^{m\times n}$, Muon constructs an update direction by orthogonalizing the momentum matrix. This prevents rank collapse of the update by effectively replacing the singular-value spectrum with an identity while removing anisotropic scaling in principle subspaces.

Let $\boldsymbol{W}_t \in \mathbb{R}^{m\times n}$ be a layer weight at iteration $t$, and let $\boldsymbol{G}_t = \nabla_{\boldsymbol{W}}\mathcal{L}(\boldsymbol{\theta}_t) \in \mathbb{R}^{m\times n}$ denote its stochastic gradient computed from minibatches of $(\mathbf{x}, \mathbf{z}, \mathbf{c}, t)$ sampled as in Sec. 3.1. Muon maintains a momentum buffer $\boldsymbol{M}_t \in \mathbb{R}^{m\times n}$ and performs

$$\boldsymbol{M}_t = \mu\, \boldsymbol{M}_{t-1} + \boldsymbol{G}_t, \quad (5)$$

$$\boldsymbol{O}_t = \mathrm{Orth}(\mu \boldsymbol{M}_t + \boldsymbol{G}_t), \quad (6)$$

$$\boldsymbol{W}_t = \boldsymbol{W}_{t-1}(1-\eta\lambda) - \eta \cdot 0.2\sqrt{\max(m,n)}\boldsymbol{O}_t, \quad (7)$$

where $\eta$, $\mu$ and $\lambda$ are the learning rate, momentum coefficient and weight decay rate, respectively, and $0.2\sqrt{\max(m,n)}$ is a dimensional scaling factor that improves scalability [14]. Note that we use Nesterov-type momentum [19]. The operator $\mathrm{Orth}(\cdot)$ maps an input matrix to its closest semi-orthogonal matrix in the sense of the polar factor. Concretely, if the SVD of $\boldsymbol{M}$ is $\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^\top$, then

$$\mathrm{Orth}(\boldsymbol{M}) \;:=\; \boldsymbol{U}\boldsymbol{V}^\top. \quad (8)$$

In practice, Muon avoids explicit SVD by approximating $\boldsymbol{U}\boldsymbol{V}^\top$ via Newton–Schulz iterations for the inverse square root. Several variants further improve numerical accuracy and speed [1].

### 3.3 Subspace Interference in Muon

In DiT, many independent parameters are initialized within the same module for efficiency (e.g., scale projection and shift projection in AdaLN). However, the projection structures and gradient statistics of these parameter blocks can differ substantially. When Muon is applied to a concatenation of such heterogeneous blocks, the orthogonalization step can induce unexpected **coupling** across otherwise unrelated subspaces.

Consider a toy example with $N$ gradient submatrices $\boldsymbol{G}_1, \ldots, \boldsymbol{G}_N \in \mathbb{R}^{d\times d}$ stacked into a tall matrix

$$\boldsymbol{G} \in \mathbb{R}^{Nd\times d}, \qquad \boldsymbol{G} = [\boldsymbol{G}_1, \boldsymbol{G}_2, \cdots, \boldsymbol{G}_N]^\top. \quad (9)$$

Muon approximates an orthogonalized update through Newton–Schulz iterations:

$$\boldsymbol{U} \approx \boldsymbol{G}(\boldsymbol{G}^\top \boldsymbol{G})^{-1/2}. \tag{10}$$

**Vanilla Muon (coupled).** If Muon is applied to the stacked matrix $\boldsymbol{G}$ in Eq. (9), the update follows Eq. (10) with

$$\boldsymbol{U}_i' = \boldsymbol{G}_i \left( \sum_{j=1}^{N} \boldsymbol{G}_j^\top \boldsymbol{G}_j \right)^{-1/2}. \tag{11}$$

**Chunked Muon (decoupled).** If Muon is instead applied independently to each block, the $i$-th update direction becomes

$$\boldsymbol{U}_i = \boldsymbol{G}_i(\boldsymbol{G}_i^\top \boldsymbol{G}_i)^{-1/2}. \tag{12}$$

Here the preconditioner $(\boldsymbol{G}_i^\top \boldsymbol{G}_i)^{-1/2}$ depends only on the statistics of $\boldsymbol{G}_i$. Comparing Eq. (12) and Eq. (11), we observe that stacked Muon replaces the block-specific preconditioner with a shared one $(\boldsymbol{G}^\top \boldsymbol{G})^{-1/2}$. This shared preconditioner mixes gradient covariance structures across blocks that may correspond to unrelated roles. When the dominant principal directions of $\boldsymbol{G}_1$ and $\boldsymbol{G}_2$ are misaligned, the summed covariance distorts each block's preferred descent geometry, producing severe subspace interference. Further theoretical discussions and analyses of CMuon are provided in Appendix B.

### 3.4 Algorithm Design

Our algorithm is intentionally simple yet efficient. For parameter blocks that exhibit cross-subspace coupling under fused Muon, we partition each fused matrix into multiple chunks and perform orthogonalization independently on each chunk. Following prior DiT implementations [30, 40], the fused matrices that most benefit from chunking fall into three categories: (i) AdaLN (Adaptive LayerNorm) modulation, (ii) attention QKV projections, and (iii) FFN gate+up projections.

Table 1 summarizes our chunking scheme for DiT-XL with hidden dimension $d = 1152$. In all cases, we split along the longer dimension so that each chunk corresponds to a semantically independent sub-matrix before applying Newton–Schulz orthogonalization.

**CMuon update.** Algorithm 1 formalizes one optimization step of our CMuon. For each parameter tensor, we (i) apply momentum to the raw gradient, (ii) split into chunks and orthogonalize each chunk via Newton–Schulz iterations, and (iii) apply the scaled update. Here we omit the detailed reshape operation for concise expression.

**Table 1:** Chunk configuration for fused linear layers in DiT-XL ($d = 1152$). Each fused weight matrix is split along the longer dimension into semantically independent sub-matrices before Newton–Schulz orthogonalization.

| Layer Type | Original Shape | Chunk Dim. | #Chunks | Per-Chunk Shape |
|---|---|---|---|---|
| QKV Projection | [3456, 1152] | 0 | 3 | [1152, 1152] |
| MLP Gate+Up | [6144, 1152] | 0 | 2 | [3072, 1152] |
| AdaLN Modulation | [6912, 1152] | 0 | 6 | [1152, 1152] |

**Learning-rate adjustment under chunking.** To balance the update scale between blocks using AdamW and Muon, we adopt the Moonlight variant of Muon scaling, which allows us to directly reuse hyperparameters from AdamW. When chunking is enabled, we recompute the scaling factor based on the per-chunk dimensions so that the overall Frobenius norm of the stacked update remains unchanged. In this way, chunking only redistributes the norm across chunks.

Specifically, let a fused 2D gradient be $\boldsymbol{G} \in \mathbb{R}^{(N d_{\text{out}}) \times d_{\text{in}}}$ with $d_{\text{out}} \geq d_{\text{in}}$, where $N$ denotes the number of chunks and each chunk $\boldsymbol{G}_i \in \mathbb{R}^{d_{\text{out}} \times d_{\text{in}}}$. Under

**Algorithm 1:** Chunked Muon Optimizer Step

**Input:** Tuple set $\mathcal{P}_t = \{(\boldsymbol{W}_t, \boldsymbol{G}_t, d_{\text{chunk}}, N_{\text{chunk}}, \alpha)\}$ over all Muon parameters, where $(d_{\text{chunk}}, N_{\text{chunk}}, \alpha)$ is initialized by Algorithm 2; learning rate $\eta$, momentum $\mu$, weight decay $\lambda$, and Newton–Schulz iterations $K$.

Define NewtonSchulz$(\cdot, \cdot)$ in Algorithm 3.
**for** $t = 1, 2, \ldots$ **do**
  **foreach** $(\boldsymbol{W}_t, \boldsymbol{G}_t, d_{\text{chunk}}, N_{\text{chunk}}, \alpha) \in \mathcal{P}_t$ **do**
    $\boldsymbol{M}_t \leftarrow \mu\, \boldsymbol{M}_{t-1} + \boldsymbol{G}_t$.
    $\boldsymbol{G}_t \leftarrow \boldsymbol{G}_t + \mu\, \boldsymbol{M}_t$.
    **if** $N_{\text{chunk}} > 1$ **then**
      Split $\boldsymbol{G}_t$ along dimension $d_{\text{chunk}}$ into $\{\boldsymbol{G}_t^{(i)}\}_{i=1}^{N_{\text{chunk}}}$.
      **for** $i = 1, \ldots, N_{\text{chunk}}$ **do**
        $\boldsymbol{O}_t^{(i)} \leftarrow \text{NewtonSchulz}(\boldsymbol{G}_t^{(i)}, K)$.
      **end**
      $\boldsymbol{O}_t \leftarrow \text{Concat}\big(\boldsymbol{O}_t^{(1)}, \ldots, \boldsymbol{O}_t^{(N_{\text{chunk}})}\big)$ along dimension $d_{\text{chunk}}$.
    **else**
      $\boldsymbol{O}_t \leftarrow \text{NewtonSchulz}(\boldsymbol{G}_t, K)$.
    **end**
    $\boldsymbol{W}_t \leftarrow (1 - \eta\lambda)\, \boldsymbol{W}_t \; - \; \eta\, \alpha\, \boldsymbol{O}_t$.
  **end**
**end**

Moonlight scaling,

$$\alpha = 0.2\sqrt{\max(Nd_{\text{out}},\, d_{\text{in}})} = 0.2\sqrt{Nd_{\text{out}}}, \tag{13}$$

$$\alpha_c = 0.2\sqrt{\max(d_{\text{out}},\, d_{\text{in}})} = 0.2\sqrt{d_{\text{out}}}. \tag{14}$$

Let $\mathrm{Orth}(\cdot)$ denote the polar factor in (8) (approximated by Newton–Schulz in Algorithm 3). For $m \geq n$, $\mathrm{Orth}(\boldsymbol{G})$ has orthonormal columns, hence $\|\mathrm{Orth}(\boldsymbol{G})\|_F = \sqrt{n} = \sqrt{d_{\text{in}}}$. The Frobenius norm of the fused update becomes

$$\|\alpha\,\mathrm{Orth}(\boldsymbol{G})\|_F = \alpha\sqrt{d_{\text{in}}} = 0.2\sqrt{Nd_{\text{out}}d_{\text{in}}}. \tag{15}$$

where $\mathrm{Orth}(\cdot)$ is defined in (8).

For the chunked counterpart obtained by stacking the chunk updates,

$$\Big\|\alpha_c\,[\mathrm{Orth}(\boldsymbol{G}_1); \mathrm{Orth}(\boldsymbol{G}_2); \cdots; \mathrm{Orth}(\boldsymbol{G}_N)]\Big\|_F = \alpha_c\sqrt{Nd_{\text{in}}} \tag{16}$$

$$= 0.2\sqrt{Nd_{\text{out}}d_{\text{in}}}. \tag{17}$$

Comparing Eq. (15) and Eq. (17), we observe that the Frobenius norm of the full $(Nd_{\text{out}}) \times d_{\text{in}}$ update is preserved. Therefore, under the Moonlight formulation, chunking maintains the global update magnitude while only redistributing the norm across chunks.

Furthermore, we find that scaling the learning-rate multiplier of the chunked components by $\sqrt{N_{\text{chunk}}}$ can accelerate early-stage convergence without compromising the final performance. Therefore, we provide an optional rescaling switch $r$ in Algorithm 2. Detailed ablation results are discussed in Table 5.

## 4 Experiments

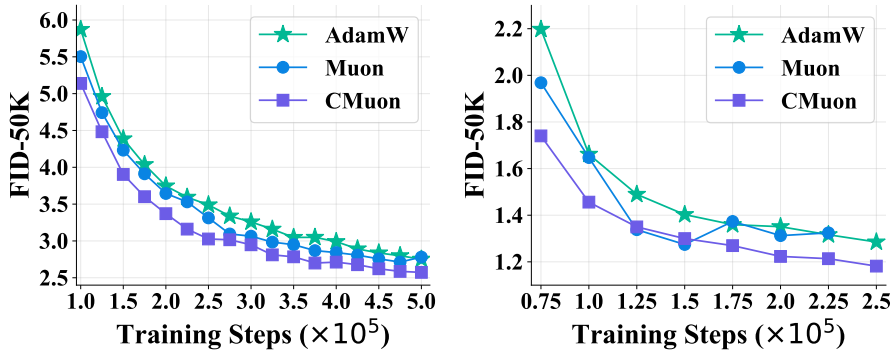
### 4.1 Experimental Setup

**Models and datasets.** We conduct experiments on ImageNet-1K [4] at a resolution of $256 \times 256$, following standard preprocessing protocols used in prior

**Algorithm 2:** Chunked Muon Initialization

**Input:** Muon parameter set $\{\boldsymbol{W}^{(j)}\}$ with associated names; chunking configuration in Table 1; rescaling flag $r \in \{\text{False}, \text{True}\}$.

Initialize momentum buffers $\boldsymbol{M}_0^{(j)} \leftarrow \boldsymbol{0}$ for each parameter $\boldsymbol{W}^{(j)}$.
**foreach** *Muon parameter* $\boldsymbol{W}^{(j)}$ *with 2-dimensional shape* **do**
  Determine $(d_{\text{chunk}}, N_{\text{chunk}})$ according to Table 1 and obtain shape $(d_{\text{out}}, d_{\text{in}})$ after chunking.
  $\alpha^{(j)} \leftarrow 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$.
  **if** $r = \text{True}$ **then**
    $\alpha^{(j)} \leftarrow \alpha^{(j)} \cdot \sqrt{N_{\text{chunk}}}$.
  **end**
  Store $(d_{\text{chunk}},\, N_{\text{chunk}},\, \alpha^{(j)})$ in the optimizer state of $\boldsymbol{W}^{(j)}$.
**end**




**(a) DiT-B.** FID@50k versus training progress for AdamW, Muon, and CMuon. Muon/CMuon accelerate early-stage improvement, while CMuon sustains gains into late training.

**(b) DiT-XL .** FID@50k versus training progress for AdamW, Muon, and CMuon. CMuon maintains continued FID reduction and reaches the same FID up to 2× faster than AdamW.

**Fig. 2: DiT training comparison on ImageNet-1K** $256 \times 256$**.** FID@50k (lower is better) as a function of training progress for AdamW, Muon, and CMuon at two model scales. Muon and CMuon improve markedly faster in early training, while CMuon further preserves late-stage convergence and achieves better final quality.

diffusion-modeling work [5]. All models adopt DiTs [20] as the backbone.[1] We perform latent-space generative modeling using VA-VAE [40] as the default autoencoder, and additionally employ SD-VAE [23] in ablation studies to assess the sensitivity of our findings to the choice of VAE. For sampling, we use model guidance to reduce the inference overhead induced by classifier-free guidance [8, 31]. Unless otherwise specified, all experiments are evaluated with 30 NFEs.

**Implementation details.** Following standard practices, we evaluate EMA models (decay 0.9999) using FID-50K [6]. Training employs a constant learning rate, a global batch size of 1024, `bf16` precision, and gradient clipping (max norm 1.0). AdamW parameters are set to $(\beta_1, \beta_2) = (0.9, 0.95)$ with 0 weight decay. Crucially, Muon and CMuon optimize only 2D weights in attention/FFN/AdaLN projections, while AdamW handles 1D parameters, embeddings, and the final layer [9, 37]. See Appendix A for further details.

### 4.2 Performance of CMuon in DiT Training

We evaluate CMuon against AdamW and vanilla Muon on DiT-Base and DiT-XL . As Figure 2 shows, while both Muon variants accelerate early-stage training, vanilla Muon plateaus in later stages. In contrast, CMuon sustains its convergence rate throughout training. Ultimately, CMuon achieves an FID of 1.18

[1] Following [30], we apply minor architectural modifications for neural network for improved stability and performance.

**Table 2: Comparison of optimizers on image generation task on class-conditional ImageNet-1K.** Across settings, vanilla Muon reduces FID faster in the early stage, but its improvement diminishes later and becomes comparable to AdamW by 200 epochs. In contrast, CMuon maintains a faster late-stage convergence than AdamW while achieving higher final accuracy, consistently converging to a lower FID.

| Optimizer | Model | #Params | VAE | NFE (↓) | FID (↓) | | |
|---|---|---|---|---|---|---|---|
| | | | | | ep80 | ep200 | ep400 |
| AdamW | DiT-B | 130M | VA-VAE | 30 | 5.87 | 3.49 | 2.78 |
| Muon | DiT-B | 130M | VA-VAE | 30 | 5.50 | 3.31 | 2.78 |
| CMuon | DiT-B | 130M | VA-VAE | 30 | 5.14 | 3.03 | 2.57 |
| AdamW | DiT-B | 130M | SD-VAE | 30 | 5.87 | 3.60 | – |
| Muon | DiT-B | 130M | SD-VAE | 30 | 5.39 | 3.30 | – |
| CMuon | DiT-B | 130M | SD-VAE | 30 | 4.94 | 3.06 | – |
| AdamW | DiT-XL | 675M | VA-VAE | 30 | 1.66 | 1.30 | 1.21 |
| Muon | DiT-XL | 675M | VA-VAE | 30 | 1.65 | 1.29 | – |
| CMuon | DiT-XL | 675M | VA-VAE | 30 | **1.46** | **1.18** | – |

**(a) Muon** after training 200 epochs, FID@50k = 1.29.

**(b) CMuon** after training 200 epochs, FID@50k = 1.18.

**Fig. 3: Qualitative comparison on ImageNet-1K** 256 × 256**.** Random class-conditional samples generated by DiT-XL trained with Muon (left) and CMuon (right). All images are sampled with the same sampling configuration and NFE = 30.

in just 200 epochs, outperforming AdamW's 400-epoch FID of 1.21 (Table 2). Overall, CMuon delivers over a 2x training speedup compared to AdamW.

Figure 3 provides a qualitative comparison. Given the same DiT-XL setup, 200-epoch budget, and 30 NFE, CMuon yields visually cleaner and more coherent images than vanilla Muon. Specifically, CMuon better captures fine details and mitigates common failures like texture artifacts and structural distortions. Aligning with our FID results, this confirms that chunked optimization improves not only early-stage speed but also final perceptual quality.

**Table 3: Ablation Study on Chunked Blocks: Impact of QKV, FFN, and AdaLN Projections on 130M DiT-B Models.** The table presents the performance of the 130M DiT-B model on the FID metric at 80 and 200 epochs under different configurations of chunked blocks. The results show that the inclusion of FFN, QKV, and AdaLN projections leads to a notable improvement in FID at both epochs compared to other individual components, highlighting the importance of combining these elements for enhanced model performance.

| Chunked Blocks | FID@80ep | FID@200ep |
|---|---|---|
| None | 5.50 | 3.31 |
| FFN | 5.43 | 3.28 |
| QKV | 5.32 | 3.35 |
| AdaLN | 6.00 | 3.23 |
| FFN, QKV, AdaLN | **5.14** | **3.02** |

## 5 Further Ablation Studies

### 5.1 Ablation on Blocks for Chunking

To more precisely validate the importance of applying chunking to different architectural blocks, we conduct an ablation study on the 130M DiT-B model by selectively enabling chunking for different projection layers. In this experiment, +FFN denotes applying chunking to the gate and up projections in the feed-forward network (FFN), +QKV denotes chunking the query, key, and value projections in the multi-head attention module, and +AdaLN refers to chunking the projection layer inside Adaptive LayerNorm. Under this setting, None corresponds to the original Muon optimizer without chunking, while the other configurations represent variants of our CMuon with different block-level chunking strategies.

We report the FID after 80 and 200 training epochs to evaluate both early-stage convergence and final generative performance. As shown in Table 3, applying chunking to any single block individually provides only marginal improvements in the later stage of training, reducing the final FID by at most 0.1 compared with the baseline. Nevertheless, a modest acceleration in convergence can be observed for several configurations. In contrast, when chunking is jointly applied to FFN, QKV, and AdaLN projections, the improvement becomes substantially more pronounced. Specifically, the final FID decreases from 3.31 to 3.02 at 200 epochs, corresponding to an improvement of approximately 0.3, while the early-stage performance at 80 epochs also shows a clear gain. These results suggest that chunking different architectural blocks produces complementary optimization benefits, and jointly applying chunking to FFN, QKV, and AdaLN yields the most effective and balanced configuration for stabilizing and accelerating Muon-based training.

**Table 4: Ablation study on Muon learning rate scaling strategies for DiT-B, with all models trained for 200 epochs.**This table compares the performance of different learning rate scaling strategies, including Vanilla, KellerJordan, and MoonLight, on both Muon and CMuon models. The scaling strategies are evaluated based on their respective FID scores, with MoonLight showing the best performance in both cases.

| **Scale Function** | **Vanilla** | **KellerJordan** | **MoonLight** |
|---|---|---|---|
| **Scaling factor** | $\times 1$ | $\times \max\left(\sqrt{\frac{d_{\text{out}}}{d_{\text{in}}}}, 1\right)$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ |
| Muon | 8.73 | 7.40 | **3.31** |
| CMuon | 6.94 | 6.00 | **3.03** |

### 5.2 Effect of Learning Rate Scaling Strategies

When transitioning from AdamW-style optimization to Muon, several block-wise learning rate scaling strategies have been proposed to stabilize training across different parameter shapes. To evaluate the robustness of CMuon under different scaling rules, we conduct an ablation study using three representative strategies: Vanilla (no learning rate scaling), the Keller–Jordan strategy, and the MoonLight strategy. The detailed scaling configurations are summarized in Table 4. In particular, $d_{\text{out}}$ and $d_{\text{in}}$ denote the output and input dimensions of a parameter matrix after reshaping it into a two-dimensional form. Following our standard experimental protocol, all models are trained on the 130M DiT-B architecture for 200 epochs, and the final FID scores are reported in Table 4.

The results demonstrate that CMuon consistently achieves lower FID scores than the original Muon optimizer across all three scaling strategies, indicating that the proposed chunking mechanism improves optimization stability regardless of the specific learning rate scaling rule. It is also worth noting that the Vanilla and Keller–Jordan configurations produce noticeably worse FID compared with the MoonLight strategy. This behavior mainly arises because we directly adopt the learning rate originally tuned for AdamW without further retuning, which leads to suboptimal scaling for the Vanilla and Keller–Jordan rules. In contrast, the MoonLight scaling naturally aligns better with this learning rate choice and therefore yields significantly better performance. For this reason, we adopt the MoonLight scaling strategy as the default configuration in the remainder of this paper.

### 5.3 Ablation Study on Learning Rate Rescaling

As mentioned at the end of Section 9, we observe that rescaling the learning rate of the chunked blocks by a factor of $\sqrt{N_{\text{chunk}}}$ can substantially improve early-stage convergence without slowing down optimization in the later stages. To isolate the effect of this rescaling trick, we apply it to both Muon and CMuon and compare their FID scores after 40 and 80 training epochs. For the Muon + Rescale setting, we rescale only the same set of layers that are chunked in

CMuon, while keeping these layers unchunked and optimized with standard Muon. Such operation ensures that the comparison targets the learning-rate effect rather than the chunking operation itself. All configurations follow the same Scaling Factor MoonLight and the results are summarized in Table 5.

As shown in Table 5, learning-rate rescaling and chunking exhibit clear complementarity: each brings consistent gains, and their combination yields the best overall performance. In particular, the two Rescale configurations achieve the lowest FID at 40 epochs, indicating that $\sqrt{N_{chunk}}$ rescaling primarily accelerates early diffusion training. By 80 epochs, the configurations using CMuon deliver the best FID, suggesting that chunking plays the dominant role in improving the attainable final performance. Taken together, these results imply a division of labor between the two techniques: rescaling mainly improves early-stage optimization speed, whereas chunking largely determines the ultimate FID improvements and remains the primary contributor to late-stage gains.

### 5.4 Ablation Study on Learning Rate

To ensure that our comparisons are conducted under near-optimal hyperparameters, we ablate the learning rate when training DiT-XL with 675M parameters. Motivated by the commonly adopted setting in UCGM [30], we evaluate learning rates in the neighborhood of $2 \times 10^{-4}$, specifically $\{1, 2, 3\} \times 10^{-4}$, and report the resulting FID@50k at multiple training horizons (80/140/200 epochs) for both AdamW and CMuon. Table 6 summarizes the sensitivity of final image quality to learning rate and training length, enabling a fair assessment of optimizer behavior without confounding from poorly tuned learning rates.

Table 6 shows that $2 \times 10^{-4}$ is consistently near-optimal across both optimizers, achieving strong performance at all evaluated epochs. While AdamW can reach a competitive FID of 1.26 with a larger learning rate ($3 \times 10^{-4}$) after 200 epochs, CMuon attains comparable or better quality substantially earlier: for instance, CMuon achieves FID 1.27 at 140 epochs with $2 \times 10^{-4}$, closely matching AdamW's 200-epoch result, thereby maintaining the convergence speedup observed in our main experiments. Overall, this ablation confirms that our reported gains are not an artifact of an unfavorable learning-rate choice

**Table 5: Ablation study on learning-rate rescaling for chunked layers.** This table explores the effect of varying learning-rate rescaling on chunked layers in the Muon and CMuon optimizers. We observe the influence of different rescaling strategies on the FID scores at both 40 and 80 epochs, showing significant improvements when rescaling is applied based on chunk size.

| Optimizer | Base LR Scale | Chunk Rescale | FID@40ep | FID@80ep |
|---|---|---|---|---|
| Muon | $0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ | 5.67 | 1.65 |
| Muon | $0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times\sqrt{N_{\text{chunk}}}$ | 4.26 | 1.55 |
| CMuon | $0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ | 5.32 | 1.50 |
| CMuon | $0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times\sqrt{N_{\text{chunk}}}$ | **3.78** | **1.46** |

**Table 6: Analysis on best learning rate and training epochs when training DiT.** We report FID@50k for AdamW and CMuon across learning rates $\{1, 2, 3\} \times 10^{-4}$ and training lengths of 80/140/200 epochs.

| LR | AdamW | | | CMuon | | |
|---|---|---|---|---|---|---|
| | FID@80ep | FID@140ep | FID@200ep | FID@80ep | FID@140ep | FID@200ep |
| 1e−4 | 2.13 | – | – | 1.79 | – | – |
| 2e−4 | 1.66 | 1.36 | 1.29 | 1.46 | 1.27 | 1.18 |
| 3e−4 | 1.63 | 1.32 | 1.26 | 1.54 | 1.25 | 1.20 |

for AdamW, and that CMuon remains robust and effective under well-searched learning-rate settings.

## 6 Conclusion

We introduce CMuon to resolve the late-stage convergence plateaus of the Muon optimizer in Diffusion Transformers. Because standard DiTs fuse distinct projections (e.g., QKV, AdaLN), joint orthogonalization causes cross-subspace coupling. CMuon simply chunks these matrices back into independent sub-components before orthogonalization. On ImageNet-1K $256 \times 256$, this zero-overhead modification achieves a SOTA FID of 1.18 on DiT-XL at 200 epochs—surpassing AdamW, which requires 400 epochs to reach 1.21. CMuon establishes a highly efficient and stable optimization standard for large-scale DiT training.

## References


1. Amsel, N., Persson, D., Musco, C., Gower, R.M.: The polar express: Optimal matrix sign methods and their application to the muon algorithm. arXiv preprint arXiv:2505.16932 (2025)
2. Batifol, S., Blattmann, A., Boesel, F., Consul, S., Diagne, C., Dockhorn, T., English, J., English, Z., Esser, P., Kulal, S., et al.: Flux. 1 kontext: Flow matching for in-context image generation and editing in latent space. arXiv e-prints pp. arXiv–2506 (2025)
3. Cai, H., Cao, S., Du, R., Gao, P., Hoi, S., Hou, Z., Huang, S., Jiang, D., Jin, X., Li, L., et al.: Z-image: An efficient image generation foundation model with single-stream diffusion transformer. arXiv preprint arXiv:2511.22699 (2025)
4. Deng, J., Dong, W., Socher, R., Li, L.J., Li, K., Fei-Fei, L.: Imagenet: A large-scale hierarchical image database. In: 2009 IEEE conference on computer vision and pattern recognition. pp. 248–255. Ieee (2009)
5. Dhariwal, P., Nichol, A.: Diffusion models beat gans on image synthesis. Advances in neural information processing systems **34**, 8780–8794 (2021)
6. Heusel, M., Ramsauer, H., Unterthiner, T., Nessler, B., Hochreiter, S.: Gans trained by a two time-scale update rule converge to a local nash equilibrium. Advances in neural information processing systems **30** (2017)
7. Ho, J., Jain, A., Abbeel, P.: Denoising diffusion probabilistic models. Advances in neural information processing systems **33**, 6840–6851 (2020)

8. Ho, J., Salimans, T.: Classifier-free diffusion guidance. arXiv preprint arXiv:2207.12598 (2022)
9. Jordan, K., Jin, Y., Boza, V., Jiacheng, Y., Cesista, F., Newhouse, L., Bernstein, J.: Muon: An optimizer for hidden layers in neural networks, 2024. URL https://kellerjordan. github. io/posts/muon **6**(3), 4 (2024)
10. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
11. Kingma, D.P., Welling, M.: Auto-encoding variational bayes. arXiv preprint arXiv:1312.6114 (2013)
12. Leng, X., Singh, J., Hou, Y., Xing, Z., Xie, S., Zheng, L.: Repa-e: Unlocking vae for end-to-end tuning of latent diffusion transformers. In: Proceedings of the IEEE/CVF International Conference on Computer Vision. pp. 18262–18272 (2025)
13. Lipman, Y., Chen, R.T., Ben-Hamu, H., Nickel, M., Le, M.: Flow matching for generative modeling. arXiv preprint arXiv:2210.02747 (2022)
14. Liu, J., Su, J., Yao, X., Jiang, Z., Lai, G., Du, Y., Qin, Y., Xu, W., Lu, E., Yan, J., et al.: Muon is scalable for llm training. arXiv preprint arXiv:2502.16982 (2025)
15. Liu, X., Gong, C., Liu, Q.: Flow straight and fast: Learning to generate and transfer data with rectified flow. arXiv preprint arXiv:2209.03003 (2022)
16. Loshchilov, I., Hutter, F.: Decoupled weight decay regularization. arXiv preprint arXiv:1711.05101 (2017)
17. Lu, Y., Lu, S., Sun, Q., Zhao, H., Jiang, Z., Wang, X., Li, T., Geng, Z., He, K.: One-step latent-free image generation with pixel mean flows. arXiv preprint arXiv:2601.22158 (2026)
18. Ma, N., Goldstein, M., Albergo, M.S., Boffi, N.M., Vanden-Eijnden, E., Xie, S.: Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In: European Conference on Computer Vision. pp. 23–40. Springer (2024)
19. Nesterov, Y.: A method for solving the convex programming problem with convergence rate o (1/k2). In: Dokl akad nauk Sssr. vol. 269, p. 543 (1983)
20. Peebles, W., Xie, S.: Scalable diffusion models with transformers. In: Proceedings of the IEEE/CVF international conference on computer vision. pp. 4195–4205 (2023)
21. Pethick, T., Xie, W., Antonakopoulos, K., Zhu, Z., Silveti-Falls, A., Cevher, V.: Training deep learning models with norm-constrained lmos. arXiv preprint arXiv:2502.07529 (2025)
22. Riabinin, A., Shulgin, E., Gruntkowska, K., Richtárik, P.: Gluon: Making muon & scion great again!(bridging theory and practice of lmo-based optimizers for llms). arXiv preprint arXiv:2505.13416 (2025)
23. Rombach, R., Blattmann, A., Lorenz, D., Esser, P., Ommer, B.: High-resolution image synthesis with latent diffusion models. In: Proceedings of the IEEE/CVF conference on computer vision and pattern recognition. pp. 10684–10695 (2022)
24. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015)
25. Schaipp, F.: Optimization benchmark for diffusion models on dynamical systems. arXiv preprint arXiv:2510.19376 (2025)
26. Semenov, A., Pagliardini, M., Jaggi, M.: Benchmarking optimizers for large language model pretraining. arXiv preprint arXiv:2509.01440 (2025)
27. Song, J., Meng, C., Ermon, S.: Denoising diffusion implicit models. arXiv preprint arXiv:2010.02502 (2020)

28. Song, Y., Sohl-Dickstein, J., Kingma, D.P., Kumar, A., Ermon, S., Poole, B.: Score-based generative modeling through stochastic differential equations. arXiv preprint arXiv:2011.13456 (2020)
29. Su, J.: Why is the adam update rms 0.2? (Sep 2025), https://www.spaces.ac.cn/archives/11267
30. Sun, P., Jiang, Y., Lin, T.: Unified continuous generative models. arXiv preprint arXiv:2505.07447 (2025)
31. Tang, Z., Bao, J., Chen, D., Guo, B.: Diffusion models without classifier-free guidance. arXiv preprint arXiv:2502.12154 (2025)
32. Team, K., Bai, Y., Bao, Y., Chen, G., Chen, J., Chen, N., Chen, R., Chen, Y., Chen, Y., Chen, Y., et al.: Kimi k2: Open agentic intelligence. arXiv preprint arXiv:2507.20534 (2025)
33. Team, M.L., Ma, H., Tan, H., Huang, J., Wu, J., He, J.Y., Gao, L., Xiao, S., Wei, X., Ma, X., et al.: Longcat-image technical report. arXiv preprint arXiv:2512.07584 (2025)
34. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
35. Wang, S., Tian, Z., Huang, W., Wang, L.: Ddt: Decoupled diffusion transformer. arXiv preprint arXiv:2504.05741 (2025)
36. Wen, K., Hall, D., Ma, T., Liang, P.: Fantastic pretraining optimizers and where to find them. arXiv preprint arXiv:2509.02046 (2025)
37. Wu, B., Zou, C., Li, C., Huang, D., Yang, F., Tan, H., Peng, J., Wu, J., Xiong, J., Jiang, J., et al.: Hunyuanvideo 1.5 technical report. arXiv preprint arXiv:2511.18870 (2025)
38. Wu, C., Li, J., Zhou, J., Lin, J., Gao, K., Yan, K., Yin, S.m., Bai, S., Xu, X., Chen, Y., et al.: Qwen-image technical report. arXiv preprint arXiv:2508.02324 (2025)
39. Xie, E., Chen, J., Chen, J., Cai, H., Tang, H., Lin, Y., Zhang, Z., Li, M., Zhu, L., Lu, Y., et al.: Sana: Efficient high-resolution image synthesis with linear diffusion transformers. arXiv preprint arXiv:2410.10629 (2024)
40. Yao, J., Yang, B., Wang, X.: Reconstruction vs. generation: Taming optimization dilemma in latent diffusion models. In: Proceedings of the Computer Vision and Pattern Recognition Conference. pp. 15703–15712 (2025)
41. Yu, S., Kwak, S., Jang, H., Jeong, J., Huang, J., Shin, J., Xie, S.: Representation alignment for generation: Training diffusion transformers is easier than you think. arXiv preprint arXiv:2410.06940 (2024)
42. Yu, Y., Xiong, W., Nie, W., Sheng, Y., Liu, S., Luo, J.: Pixeldit: Pixel diffusion transformers for image generation. arXiv preprint arXiv:2511.20645 (2025)
43. Zhou, X., Li, Q., Hu, X., Chen, H., Gu, S.: Guiding a diffusion transformer with the internal dynamics of itself. arXiv preprint arXiv:2512.24176 (2025)

# A Experimental Settings

## A.1 Detailed Settings of Training and Sampling

In al our experiments, we use two types of DiT backbones following the implementation of [30], namely DiT-B and DiT-XL. Their architectural configurations are summarized in Table 7. Specifically, DiT-B contains 12 layers with a hidden size of 768, 12 attention heads, and a head dimension of 64, while DiT-XL contains 28 layers with a hidden size of 1152, 16 attention heads, and a head dimension of 72. Both models use an MLP ratio of 4.0 and a patch size of 2. Overall, DiT-XL scales DiT-B primarily through increased depth and width, thereby providing substantially larger model capacity.

For image generation, we use the same training objective and sampling hyperparameters as UCGM [30]. For each resolution and latent autoencoder, we directly adopt the same settings as UCGM. The sampling uses 30 steps for VA-VAE and 40 steps for SD-VAE and DC-AE. The remaining implementation details can be found in the [30].

## A.2 Hyperparameter Settings for Optimization

In this section, we summarize the default hyperparameter settings used in the ImageNet-1K experiments [4]. Unless otherwise specified, all models are trained with a global batch size of 1024, an exponential moving average (EMA) decay of 0.9999, `bf16` mixed precision, and gradient clipping with a maximum norm of 1.0. For both Muon and CMuon, Muon-style updates are applied exclusively to the 2D weight matrices in the attention, FFN, and AdaLN projections. In contrast, 1D parameters, embeddings, and the final layer are optimized using AdamW with $\beta_1 = 0.9$, $\beta_2 = 0.95$, and a weight decay of 0. The detailed settings are illustrated in Table 8.

## A.3 Further Ablation Studies on Chunked Blocks

We conduct additional ablation studies on chunked blocks to evaluate the impact of various components such as the QKV, FFN, and AdaLN projections in a 130M DiT-B model based on Table 3. These experiments aim to investigate how different configurations of chunked blocks affect the performance of the model during training, specifically focusing on the FID at 80 and 200 epochs. The

**Table 7:** Model configurations for DiT-B and DiT-XL.

| Model | Params | Depth | Hidden Size | Head Number | Hidden Dim | MLP Ratio | Patch Size |
|---|---|---|---|---|---|---|---|
| DiT-B | 130M | 12 | 768 | 12 | 64 | 4.0 | 2 |
| DiT-XL | 675M | 28 | 1152 | 16 | 72 | 4.0 | 2 |

results provide insight into the significance of each component in the model's performance.

The experimental results are summarized in Table 9, which demonstrates the effect of different block configurations on the FID scores. The findings suggest that incorporating QKV and FFN projections consistently improves the model's performance, as reflected by the lower FID values at both 80 and 200 epochs.

### A.4 Image Synthesis on ImageNet 512 × 512

We evaluate the performance of the 675M DiT-XL model on the ImageNet 512×512 dataset using three different optimizers: AdamW, Muon, and CMuon. The objective is to assess the effectiveness of each optimizer in generating high-resolution images. We use the 675M DiT-XL model with a DC-AE encoder and train it for a total of 400 epochs. During the training process, we closely monitor the FID scores at multiple epochs: 80, 200, and 400.

Table 10 presents the detailed FID scores for each optimizer at different training epochs. As shown in the table, AdamW achieves an FID of 2.67 at 80 epochs, which improves to 1.67 at 400 epochs. Muon, on the other hand, performs better than AdamW throughout the training process, with an FID of 2.16 at 80 epochs, 1.81 at 200 epochs, and 1.65 at 400 epochs. Most importantly, CMuon outperforms both AdamW and Muon, reaching an FID of 1.60 at 400 epochs, the best score among all the optimizers tested. This demonstrates that the CMuon optimizer not only performs well in the early convergence phase but also continues to improve, yielding the best results in terms of FID at higher epochs, thus validating its effectiveness for high-resolution image generation.

**Table 8: Default hyperparameter configurations for Muon and CMuon on ImageNet-1K.** The table lists the default hyperparameter settings used in the DiT-B and DiT-XL models during the ImageNet-1K experiments.

| VAE/AE | Model | Optimizer | NS Steps | LR | Base LR Scale | Chunk Rescale |
|---|---|---|---|---|---|---|
| VA-VAE | DiT-B | AdamW | - | $2 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ |
| VA-VAE | DiT-B | Muon | 5 | $2 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ |
| VA-VAE | DiT-B | CMuon | 5 | $2 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times\sqrt{N_{\text{chunk}}}$ |
| SD-VAE | DiT-B | AdamW | - | $3 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ |
| SD-VAE | DiT-B | Muon | 5 | $3 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ |
| SD-VAE | DiT-B | CMuon | 5 | $3 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times\sqrt{N_{\text{chunk}}}$ |
| DC-AE | DiT-XL | AdamW | - | $2 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ |
| DC-AE | DiT-XL | Muon | 6 | $2 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ |
| DC-AE | DiT-XL | CMuon | 6 | $2 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times\sqrt{N_{\text{chunk}}}$ |
| SD-VAE | DiT-XL | AdamW | - | $2 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ |
| SD-VAE | DiT-XL | Muon | 6 | $2 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times 1$ |
| SD-VAE | DiT-XL | CMuon | 6 | $2 \times 10^{-4}$ | $\times 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}$ | $\times\sqrt{N_{\text{chunk}}}$ |

**Table 9: Full ablation study of chunked blocks: QKV, FFN, and AdaLN projections in 130M DiT-B models.** This study evaluates the impact of different combinations of QKV, FFN, and AdaLN projections on the model's performance, focusing on the FID at 80 and 200 epochs. The results offer insights into how these components contribute to the model's training and final accuracy.

| AdaLN | QKV | FFN | FID@80ep | FID@200ep |
|---|---|---|---|---|
| | | | 5.50 | 3.31 |
| | | ✓ | 5.43 | 3.28 |
| | ✓ | | 5.32 | 3.35 |
| ✓ | | | 6.00 | 3.23 |
| | ✓ | ✓ | **5.11** | 3.17 |
| ✓ | | ✓ | 5.69 | 3.26 |
| ✓ | ✓ | | 5.28 | 3.15 |
| ✓ | ✓ | ✓ | 5.14 | **3.02** |

### A.5 Further Exploration on Pixel-Space DiT

In this section, we extend our investigation to Pixel-Space DiT, which allows us to evaluate the performance of the model in a different space compared to the original approach. Specifically, we compare the performance of Pixel-Space DiT using different optimizers, including AdamW, Muon, and a chunked architecture, referred to as CMuon. As shown in Table 11, PixelDiT's performance decreases when switching to Muon, but the performance does not degrade as drastically when using CMuon. This indicates that our architectural improvements to the chunked design are still valuable in the pixel-space setting, as CMuon retains a more stable performance compared to Muon.

The relatively better performance of AdamW can be attributed to specific architectural modifications we made during our architecture search for PixelDiT [42]. Since PixelDiT is not open-source, our adjustments–particularly to the bottleneck and final layers–could cause the model to slightly overfit to the Adam optimizer, enhancing its performance on this specific configuration.

### A.6 Training Efficiency with System-Level Optimization

We evaluate the training efficiency of the Muon optimizer and the proposed CMuon on a cluster of 8× H100 GPUs. Because Muon-style optimizers require

**Table 10: Results of image generation with 512×512 resolution on class-conditional ImageNet-1K.** The model trained here is a 675M DiT-XL with a DC-AE encoder.

| Optimizer | FID@80ep | FID@200ep | FID@400ep |
|---|---|---|---|
| AdamW | 2.67 | 1.94 | 1.67 |
| Muon | **2.16** | **1.81** | 1.65 |
| CMuon | 2.38 | 1.86 | **1.60** |

**Table 11: Results of image generation with 512×512 resolution on class-conditional ImageNet-1K.** We are training a 797M PixelDiT-XL with a batch size of 256 and a learning rate of $10^{-4}$. REPA is used for training acceleration.

| Optimizer | FID@40ep | FID@80ep | FID@160ep |
|---|---|---|---|
| AdamW | 3.78 | 2.95 | 2.50 |
| Muon | 4.45 | 3.26 | 2.70 |
| CMuon | 4.27 | 3.10 | 2.56 |

Newton–Schulz iterations on layer-wise matrices, a naive implementation can introduce non-trivial computational overhead relative to first-order optimizers such as Adam. To mitigate this issue, we implement several system-level optimizations that substantially improve the training throughput of Muon-based methods.

**Optimization 1: Parallelized Newton–Schulz with overlapped communication.** We shard parameter matrices across GPUs so that different devices perform NS iterations on different parameter blocks in parallel. The computation of NS updates is overlapped with inter-GPU communication, which effectively hides part of the synchronization overhead and improves GPU utilization.

**Optimization 2: Kernel-level acceleration.** For unchunked parameters, we implement a Triton-based fused kernel to accelerate Newton–Schulz computation. For chunked parameters, we further employ a batched Newton–Schulz implementation so that multiple chunks can be processed simultaneously on the GPU, thereby improving arithmetic intensity and reducing kernel launch overhead.

**Maximum training throughput.** Table 12 reports the maximum training throughput, measured in iterations per second, under different optimizers. We consider two representative configurations: DiT-B and DiT-XL with VA-VAE on ImageNet. Without system-level optimization, Muon and CMuon exhibit noticeable overhead relative to Adam. After the proposed optimizations are applied

**Table 12: Maximum training throughput (iterations/s) on 8× H100 GPUs.**

| Optimizer | System Opt. | DiT-B | DiT-XL |
|---|---|---|---|
| Adam | | 9.51 | 2.32 |
| Muon | | 7.60 | 1.95 |
| Muon | ✓ | 8.98 | 2.27 |
| CMuon | | 6.24 | 1.72 |
| CMuon | ✓ | **9.07** | **2.30** |

**Algorithm 3:** Newton–Schulz Iteration for Approximate Orthogonalization

**Input :** Gradient matrix $\boldsymbol{G} \in \mathbb{R}^{m\times n}$; number of iterations $K$.
**Output:** Approximately orthogonalized matrix $\boldsymbol{X}_K \in \mathbb{R}^{m\times n}$.

Set the quintic coefficients $a = 3.4445$, $b = -4.7750$, and $c = 2.0315$.
$\boldsymbol{X}_0 \leftarrow \boldsymbol{G}/\|\boldsymbol{G}\|_F$.
transposed $\leftarrow$ False.
**if** $m > n$ **then**
  $\boldsymbol{X}_0 \leftarrow \boldsymbol{X}_0^\top$.
  transposed $\leftarrow$ True.
**end**
**for** $k = 0, \ldots, K-1$ **do**
  $\boldsymbol{A} \leftarrow \boldsymbol{X}_k \boldsymbol{X}_k^\top$.
  $\boldsymbol{B} \leftarrow b\,\boldsymbol{A} + c\,\boldsymbol{A}^2$.
  $\boldsymbol{X}_{k+1} \leftarrow a\,\boldsymbol{X}_k + \boldsymbol{B}\,\boldsymbol{X}_k$.
**end**
**if** transposed **then**
  $\boldsymbol{X}_K \leftarrow \boldsymbol{X}_K^\top$.
**end**
**return** $\boldsymbol{X}_K$.

(denoted as **System Opt.**), the throughput of Muon-based methods becomes comparable to that of Adam while preserving their optimization advantages. In particular, CMuon with Flash achieves nearly identical throughput to Adam for both DiT-B and DiT-XL.

## B Additional Analyses and Details

This section provides additional implementation details and analyses of the proposed method. In particular, it describes the practical implementation of the Newton–Schulz iteration used for Muon-style matrix orthogonalization.

### B.1 Implementation of Newton–Schulz Iteration

Muon-style updates require an efficient approximation of matrix orthogonalization. In practice, we adopt the Newton–Schulz iteration to transform the input gradient matrix into an approximately orthogonalized matrix with low computational overhead. Given a gradient matrix $\boldsymbol{G} \in \mathbb{R}^{m\times n}$, we first normalize it by its Frobenius norm to improve numerical stability. When the matrix is tall ($m > n$), we transpose it before applying the iteration so that the computation is performed on the more favorable orientation. The output is transposed back afterward when necessary.

### B.2 Moonlight Muon Scaling for RMS Norm Preservation

In the Moonlight variant of Muon scaling, we aim to maintain the global Frobenius norm of the update while redistributing it across chunks. The learning rate scaling factor $\alpha$ for both the full update and the chunked components is defined as:

$$\boldsymbol{U}_{\text{muon}} = 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}\text{Orth}(\boldsymbol{M}), \tag{18}$$

$$\boldsymbol{U}_{\text{adam}} = \boldsymbol{M}/\sqrt{\boldsymbol{V} + \epsilon}, \tag{19}$$

where $\boldsymbol{M}$ is the momentum matrix, and $\boldsymbol{V}$ is the variance matrix. Both $\boldsymbol{M}$ and $\boldsymbol{V}$ are two-dimensional matrices, with dimensions corresponding to the number of model parameters and their gradients. The update term $\boldsymbol{U}$ represents the update step applied to the parameters of the model during optimization.

[29] observed that the RMS norm of updates in AdamW typically stabilizes around 0.2-0.3 during training. This stability holds across models of different sizes and configurations, indicating a fundamental property of the AdamW optimizer with $\beta_1 = 0.9$ and $\beta_2 = 0.95$. By approximating the RMS value of AdamW's update, we obtain:

$$\text{RMS}(\boldsymbol{U}_{\text{adam}}) \approx 0.2. \tag{20}$$

For the Moonlight Muon scaling, we compute the RMS norm as follows:

$$\text{RMS}(\boldsymbol{U}_{\text{muon}}) = 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}\text{RMS}(\text{Orth}(\boldsymbol{M})) \tag{21}$$

$$= 0.2\sqrt{\max(d_{\text{out}}, d_{\text{in}})}\sqrt{\frac{\|\text{Orth}(\boldsymbol{M})\|_F^2}{d_{\text{out}} d_{\text{in}}}} \tag{22}$$

$$= 0.2\sqrt{\frac{\max(d_{\text{out}}, d_{\text{in}})\min(d_{\text{out}}, d_{\text{in}})}{d_{\text{out}} d_{\text{in}}}} \tag{23}$$

$$= 0.2 \tag{24}$$

Thus, both the AdamW and Moonlight Muon updates yield an RMS norm of approximately 0.2, ensuring that the norm preservation property holds for the chunked updates in Moonlight LR scaling.