Scaling Diffusion Transformers Efficiently via μP

Chenyu Zheng[1,2,3,*] **Xinyu Zhang**[4], **Rongzhen Wang**[1,2,3], **Wei Huang**[5], **Zhi Tian**[4],
Weilin Huang[4], **Jun Zhu**[6], **Chongxuan Li**[1,2,3,†]

[1] Gaoling School of Artificial Intelligence, Renmin University of China
[2] Beijing Key Laboratory of Research on Large Models and Intelligent Governance
[3] Engineering Research Center of Next-Generation Intelligent Search and Recommendation, MOE
[4] ByteDance Seed [5] RIKEN AIP [6] Dept. of Comp. Sci. & Tech., Tsinghua University

Abstract

Diffusion Transformers have emerged as the foundation for vision generative models, but their scalability is limited by the high cost of hyperparameter (HP) tuning at large scales. Recently, Maximal Update Parametrization (μP) was proposed for vanilla Transformers, which enables stable HP transfer from small to large language models, and dramatically reduces tuning costs. However, it remains unclear whether μP of vanilla Transformers extends to diffusion Transformers, which differ architecturally and objectively. In this work, we generalize standard μP to diffusion Transformers and validate its effectiveness through large-scale experiments. First, we rigorously prove that μP of mainstream diffusion Transformers, including DiT, U-ViT, PixArt-α, and MMDiT, aligns with that of the vanilla Transformer, enabling the direct application of existing μP methodologies. Leveraging this result, we systematically demonstrate that DiT-μP enjoys robust HP transferability. Notably, DiT-XL-2-μP with transferred learning rate achieves $2.9\times$ faster convergence than the original DiT-XL-2. Finally, we validate the effectiveness of μP on text-to-image generation by scaling PixArt-α from 0.04B to 0.61B and MMDiT from 0.18B to 18B. In both cases, models under μP outperform their respective baselines while requiring small tuning cost—only 5.5% of one training run for PixArt-α and 3% of consumption by human experts for MMDiT-18B. *These results establish μP as a principled and efficient framework for scaling diffusion Transformers.*

1 Introduction

Owing to its generality and scalability, diffusion Transformers [52; 1] have become the backbone of modern vision generation models, with widespread applications in various tasks such as image generation [55; 3; 16; 35; 20] and video generation [6; 81; 62; 2; 77]. As datasets grow and task complexity increases, further scaling of diffusion Transformers has become inevitable and is attracting increasing attention [41; 40; 29; 17; 75]. However, as model sizes reach billions of parameters, hyperparameter (HP) tuning becomes prohibitively expensive, often hindering the model from achieving its full potential. This underscores the urgent need for a principled approach to efficiently identify the optimal HPs when scaling diffusion Transformers.

Maximal Update Parametrization (μP) [73; 70; 72] was recently proposed as a promising solution to the HP selection problem for large-scale vanilla Transformer [61]. It stabilizes optimal HPs across different model widths, enabling direct transfer of HPs searched from small models to large models (a.k.a., μTransfer) and significantly reducing tuning costs at scale. Due to its strong transferability, μP has been applied to the pretraining of large language models (LLMs) [73; 14; 28; 43; 42; 82].

---

[*] Work done during an internship at ByteDance Seed.
[†] Correspondence to Chongxuan Li <chongxuanli@ruc.edu.cn>.

(a) Samples produced by the MMDiT-μP-18B.

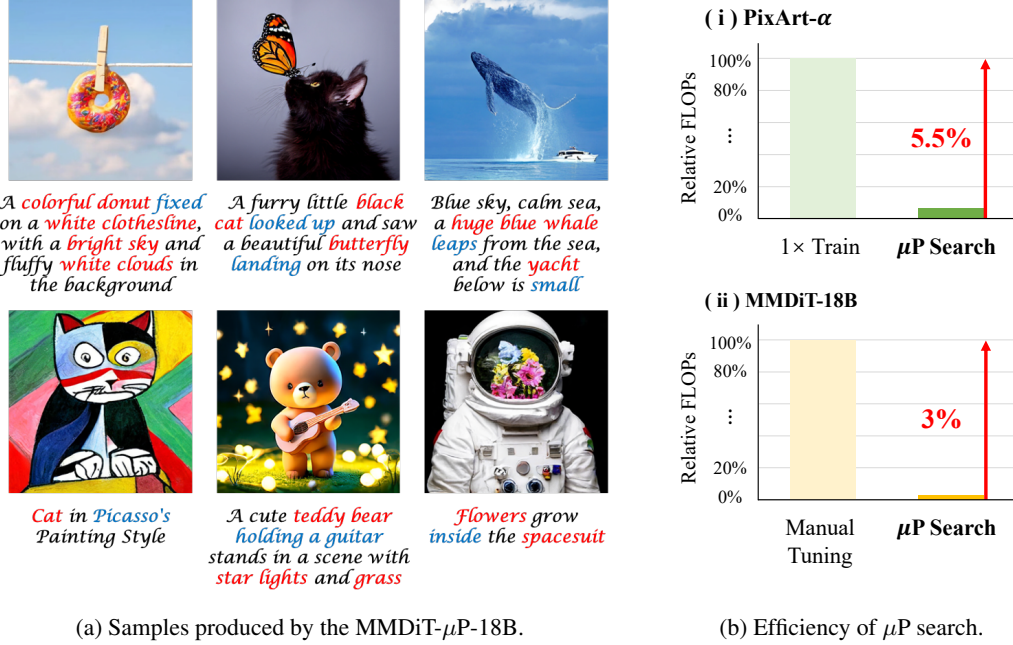(b) Efficiency of μP search.

Figure 1: **Visualization results and efficiency of HP search under μP.** (a) Samples generated by the MMDiT-μP-18B model exhibit strong fidelity and precision in aligning with the provided textual descriptions. (b) HP search for large diffusion Transformers is efficient under μP, requiring only 5.5% FLOPs of a single training run for PixArt-α and just 3% FLOPs of the human experts for MMDiT-18B.

Unfortunately, diffusion Transformers [52; 1; 16; 8] differ fundamentally from vanilla Transformers [61]. First, their architectures incorporate additional components to integrate text information and diffusion timesteps for final vision generation. Second, they operate under a distinct generative framework based on iterative denoising, in contrast to the autoregressive paradigms typically used in vanilla Transformers (e.g., LLMs) [22; 80]. As a result, existing μP theory and its associated HP transfer properties may not directly apply to diffusion Transformers. This paper systematically investigates this issue, as detailed below.

First, we extend the standard μP theory from vanilla Transformers to diffusion Transformers in Section 3.1. Using the Tensor Programs technique [72; 70; 67], we rigorously prove that the μP formulation of mainstream diffusion Transformers, including DiT [52], U-ViT [1], PixArt-α [8], and MMDiT from Stable Diffusion 3 [16], matches that of vanilla Transformers (Theorem 3.1). This compatibility enables us to directly apply existing practical methodologies developed for standard μP to diffusion Transformers, as described in Section 3.2.

Based on the rigorous μP result of diffusion Transformers, we then conduct a systematic study of DiT-μP on the image generation task using the ImageNet dataset [13], presented in Section 4. We first verify the stable HP transferability of DiT-μP across widths, batch sizes, and training steps. We then μTransfer the optimal learning rate searched from small models to DiT-XL-2-μP. Notably, DiT-XL-2-μP trained with the transferred learning rate achieves $2.9\times$ faster convergence than the original DiT-XL-2 [52], suggesting that μP offers an efficient principle for scaling diffusion Transformers.

Finally, we further validate the efficiency of μP on large-scale text-to-image generation tasks. In Section 5.1, we apply the μTransfer algorithm to PixArt-α [8], scaling the model from 0.04B to 0.61B. In Section 5.2, we apply it to MMDiT [16], scaling from 0.18B to 18B. In both cases, diffusion Transformers under μP consistently outperform their respective baselines with small HP tuning cost on proxy tasks. For PixArt-α, tuning consumes only 5.5% of the FLOPs required for a full pretraining run, while for MMDiT, tuning uses just 3% of the FLOPs typically consumed by human experts. These real-world experiments further confirm the scalability and reliability of μP.

2 Preliminaries

We begin by establishing the necessary background for diffusion Transformers and μP. Detailed discussion of additional related work is placed in Appendix A.

Table 1: μ**P for (diffusion) Transformers with Adam/AdamW optimizer.** We use purple text to highlight the differences between μP and standard parameterization (SP) in practice (e.g., Kaiming initialization [25]), and gray text to indicate the SP settings. Formal definitions of the weight type are provided in Appendix B.

	Input weights	Hidden weights	Output weights
a_W	0	0	1 (0)
b_W	0	$1/2$	0 ($1/2$)
c_W	0	1 (0)	0

2.1 Diffusion Transformers

Due to its superior scalability and compatibility, Transformers [52; 1; 16; 30] have replaced CNNs (e.g., U-Net [54; 53; 63]) as the backbone for advanced diffusion models [55; 3; 16; 35; 20]. U-ViT [1] firstly introduces Transformers with long skip connections between shallow and deep layers for various image generation tasks and achieves remarkable performance. After that, DiT [52] proposes Transformers with adaptive layer normalization (adaLN) blocks for class-to-image generation and demonstrates strong scalability with respect to network complexity. PixArt-α [8] extends DiT for text-conditional image generation by incorporating cross-attention for text features and an efficient shared adaLN-single block. MMDiT [16] further extends DiT by introducing two separate parameter sets for image and text modalities, along with a joint attention mechanism to facilitate multimodal interaction. As diffusion Transformers continue to scale, increasing attention is being paid to principled approaches for scaling [41; 40; 29; 75].

2.2 Maximal Update Parametrization

In this section, we provide a practical overview of μP, with a focus on the widely used AdamW optimizer [47]. The comprehensive review of the theoretical foundations of μP is in Appendix B.

μP identifies a unified parameterization that applies to common architectures expressive in NE$\otimes$OR$\top$ Program [72] (e.g., vanilla Transformer [61]), offering strong guidance for practical HP transfer across model widths, batch sizes, and training steps. Under μP, HPs can be tuned on a small proxy task (e.g., 0.04B parameters and 6B tokens in [73]) and directly transferred to a large-scale pretraining task (e.g., 6.7B parameters and 300B tokens in [73]), significantly reducing tuning costs at scale. As a result, μP has been widely adopted in the pretraining of LLMs [73; 14; 28; 43; 23; 42; 82].

Concretely, μP is implemented by analytically adjusting HPs with model width, typically involving the weight multiplier, initialization variance, and learning rate (a.k.a., abc-parameterization). Formally, let n denote the network width, we set each weight as $\boldsymbol{W} = \phi_W n^{-a_W} \widetilde{\boldsymbol{W}}$, where the trainable component $\widetilde{\boldsymbol{W}}$ is initialized as $\widetilde{W}_{ij} \sim \mathcal{N}(0, \sigma_W^2 n^{-2b_W})$, and its learning rate is $\eta_W n^{-c_W}$. Henceforth, we call the width-independent parts (ϕ_W, σ_W, η_W) as **base HPs**. As summarized in Table 1, μP identifies values of a_W, b_W, and c_W that enable models of different widths to share the (approximately) **same optimal base HPs** ϕ_W^*, σ_W^*, and η_W^*. μP adjusion for other HPs is placed in Appendix B.4.

The most related work to us is AuraFlow v0.1 [10], which applied μP empirically to an MMDiT-style model for learning rate transfer. Compared to AuraFlow v0.1, our work makes additional non-trivial contributions. First, we provide a rigorous theoretical proof for mainstream diffusion Transformers, thereby justifying the validity of μP for this family of models. Second, we systematically validate the HP transferability of diffusion Transformers under μP, across multiple widths, batch sizes, and training steps. Finally, for MMDiT in particular, we conduct a more extensive search over multiple HPs and scale the model up to 18B parameters, providing detailed intermediate results and comparisons.

3 Scaling Diffusion Transformers by μP

In this section, we extend the principles of μP to scale diffusion Transformers. In Section 3.1, we prove that despite fundamental differences from vanilla Transformer, mainstream diffusion Transformers—including U-ViT [1], DiT [52], PixArt-α [8], and MMDiT [16]—adhere to the standard μP formulation summarized in Table 1. Then, in Section 3.2, we introduce the practical methodology for applying μP to diffusion Transformers, as illustrated in Figure 2.
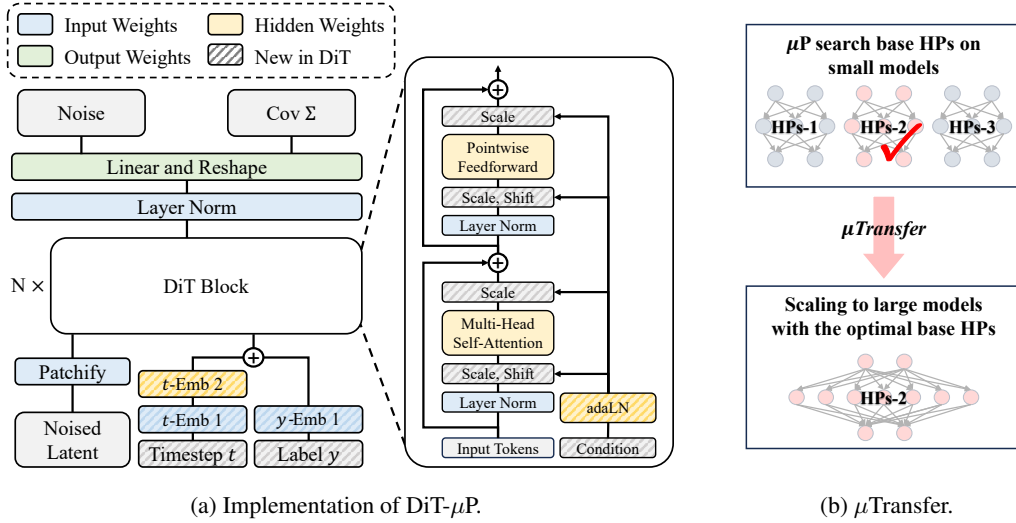
(a) Implementation of DiT-μP.

(b) μTransfer.

Figure 2: **A overview of applying μP to diffusion Transformers.** (a) We illustrate the implementation of μP for DiT as an example. The abc-parameterization of each weight is adjusted based on its type and visualized using different colors. Modules that differ from the vanilla Transformer are also highlighted. (b) We μTransfer the optimal base HPs searched from multiple trials on small models to pretrain the target large models.

3.1 μP of Diffusion Transformers

As mentioned in Section 2, the existing μP results [72; 73] in Table 1 apply only to architectures expressible as a NE⊗OR⊤ program. Therefore, it is crucial to determine whether the diffusion Transformers can be represented in this framework. The following theorem establishes that several prominent diffusion Transformers [52; 1; 8; 16] are compatible with the existing μP results.

Theorem 3.1 (μP of diffusion Transformers, proof in Appendix C). *The forward passes of mainstream diffusion Transformers (U-ViT [1], DiT [52], Pixart-α [8], and MMDiT [16]) can be represented within the NE⊗OR⊤ Program. Therefore, their μP matches the standard μP presented in Table 1.*

The proof of Theorem 3.1 relies on rewriting the forward pass of diffusion Transformers using the three operators defined in the NE⊗OR⊤ Program [72]. Given that the representability of standard Transformer [61] within the NE⊗OR⊤ Program has been established [67], we focus on demonstrating that the novel modules specific to diffusion Transformers can also be expressed in this formalism. These modules primarily serve to integrate text and diffusion timestep information for vision generation; examples include the adaLN blocks of DiT [52] in Figure 2a.

Theorem 3.1 ensures that existing practical methodologies developed for standard μP apply directly to DiT, U-ViT, PixArt-α, and MMDiT. Moreover, our analysis technique naturally extends to other variants of diffusion Transformers [49; 7; 19; 59; 48]. To the best of our knowledge, mainstream diffusion Transformer variants in use can be expressed within the NE⊗OR⊤ Program.

3.2 Practical Methodology of μP in Diffusion Transformers

Given the rigorous μP result for diffusion Transformers established in Theorem 3.1, this section introduces how to apply μP to diffusion Transformers in practice. In addition, our code is available at *https://github.com/ML-GSAI/Scaling-Diffusion-Transformers-muP*.

3.2.1 Implementation of μP

The width of a multi-head attention layer is determined by the product of the head dimension and the number of attention heads. Thus, there are two degrees of freedom when scaling the width of diffusion Transformers. Theoretically, recent theoretical advances [5] reveal an important difference: when the head dimension tends to infinity, multi-head self-attention can collapse to single-head self-attention, losing the diversity of attention patterns. In contrast, scaling the number of heads avoids this degeneracy and preserves the expressivity of the multi-head mechanism. Empirically, well-known models in real-world large-scale practice [73; 22; 43; 28] favor increasing the number of

heads rather than the head dimension. Given these theoretical and empirical insights, we fix the head dimension and scale the number of heads in this work.

Given scaling the number of heads, we implement the μP of diffusion Transformers following the standard procedure in [73].[3] Specifically, we replace the vanilla width n in the abc-parameterization (see Section 2) with the width ratio n/n_{base} when standard parameterization (SP) and μP differ in Table 1, where n_{base} is a fixed base width. This implementation remains consistent with Table 1 since n_{base} is a constant. For example, for a hidden weight matrix $\boldsymbol{W}$ with width n and base HPs ϕ_W, σ_W^2, η (maybe searched from proxy model), we set $\boldsymbol{W} = \phi_W \widetilde{\boldsymbol{W}}$ with initial $\widetilde{\boldsymbol{W}} \sim \mathcal{N}(0, \sigma_W^2/n)$, as in SP. In contrast, its learning rate is $\eta n_{\text{base}}/n$ rather than η/n, where μP differs from SP. Finally, we also follow the suggestion from [73] to initialize the output weights by zero (i.e., $\sigma_{\text{out}}^2 = 0$).

3.2.2 Base Hyperparameter Transferability of μP

Prior work has shown that μP provides strong guidance for base HP transferability in image classification and language modeling tasks [73; 28; 64; 4; 24; 60; 14]. Building on the rigorous μP form for diffusion Transformers in Theorem 3.1, we further validate its HP transferability in the context of visual generation. We summarize the methodology for verifying base HP transferability across widths, batch sizes, and training steps in Algorithm 1, 2, and 3 in Appendix D, respectively. We describe how to verify HP transfer across widths below; those for batch size and step are similar.

For simplicity, we describe how to evaluate the transferability of the base learning rate η across widths; similar procedures apply to other HPs, such as the multiplier ϕ. We define $\{n_i\}_{i=1}^P$ and $\{\eta_j\}_{j=1}^R$ as the sets of widths and base learning rates used in the evaluation. Given a fixed batch size B and training iterations of T, we train PR diffusion Transformers, each parameterized by μP with base width n_{base}, true width n_i, and base learning rate η_j. Finally, given evaluation metrics, if models at different widths (approximately) share the same optimal base learning rate, we conclude that learning rate transferability holds for diffusion Transformers. In Section 4, we verify that diffusion Transformers under the μP exhibit robust transferability of base HPs.

3.2.3 μTransfer from Proxy Task to Target Task

Once base HP transferability is validated for diffusion Transformers, we can directly apply the μTransfer algorithm [73] (see Algorithm 4 in Appendix D) to zero-shot transfer base HPs from a proxy task to a target task. Specifically, both the proxy model with width n_{proxy} and the target model with width n_{target} are parameterized by μP using the same base width n_{base}. We first search for an optimal combination of base HPs using various proxy models trained with a small batch size B_{proxy} and a limited training steps T_{proxy}. These optimal base HPs are then used to train the large target model with a larger batch size B_{target} and longer training iteration T_{target}. Experimental results in Section 5 show the superior performance of the μTransfer in real-world vision generation.

4 Systematic Investigation for DiT-μP on ImageNet

In this section, we first empirically verify the base HP transferability of DiT [52] under μP. We then μTransfer the optimal learning rate to train DiT-XL-2-μP, which achieves $2.9\times$ faster convergence.

4.1 Basic Experimental Settings

To ensure a fair comparison between DiT-μP and the original DiT [52], we adopt the default configurations from [52] and describe the basic setup in detail below.

Dataset. We train DiT and DiT-μP on the ImageNet training set [13], which contains 1.28M images across 1,000 classes. All images are resized to a resolution of 256×256 during training, following standard practice in generative modeling benchmarks [52; 1; 49].

Architecture of DiT-μP. The architecture of DiT-μP models is identical to that of DiT-XL-2, except for the width. We fix the attention head dimension at 72 (as in DiT-XL-2) and vary the number of heads. The base width n_{base} in the μP setup corresponds to 288, which uses 4 attention heads.

---

[3]To scaling head dimension, Yang et al. [73] additionally change the calculation of attention logit from $\boldsymbol{q}^\top \boldsymbol{k}/\sqrt{d}$ to $\boldsymbol{q}^\top \boldsymbol{k}/d$, where query $\boldsymbol{q}$ and key $\boldsymbol{k}$ have dimension d. We do not use this modification [5].

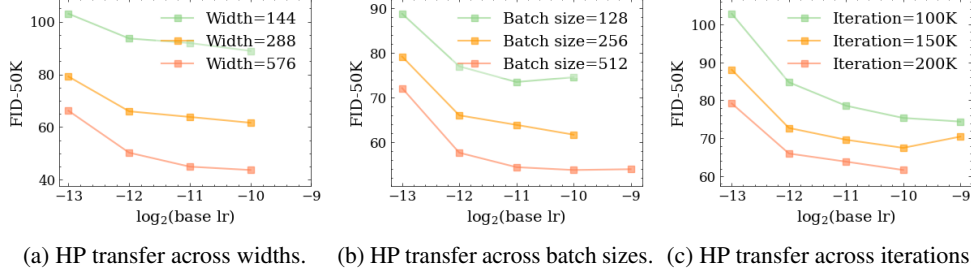(a) HP transfer across widths.   (b) HP transfer across batch sizes.   (c) HP transfer across iterations.

Figure 3: **DiT-μP enjoys base HP transferability**. Unless otherwise specified, we use a model width of 288, a batch size of 256, and a training iteration of 200K. Missing data points indicate training instability, where the loss explodes. Under μP, the base learning rate can be transferred across model widths, batch sizes, and steps.

Training. We train DiT and DiT-μP using the AdamW [47]. Following the original DiT setup [52], we do not apply any learning rate schedule or weight decay, and constant learning rates are used in all experiments. The original DiT-XL-2 is trained with a learning rate 10^{-4} and a batch size of 256.

Evaluation metrics. To comprehensively evaluate generation performance, we report FID [26], sFID [50], Inception Score [56], precision, and recall [34] on 50K generated samples without classifier-free guidance (cfg), as in Table 4 of [52]. In the main text, we present the FID results, while the remaining metrics are provided in Appendix E.2.

4.2 Base Hyperparameters Transferability of DiT-μP

In this section, we evaluate the HP transferability of DiT-μP across different widths, batch sizes, and training steps. We focus primarily on the base learning rate, as it has the most significant impact on performance [73; 28; 14; 43]. *Similar results for other HP (weight multiplier) are provided in Appendix E.2*. We sweep the base learning rate over the set $\{2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}\}$ across various widths, batch sizes, and training steps. FID-50K results are shown in Figure 3, and comprehensive results for other metrics are presented in Tables 8, 9, and 10 in Appendix E.2.

As presented in Figure 3, the optimal base learning rate 2^{-10} generally transfers across scaling dimensions when some minimum width (e.g. 144), batch size (e.g., 256), and training steps (e.g., 150K) are satisfied, which verifies the base HP transferability of DiT-μP. Interestingly, we observe that neural networks with μP tend to favor a large learning rate close to the maximum stable value (e.g., see Figure 3a). This aligns with empirical findings and theoretical insights reported for standard neural networks trained for multiple epochs [65; 44; 37; 38; 11], which suggest that larger learning rates introduce beneficial gradient noise to help guide optimization towards flatter minima that generalize better. Our empirical results suggest that the optimization landscape of μP shares certain similarities with that of SP, offering a direction for future theoretical investigation.

4.3 Scaling Performance of DiT-μP

We μTransfer the optimal base learning rate of 2^{-10} to train DiT-XL-2-μP with a width of 1152. The batch size is set to 256, following [52]. We evaluate DiT-μP every 400K steps without using cfg. Training continues until DiT-μP surpasses the best performance reported by the original DiT at final 7M steps [52]. To enable a detailed comparison throughout the training, we also reproduce the original DiT training using its official codebase. Complete evaluation results throughout training are provided in Table 14 in Appendix E.2.

As shown in Figure 4, DiT-XL-2-μP with the transferred base learning rate performs effectively, achieving $2.9\times$ faster convergence (2.4M steps) compared to the original DiT (7M steps). These results suggest that μP offers a simple and promising approach to improve the pretraining of large-scale diffusion Transformers. In the following sections, we further validate this claim through text-to-image generation tasks.
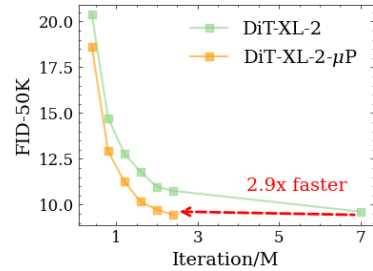

Figure 4: **μP accelerates the training of diffusion Transformers.** Considering FID-50K, DiT-XL-2-μP with transferred learning rate achieves $2.9\times$ faster convergence than the original DiT-XL-2 and a slightly better result.

5 Large-Scale Text-to-Image Generation

In this section, we verify the efficiency of μTransfer algorithm on real-world text-to-image generation tasks. Diffusion Transformers under μP outperform the baselines while requiring small tuning cost.

5.1 Scaling PixArt-α-μP on SA-1B

In this section, we perform μTransfer experiments on the PixArt-α [8], scaling from 0.04B to 0.61B parameters. Using the same pretraining setup, PixArt-α-μP with the transferred learning rate outperforms the original PixArt-α, while incurring only 5.5% FLOPs of one full pretraining run.

5.1.1 Experimental Settings

To ensure the fairest possible comparison between PixArt-α-μP and the original PixArt-α [8], we mainly adopt the original setup [8] and summarize the key components below.

Dataset. We use the SAM/SA-1B dataset [33], which contains 11M high-quality images curated for segmentation tasks with diverse object-rich scenes. For text captions, we use the SAM-LLaVA annotations released in [8]. All images are resized to a resolution of 256×256 during training.

Architecture of PixArt-α-μP models. The target PixArt-α-μP model adopts the same architecture as PixArt-α (0.61B parameters). The proxy model also follows the same architecture, differing only in width. To construct the proxy PixArt-α-μP model, we fix the attention head dimension at 72 (as in PixArt-α) and reduce the number of heads from 16 to 4 (0.04B parameters). In the μP framework, the base width n_{base} (see Section 3.2.1) is set to the proxy width of 288.

Training. PixArt-α is implemented using the official codebase and original configuration. We train the original PixArt-α and the target PixArt-α-μP model for 30 epochs with a batch size of 176×32 (approximately 59K steps).[4] The small proxy PixArt-α-μP models are trained for 5 epochs with a batch size of 176×8 (approximately 39K steps). *Notably, the model width, batch size, and training steps are all smaller than those used in the target pretraining setting.*

Hyperparameter search. In this section, we focus solely on the base learning rate. We search over the candidate set $\{2^{-13}, 2^{-12}, 2^{-11}, 2^{-10}, 2^{-9}\}$ as in Section 4, resulting in five proxy training trials.

Ratio of tuning cost to pretraining cost. We consider the FLOPs as the metric for the computational cost, then the ratio of tuning cost to pretraining cost can be estimated as

$$\texttt{ratio} = \frac{R S_{\text{proxy}} E_{\text{proxy}}}{S_{\text{target}} E_{\text{target}}} = \frac{R S_{\text{proxy}} B_{\text{proxy}} T_{\text{proxy}}}{S_{\text{target}} B_{\text{target}} T_{\text{target}}}, \tag{1}$$

where R is the number of HP search trials, S is the number of parameters, E is the training epochs, B is the batch size and T is the training iteration. The $\texttt{ratio} \approx 5.5\%$ here, as detailed in Appendix E.3.

Evaluation metrics. We evaluate text-to-image generation performance following standard practice [8; 16; 66; 1], including FID, CLIP Score, and GenEval [21]. Both FID and CLIP Score are computed on the aesthetic MJHQ-30K [39] and real MS-COCO-30K [45] datasets. MJHQ-30K contains 30K images generated by Midjourney, while MS-COCO-30K is a randomly sampled subset of the MS-COCO [45] dataset. GenEval evaluates text-image alignment using 533 test prompts. Following the official implementation [8], we use a cfg of 4.5 to generate samples.

5.1.2 Experimental Results of PixArt-α-μP

We begin by conducting a base learning rate search using the PixArt-α-μP proxy models. The evaluation results for different base learning rates are summarized in Table 2. Details of GenEval results can be found in Table 15 in Appendix E.3. Since overfitting is not observed in this setting, we include training loss as an additional evaluation metric. Overall, the base learning rate of 2^{-10} yields the best performance. Interestingly, this optimal learning rate matches that of DiT-μP (see Figure 3a). We hypothesize that this consistency arises from the architectural similarity between the two models and the fact that both the ImageNet and SAM datasets consist of real-world images. This observation suggests that optimal base HPs may exhibit some degree of transferability across different datasets and architectures.

---

[4]We confirmed with the authors that this setup is reasonable; longer training may result in overfitting.

Table 2: **Results of base learning rate search on PixArt-α-μP proxy tasks.** 0.04B proxy models with different base learning rates are trained for 5 epochs on the SAM dataset. Overall, the base learning rate 2^{-10} is optimal.

$\log_2(\text{lr})$	Training loss ↓	GenEval ↑	FID-30K (MS-COCO) ↓	FID-30K (MJHQ) ↓
-9	NaN	NaN	NaN	NaN
-10	**0.1493**	**0.083**	**47.46**	47.71
-11	0.1494	0.078	49.24	**46.31**
-12	0.1504	0.030	66.77	63.36
-13	0.1536	0.051	60.28	60.93

Table 3: **Comprehensive comparison between PixArt-α-μP and PixArt-α.** Both models are trained on the SAM dataset for 30 epochs. PixArt-α-μP (0.61B), using a base learning rate transferred from the optimal 0.04B proxy model, consistently outperforms the original baseline throughout the training process.

Epoch	Method	GenEval ↑	MJHQ		MS-COCO	
			FID-30K ↓	CLIP Score ↑	FID-30K ↓	CLIP Score ↑
10	PixArt-α [8]	0.19	38.36	25.78	34.58	28.12
	PixArt-α-μP (**Ours**)	**0.20**	**33.35**	**26.25**	**29.68**	**28.87**
20	PixArt-α [8]	0.20	35.68	26.54	30.13	28.81
	PixArt-α-μP (**Ours**)	**0.23**	**33.42**	**26.83**	**29.05**	**29.53**
30	PixArt-α [8]	0.15	42.71	26.25	37.61	28.91
	PixArt-α-μP (**Ours**)	**0.26**	**29.96**	**27.13**	**25.84**	**29.58**

We then apply μTransfer by transferring the searched optimal base learning rate of 2^{-10} to train the target PixArt-α. A comparison between PixArt-α and PixArt-α-μP throughout training is provided in Table 3 (with the complete results shown in Table 16 in Appendix E.3). The results demonstrate that PixArt-α-μP consistently outperforms PixArt-α across all evaluation metrics during training, supporting μP as an efficient and robust approach for scaling diffusion Transformers. Furthermore, we observe that the benchmark performance of PixArt-α degrades after 20 epochs, primarily due to overfitting. In contrast, PixArt-α-μP continues to improve, suggesting that μP enhances the model's generalization ability, which offers an interesting direction for future theoretical investigation.

Specifically, we emphasize that our current experimental results validate that μP works well without cfg (DiT) and with cfg (PixArt-α). Because cfg is an important factor during inference and affects the optimal hyperparameters, we strongly recommend practitioners align the evaluation of proxy models and that of the target model. In the following section, we further extend the current method to large-scale applications.

5.2 Scaling MMDiT to 18B

In this section, we validate the efficiency of μP in the large-scale setup. We scale up the MMDiT [16] architecture from 0.18B to 18B. Under the same pretraining setup, MMDiT-μP-18B with the transferred base HPs outperforms the MMDiT-18B tuned by human algorithmic experts.

5.2.1 Experimental Settings

Dataset. We train models on an internally constructed dataset comprising 820M high-quality image-text pairs. All images are resized to a resolution of 256×256 during training.

Baseline MMDiT-18B. The width and depth of MMDiT-18B are 5,120 and 16, respectively. The training objective combines a flow matching loss [46; 49] and a representation alignment (REPA) loss [76]. The model is optimized by AdamW [47], with a batch size of 4,096 and 200K training iterations. The learning rate schedule is constant with a warm-up duration. The HPs were tuned by algorithmic experts, requiring roughly 5 times the cost of full pretraining, as detailed in Appendix E.4.
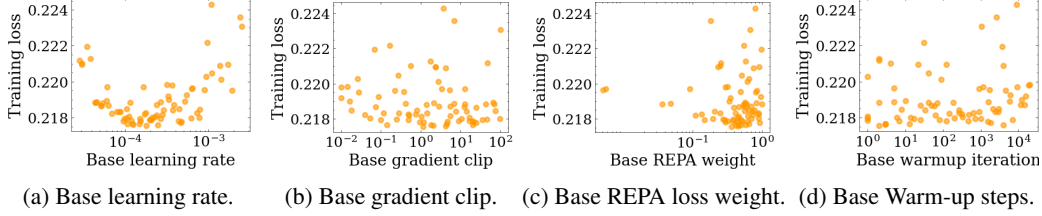
(a) Base learning rate.	(b) Base gradient clip.	(c) Base REPA loss weight.	(d) Base Warm-up steps.

Figure 5: **Results of base HP search on proxy MMDiT-μP tasks.** We train 0.18B MMDiT-μP proxy models with 80 different base HPs settings. The optimal base HPs are transferred to the training of 18B target model.

Architecture of MMDiT-μP models. The target MMDiT-μP-18B model shares the same architecture as MMDiT-18B. The proxy model also follows this architecture, differing only in width by reducing the number of attention heads. It contains 0.18B parameters (1% **of the target model**) with a width of 512. In the μP setup, the base width n_{base} is set to 1,920 (see Section 3.2.1 for definition of n_{base}).

Training of MMDiT-μP models. The training procedure for the target MMDiT-μP-18B is identical to that of the baseline MMDiT-18B models, except for the selected HPs. The proxy models are trained for 30K steps with a batch size of 4,096. We conduct 80 searches over four base HPs. Concretely, we uniformly sample base learning rate from 2.5×10^{-5} to 2.5×10^{-3}, gradient clipping from 0.01 to 100, weight of REPA loss from 0.1 to 1, and warm-up iteration from 1 to 20K. To further verify that 30K iterations are enough for the proxy task, we conducted five proxy training runs (100K steps each) using the searched optimal HPs and different learning rates, as detailed in Appendix E.4.

As derived in Appendix E.4 based on Equation (1), the total tuning FLOPs under μP amounts to 14.5% of one full pretraining cost, and thus only 3% of the human-tuned cost. We think the batch size and iterations used during the base HP search could be further reduced to lower the tuning cost. However, due to limited resources, we are unable to explore additional setups in this work.

Evaluation metrics. We use training loss to select base HPs on proxy tasks, as the dataset is passed through only once, and overfitting does not occur. We follow the standard practice in the μP literature [73; 28; 14], where the base hyperparameter is typically selected by identifying the value that aligns with the lowest envelope of the training loss plot. To assess the final text-to-image generation, in addition to the standard GenEval benchmark [21], we also created an internal benchmark comprising 150 prompts to comprehensively evaluate text-image alignment. Each prompt includes an average of five binary alignment checks (yes or no), covering a wide range of concepts such as nouns, verbs, adjectives (e.g., size, color, style), and relational terms (e.g., spatial relationships, size comparisons). Ten human experts conducted the evaluation, with each prompt generating three images, resulting in a total of 22,500 alignment tests. The final score is computed as the average correctness across all test points. The details can be found in Appendix E.4.

5.2.2 Analyzing the Results of the Random Search

The visualization of the results of the base HP search is shown in Figure 5. First, the base learning rate has the most significant impact on training loss. In our case, we observed that the envelope near 2.5×10^{-4} was stable and close to optimal, so we chose 2.5×10^{-4} as the optimal value. Interestingly, unlike the DiT and PixArt setups, the optimal learning rate here is not close to the maximum stable value. This highlights a key difference between multi-epoch training in traditional deep learning and the single-epoch training in large model pretraining [65]. Intuitively, since the gradient signal for any individual sample is not revisited, the training must be more conservative to maintain stability. Second, the optimal gradient clipping value is 1, which deviates from the common practice of using a small value (e.g., 0.1) to stabilize pretraining. Intuitively, μP favors a larger clipping value, as aggressive clipping can undermine the maximal update property central to μP. Third, the optimal weight for the REPA loss is determined to be around 0.5, consistent with the experience from existing works [76]. Finally, the warm-up iteration has a negligible impact on the training loss, so we adopt the default value of 1K in the pretraining of the target model.

5.2.3 μTransfer Results of MMDiT-μP

The comparisons of training losses, GenEval results, and human evaluations are shown in Figure 6, Table 4, and Table 5, respectively. In addition, the complete comparison of training losses and detailed

Table 4: **GenEval results of pretrained MMDiT-18B and MMDiT-μP-18B models.** MMDiT-μP-18B achieves better benchmark results with only 3% of the manual tuning cost.

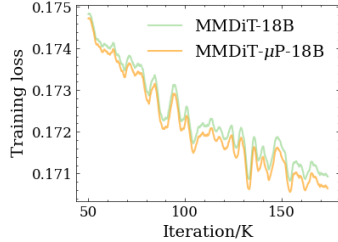| Method	Overall ↑	Single	Two	Counting	Colors	Position	Color Attribution
MMDiT-18B	0.8154	99.38	93.69	**81.88**	88.03	57.5	**68.75**
MMDiT-μP-18B	**0.8218**	99.38	**94.44**	79.69	**88.83**	**62.25**	68.5

Figure 6: **MMDiT-μP-18B achieves consistently lower training loss** than baseline after 15K steps.

Table 5: **Results of human evaluation for text-image alignment.** The alignment accuracy (acc.) is computed as the average over 22,500 human alignment tests. MMDiT-μP-18B achieves superior results with only 3% of the manual tuning cost.

Method	Alignment acc. ↑
MMDiT-18B	0.703
MMDiT-μP-18B (**Ours**)	**0.715**

visualizations are provided in Figure 8 and Figure 10 in Appendix E.4. As a result, MMDiT-μP-18B outperforms the baseline MMDiT-18B in all cases, achieving this with only 3% FLOPs of the standard manual tuning cost. These results demonstrate that μP is a reliable principle for scaling diffusion Transformers. *As models grow larger and standard HP tuning becomes prohibitively expensive, μP offers a scientifically grounded framework to unlock the full potential of large models.*

6 Discussion

There are several promising research directions building on this work. First, the principles of μP could be extended to diffusion models with more advanced and efficient architectures, such as linear transformers [66] and mixture-of-expert models [17]. Second, μP can be applied to more sophisticated optimization algorithms, including the Muon optimizer [31] and the warmup-stable-decay learning schedule [28]. Third, while our experiments suggest that a proxy model width of 256–512 and a proxy dataset size of 1/10–1/6 of the full pretraining data are sufficient for stable HP transfer in diffusion Transformers, determining the optimal proxy task size that balances tuning cost and target model performance remains an important avenue for future work. Finally, developing a learning-theoretic framework to explain the optimization dynamics [58; 51], generalization behavior [9], and downstream performance [79; 78] of diffusion Transformers under μP would be both meaningful and impactful. In summary, this provides the vision community with an initial principled approach for scaling diffusion Transformers efficiently.

Broader Impacts and Limitations. Our work has the potential to accelerate progress in generative modeling applications using diffusion Transformers, including text-to-image and video generation. However, improvements in scaling diffusion Transformers could also facilitate the creation of deepfakes for disinformation. Regarding the limitations of this work, although we demonstrate the efficiency of μP in large-scale applications, we do not identify the optimal proxy task size that balances HP tuning cost and target model performance, due to limited computational resources.

7 Conclusion

In this paper, we extend μP from standard Transformers to diffusion Transformers. By proving that mainstream diffusion Transformers share the same μP form as vanilla Transformers, we enable direct application of existing μP practice and verify the reliable base HP transfer from small to large diffusion Transformers. This leads to practical performance gains on DiT-XL-2, PixArt-α, and MMDiT-18B, while requiring a small fraction of the usual tuning effort (e.g., 3% for MMDiT-18B). Our results establish μP as a principled and efficient scaling strategy for diffusion Transformers.

Acknowledgments

References

[1] Fan Bao, Shen Nie, Kaiwen Xue, Yue Cao, Chongxuan Li, Hang Su, and Jun Zhu. All are worth words: A vit backbone for diffusion models. In *CVPR*, pages 22669–22679, 2023.

[2] Fan Bao, Chendong Xiang, Gang Yue, Guande He, Hongzhou Zhu, Kaiwen Zheng, Min Zhao, Shilong Liu, Yaole Wang, and Jun Zhu. Vidu: a highly consistent, dynamic and skilled text-to-video generator with diffusion models. *arXiv preprint arXiv:2405.04233*, 2024.

[3] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, et al. Improving image generation with better captions. *Computer Science. https://cdn. openai. com/papers/dall-e-3. pdf*, 2(3):8, 2023.

[4] Charlie Blake, Constantin Eichenberg, Josef Dean, Lukas Balles, Luke Y. Prince, Björn Deiseroth, Andrés Felipe Cruz-Salinas, Carlo Luschi, Samuel Weinbach, and Douglas Orr. u-μp: The unit-scaled maximal update parametrization. *CoRR*, abs/2407.17465, 2024.

[5] Blake Bordelon, Hamza Tahir Chaudhry, and Cengiz Pehlevan. Infinite limits of multi-head transformer dynamics. In *NeurIPS*, 2024.

[6] Tim Brooks, Bill Peebles, Connor Holmes, Will DePue, Yufei Guo, Li Jing, David Schnurr, Joe Taylor, Troy Luhman, Eric Luhman, et al. Video generation models as world simulators. *OpenAI Blog*, 1:8, 2024.

[7] Junsong Chen, Chongjian Ge, Enze Xie, Yue Wu, Lewei Yao, Xiaozhe Ren, Zhongdao Wang, Ping Luo, Huchuan Lu, and Zhenguo Li. Pixart-Σ: Weak-to-strong training of diffusion transformer for 4k text-to-image generation. In *ECCV*, volume 15090, pages 74–91, 2024.

[8] Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Zhongdao Wang, James T. Kwok, Ping Luo, Huchuan Lu, and Zhenguo Li. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. In *ICLR*, 2024.

[9] Zixiang Chen, Yuan Cao, Quanquan Gu, and Tong Zhang. A generalized neural tangent kernel analysis for two-layer neural networks. *Advances in Neural Information Processing Systems*, 33:13363–13373, 2020.

[10] cloneofsimo and Team fal. Introducing auraflow v0.1: An open exploration of large rectified flow models. `https://blog.fal.ai/auraflow/`, 2024.

[11] Jeremy Cohen, Simran Kaur, Yuanzhi Li, J. Zico Kolter, and Ameet Talwalkar. Gradient descent on neural networks typically occurs at the edge of stability. In *ICLR*, 2021.

[12] Katherine Crowson, Stefan Andreas Baumann, Alex Birch, Tanishq Mathew Abraham, Daniel Z. Kaplan, and Enrico Shippole. Scalable high-resolution pixel-space image synthesis with hourglass diffusion transformers. In *ICML*, 2024.

[13] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *CVPR*, pages 248–255, 2009.

[14] Nolan Dey, Gurpreet Gosal, Zhiming Chen, Hemant Khachane, William Marshall, Ribhu Pathria, Marvin Tom, and Joel Hestness. Cerebras-gpt: Open compute-optimal language models trained on the cerebras wafer-scale cluster. *CoRR*, abs/2304.03208, 2023.

[15] Prafulla Dhariwal and Alexander Quinn Nichol. Diffusion models beat gans on image synthesis. In *NeurIPS*, pages 8780–8794, 2021.

[16] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, Dustin Podell, Tim Dockhorn, Zion English, and Robin Rombach. Scaling rectified flow transformers for high-resolution image synthesis. In *ICML*, 2024.

[17] Zhengcong Fei, Mingyuan Fan, Changqian Yu, Debang Li, and Junshi Huang. Scaling diffusion transformers to 16 billion parameters. *CoRR*, abs/2407.11633, 2024.

[18] Peng Gao, Le Zhuo, Dongyang Liu, Ruoyi Du, Xu Luo, Longtian Qiu, Yuhang Zhang, Chen Lin, Rongjie Huang, Shijie Geng, et al. Lumina-t2x: Transforming text into any modality, resolution, and duration via flow-based large diffusion transformers. *arXiv preprint arXiv:2405.05945*, 2024.

[19] Shanghua Gao, Pan Zhou, Ming-Ming Cheng, and Shuicheng Yan. Masked diffusion transformer is a strong image synthesizer. In *ICCV*, pages 23107–23116, 2023.

[20] Yu Gao, Lixue Gong, Qiushan Guo, Xiaoxia Hou, Zhichao Lai, Fanshi Li, Liang Li, Xiaochen Lian, Chao Liao, Liyang Liu, et al. Seedream 3.0 technical report. *arXiv preprint arXiv:2504.11346*, 2025.

[21] Dhruba Ghosh, Hannaneh Hajishirzi, and Ludwig Schmidt. Geneval: An object-focused framework for evaluating text-to-image alignment. In *NeurIPS*, 2023.

[22] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[23] Ishaan Gulrajani and Tatsunori B. Hashimoto. Likelihood-based diffusion language models. In *NeurIPS*, 2023.

[24] Moritz Haas, Jin Xu, Volkan Cevher, and Leena Chennuru Vankadara. μp^2: Effective sharpness aware minimization requires layerwise perturbation scaling. In *NeurIPS*, 2024.

[25] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *ICCV*, pages 1026–1034, 2015.

[26] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *Advances in Neural Information Processing Systems*, pages 6626–6637, 2017.

[27] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8): 1735–1780, 1997.

[28] Shengding Hu, Yuge Tu, Xu Han, Chaoqun He, Ganqu Cui, Xiang Long, Zhi Zheng, Yewei Fang, Yuxiang Huang, Weilin Zhao, Xinrong Zhang, Zhen Leng Thai, Kai Zhang, Chongyi Wang, Yuan Yao, Chenyang Zhao, Jie Zhou, Jie Cai, Zhongwu Zhai, Ning Ding, Chao Jia, Guoyang Zeng, Dahai Li, Zhiyuan Liu, and Maosong Sun. Minicpm: Unveiling the potential of small language models with scalable training strategies. *CoRR*, abs/2404.06395, 2024.

[29] Zhongzhan Huang, Pan Zhou, Shuicheng Yan, and Liang Lin. Scalelong: Towards more stable training of diffusion model via scaling network long skip connection. In *NeurIPS*, 2023.

[30] Jiarui Jiang, Wei Huang, Miao Zhang, Taiji Suzuki, and Liqiang Nie. Unveil benign overfitting for transformer in vision: Training dynamics, convergence, and generalization. *Advances in Neural Information Processing Systems*, 37:135464–135625, 2024.

[31] Keller Jordan, Yuchen Jin, Vlado Boza, You Jiacheng, Franz Cecista, Laker Newhouse, and Jeremy Bernstein. Muon: An optimizer for hidden layers in neural networks, 2024. *URL https://kellerjordan. github. io/posts/muon*, 6, 2024.

[32] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *ICLR*, 2015.

[33] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloé Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross B. Girshick. Segment anything. In *ICCV*, pages 3992–4003, 2023.

[34] Tuomas Kynkäänniemi, Tero Karras, Samuli Laine, Jaakko Lehtinen, and Timo Aila. Improved precision and recall metric for assessing generative models. In *Advances in Neural Information Processing Systems*, pages 3929–3938, 2019.

[35] Black Forest Labs. Flux. `https://github.com/black-forest-labs/flux`, 2024.

[36] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998.

[37] Jaehoon Lee, Samuel S. Schoenholz, Jeffrey Pennington, Ben Adlam, Lechao Xiao, Roman Novak, and Jascha Sohl-Dickstein. Finite versus infinite neural networks: an empirical study. In *NeurIPS*, 2020.

[38] Aitor Lewkowycz, Yasaman Bahri, Ethan Dyer, Jascha Sohl-Dickstein, and Guy Gur-Ari. The large learning rate phase of deep learning: the catapult mechanism. *CoRR*, abs/2003.02218, 2020.

[39] Daiqing Li, Aleks Kamko, Ehsan Akhgari, Ali Sabet, Linmiao Xu, and Suhail Doshi. Playground v2.5: Three insights towards enhancing aesthetic quality in text-to-image generation. *CoRR*, abs/2402.17245, 2024.

[40] Hao Li, Shamit Lal, Zhiheng Li, Yusheng Xie, Ying Wang, Yang Zou, Orchid Majumder, R. Manmatha, Zhuowen Tu, Stefano Ermon, Stefano Soatto, and Ashwin Swaminathan. Efficient scaling of diffusion transformers for text-to-image generation. *CoRR*, abs/2412.12391, 2024.

[41] Hao Li, Yang Zou, Ying Wang, Orchid Majumder, Yusheng Xie, R. Manmatha, Ashwin Swaminathan, Zhuowen Tu, Stefano Ermon, and Stefano Soatto. On the scalability of diffusion-based text-to-image generation. In *CVPR*, pages 9400–9409, 2024.

[42] Xiang Li, Yiqun Yao, Xin Jiang, Xuezhi Fang, Xuying Meng, Siqi Fan, Peng Han, Jing Li, Li Du, Bowen Qin, Zheng Zhang, Aixin Sun, and Yequan Wang. FLM-101B: an open LLM and how to train it with \$100k budget. *CoRR*, abs/2309.03852, 2023.

[43] Xiang Li, Yiqun Yao, Xin Jiang, Xuezhi Fang, Chao Wang, Xinzhang Liu, Zihan Wang, Yu Zhao, Xin Wang, Yuyao Huang, Shuangyong Song, Yongxiang Li, Zheng Zhang, Bo Zhao, Aixin Sun, Yequan Wang, Zhongjiang He, Zhongyuan Wang, Xuelong Li, and Tiejun Huang. Tele-flm technical report. *CoRR*, abs/2404.16645, 2024.

[44] Yuanzhi Li, Colin Wei, and Tengyu Ma. Towards explaining the regularization effect of initial large learning rate in training neural networks. In *NeurIPS*, pages 11669–11680, 2019.

[45] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *ECCV*, volume 8693, pages 740–755, 2014.

[46] Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matthew Le. Flow matching for generative modeling. In *ICLR*, 2023.

[47] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*. OpenReview.net, 2019.

[48] Zeyu Lu, Zidong Wang, Di Huang, Chengyue Wu, Xihui Liu, Wanli Ouyang, and Lei Bai. Fit: Flexible vision transformer for diffusion model. In *ICML*, 2024.

[49] Nanye Ma, Mark Goldstein, Michael S. Albergo, Nicholas M. Boffi, Eric Vanden-Eijnden, and Saining Xie. Sit: Exploring flow and diffusion-based generative models with scalable interpolant transformers. In *ECCV*, volume 15135, pages 23–40, 2024.

[50] Charlie Nash, Jacob Menick, Sander Dieleman, and Peter W. Battaglia. Generating images with sparse representations. In *ICML*, volume 139, pages 7958–7968. PMLR, 2021.

[51] Atsushi Nitanda, Denny Wu, and Taiji Suzuki. Convex analysis of the mean field langevin dynamics. In *AISTATS*, volume 151, pages 9741–9757, 2022.

[52] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, pages 4172–4182. IEEE, 2023.

[53] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: Improving latent diffusion models for high-resolution image synthesis. *arXiv preprint arXiv:2307.01952*, 2023.

[54] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, volume 9351, pages 234–241, 2015.

[55] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L. Denton, Seyed Kamyar Seyed Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, Jonathan Ho, David J. Fleet, and Mohammad Norouzi. Photorealistic text-to-image diffusion models with deep language understanding. In *NeurIPS*, 2022.

[56] Tim Salimans, Ian J. Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training gans. In *Advances in Neural Information Processing Systems*, pages 2226–2234, 2016.

[57] Kyle Sargent, Kyle Hsu, Justin Johnson, Li Fei-Fei, and Jiajun Wu. Flow to the mode: Mode-seeking diffusion autoencoders for state-of-the-art image tokenization. *CoRR*, abs/2503.11056, 2025.

[58] Mei Song, Andrea Montanari, and P Nguyen. A mean field view of the landscape of two-layers neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

[59] Yuchuan Tian, Zhijun Tu, Hanting Chen, Jie Hu, Chao Xu, and Yunhe Wang. U-dits: Down-sample tokens in u-shaped diffusion transformers. In *NeurIPS*, 2024.

[60] Leena Chennuru Vankadara, Jin Xu, Moritz Haas, and Volkan Cevher. On feature learning in structured state space models. In *NeurIPS*, 2024.

[61] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[62] Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, Jianyuan Zeng, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.

[63] Rongzhen Wang, Yan Zhang, Chenyu Zheng, Chongxuan Li, and Guoqiang Wu. A theory for conditional generative modeling on multiple data sources. *arXiv preprint arXiv:2502.14583*, 2025.

[64] Mitchell Wortsman, Peter J. Liu, Lechao Xiao, Katie E. Everett, Alexander A. Alemi, Ben Adlam, John D. Co-Reyes, Izzeddin Gur, Abhishek Kumar, Roman Novak, Jeffrey Pennington, Jascha Sohl-Dickstein, Kelvin Xu, Jaehoon Lee, Justin Gilmer, and Simon Kornblith. Small-scale proxies for large-scale transformer training instabilities. In *ICLR*, 2024.

[65] Lechao Xiao. Rethinking conventional wisdom in machine learning: From generalization to scaling. *CoRR*, abs/2409.15156, 2024.

[66] Enze Xie, Junsong Chen, Junyu Chen, Han Cai, Haotian Tang, Yujun Lin, Zhekai Zhang, Muyang Li, Ligeng Zhu, Yao Lu, and Song Han. SANA: efficient high-resolution image synthesis with linear diffusion transformers. *CoRR*, abs/2410.10629, 2024.

[67] Greg Yang. Wide feedforward or recurrent neural networks of any architecture are gaussian processes. In *NeurIPS*, pages 9947–9960, 2019.

[68] Greg Yang. Tensor programs II: neural tangent kernel for any architecture. *CoRR*, abs/2006.14548, 2020.

[69] Greg Yang. Tensor programs III: neural matrix laws. *CoRR*, abs/2009.10685, 2020.

[70] Greg Yang and Edward J. Hu. Tensor programs IV: feature learning in infinite-width neural networks. In Marina Meila and Tong Zhang, editors, *ICML*, volume 139, pages 11727–11737. PMLR, 2021.

[71] Greg Yang and Etai Littwin. Tensor programs iib: Architectural universality of neural tangent kernel training dynamics. In *ICML*, volume 139, pages 11762–11772, 2021.

[72] Greg Yang and Etai Littwin. Tensor programs ivb: Adaptive optimization in the infinite-width limit. *CoRR*, abs/2308.01814, 2023.

[73] Greg Yang, Edward J. Hu, Igor Babuschkin, Szymon Sidor, Xiaodong Liu, David Farhi, Nick Ryder, Jakub Pachocki, Weizhu Chen, and Jianfeng Gao. Tensor programs V: tuning large neural networks via zero-shot hyperparameter transfer. *CoRR*, abs/2203.03466, 2022.

[74] Greg Yang, Dingli Yu, Chen Zhu, and Soufiane Hayou. Tensor programs VI: feature learning in infinite depth neural networks. In *ICLR*, 2024.

[75] Yuanyang Yin, Yaqi Zhao, Mingwu Zheng, Ke Lin, Jiarong Ou, Rui Chen, Victor Shea-Jay Huang, Jiahao Wang, Xin Tao, Pengfei Wan, Di Zhang, Baoqun Yin, Wentao Zhang, and Kun Gai. Towards precise scaling laws for video diffusion transformers. *CoRR*, abs/2411.17470, 2024.

[76] Sihyun Yu, Sangkyung Kwak, Huiwon Jang, Jongheon Jeong, Jonathan Huang, Jinwoo Shin, and Saining Xie. Representation alignment for generation: Training diffusion transformers is easier than you think. *CoRR*, abs/2410.06940, 2024.

[77] Min Zhao, Guande He, Yixiao Chen, Hongzhou Zhu, Chongxuan Li, and Jun Zhu. Riflex: A free lunch for length extrapolation in video diffusion transformers. *arXiv preprint arXiv:2502.15894*, 2025.

[78] Chenyu Zheng, Guoqiang Wu, Fan Bao, Yue Cao, Chongxuan Li, and Jun Zhu. Revisiting discriminative vs. generative classifiers: Theory and implications. In *International conference on machine learning*, pages 42420–42477. PMLR, 2023.

[79] Chenyu Zheng, Guoqiang Wu, and Chongxuan Li. Toward understanding generative data augmentation. *Advances in neural information processing systems*, 36:54046–54060, 2023.

[80] Chenyu Zheng, Wei Huang, Rongzhen Wang, Guoqiang Wu, Jun Zhu, and Chongxuan Li. On mesa-optimization in autoregressively trained transformers: Emergence and capability. *Advances in Neural Information Processing Systems*, 37:49081–49129, 2024.

[81] Zangwei Zheng, Xiangyu Peng, Tianji Yang, Chenhui Shen, Shenggui Li, Hongxin Liu, Yukun Zhou, Tianyi Li, and Yang You. Open-sora: Democratizing efficient video production for all. *arXiv preprint arXiv:2412.20404*, 2024.

[82] Yutao Zhu, Kun Zhou, Kelong Mao, Wentong Chen, Yiding Sun, Zhipeng Chen, Qian Cao, Yihan Wu, Yushuo Chen, Feng Wang, et al. Yulan: An open-source large language model. *arXiv preprint arXiv:2406.19853*, 2024.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: Our abstract and introduction accurately reflect the theoretical contribution in Section 3.1 and experimental contribution in Section 4 and 5.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: See details in Section 6.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

Justification: Assumption used to simplify the proof can be found in Assumption C.1, and the completed proof can be found in Appendix C.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: We introduce the necessary configurations in Section 4, Section 5 and Appendix E. We also open-source our code at *https://github.com/ML-GSAI/Scaling-Diffusion-Transformers-muP*.

Guidelines:

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

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

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

Answer: [Yes]

Justification: We also open-source our code at *https://github.com/ML-GSAI/Scaling-Diffusion-Transformers-muP* for DiT and PixArt-α experiments. However, the data for the MMDiT experiments is not available now because it is an internal dataset. We will try to release it in the future.

Guidelines:

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

6. **Experimental setting/details**

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

Answer: [Yes]

Justification: We introduce the necessary configurations in Section 4, Section 5 and Appendix E. We also open-source our code at *https://github.com/ML-GSAI/Scaling-Diffusion-Transformers-muP* for DiT and PixArt-α experiments.

Guidelines:

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

7. **Experiment statistical significance**

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

Answer: [No]

Justification: Error bars are not reported because it would be too computationally expensive. expensive

Guidelines:

- The answer NA means that the paper does not include experiments.
- The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

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

8. **Experiments compute resources**

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

Answer: [Yes]

Justification: See details in Appendix E.

Guidelines:

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

9. **Code of ethics**

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

Answer: [Yes]

Justification: We confirm the research conducted in the paper satisfies the NeurIPS Code of Ethics.

Guidelines:

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

10. **Broader impacts**

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

Answer: [Yes]

Justification: See details in Section 6.

Guidelines:

- The answer NA means that there is no societal impact of the work performed.

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

11. **Safeguards**

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

Answer: [NA]

Justification: This paper poses no such risks.

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: See details in Table 7 in Appendix E.

Guidelines:

- The answer NA means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.

- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

Answer: [No]

Justification: The internal dataset and 18B models are not available to the public now. We will try to release them in the future.

Guidelines:

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

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

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

Answer: [Yes]

Justification: See details in human evaluation in Appendix E.4.5.

Guidelines:

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

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

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

Answer: [NA]

Justification: Our human evaluation process is safe, and the generated images are legal.

Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.

- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

Answer: [NA]

Justification: We only use the LLMs to polish our writing.

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (`https://neurips.cc/Conferences/2025/LLM`) for what should or should not be described.

Contents of Appendix

Appendix A   Additional Related Work

A.1   Scaling Diffusion Transformers

HourglassDiT [12] introduces a hierarchical architecture with downsampling and upsampling in DiT, reducing computational complexity for high-resolution image generation. SD3 [16] scales MMDiT to 8B parameters by increasing the backbone depth. Large-DiT [18] incorporates LLaMA's text embeddings and scales the DiT backbone, demonstrating that scaling to 7B parameters improves convergence speed. DiT-MoE [17] further scales diffusion Transformers to 16.5B parameters using a mixture-of-experts architecture. Li et al. [40] empirically investigate the scaling behavior of various diffusion Transformers, finding that U-ViT [1] scales more efficiently than cross-attention-based DiT variants. Yin et al. [75] systematically analyze scaling laws for video diffusion Transformers, enabling the prediction of optimal hyperparameters (HPs) for any model size and compute budget. We note that all these works adopt standard parameterization (SP), which prevents the transfer of optimal HPs across different model sizes, and thus suffer from heavy tuning costs at scale.

A.2   Applications of μP in AIGC

Recently, μP has been successfully applied to the pretraining of large language models (LLMs) [73; 14; 28; 43; 23; 42; 82], reducing HP tuning costs and stabilizing training. For diffusion Transformers, in addition to AuraFlow v0.1 [10], some works have also employed μP. For example, Gulrajani and Hashimoto [23] searches for the base learning rate under μP and transfers it to a 1B-parameter diffusion language model, while Sargent et al. [57] uses μP to tune the HPs of a Transformer-based diffusion autoencoder, achieving state-of-the-art image tokenization across multiple compression rates. However, these studies differ fundamentally from ours. First, while they assume the HP transferability property of original μP (even though it may not hold in their setups), our work provides rigorous theoretical guarantees for the μP of diffusion Transformers and systematically verifies its HP transferability. Second, we validate the scalability of μP at up to 18B parameters, significantly larger than the models used in prior work.

Appendix B   Theoretical Background of μP

In this section, we review the theoretical foundations of μP. For simplicity, we mainly focus on the Adam/AdamW optimizer widely used in practice. Besides, the ϵ in the optimizer is assumed to be zero, which is very small in practice (e.g., 10^{-8}).

B.1 abc-Parameterization Setup

For completeness, we restate the abc-parameterization from Section 2, which underpins typical μP theory. The abc-parameterization specifies the multiplier, initial variance, and learning rate for each weight [70; 73]. Specifically, let n denote the network width, we parameterize each weight as $\boldsymbol{W} = \phi_W n^{-a_W} \widetilde{\boldsymbol{W}}$, where the trainable component $\widetilde{\boldsymbol{W}}$ is initialized as $\widetilde{W}_{ij} \sim \mathcal{N}(0, \sigma_W^2 n^{-2b_W})$. The learning rate for $\widetilde{\boldsymbol{W}}$ is parameterized as $\eta_W n^{-c_W}$. The width-independent quantities ϕ_W, σ_W, and η_W are referred to as base HPs and can be transferred across different widths.

B.2   NE⊗OR⊤ Program

μP is originally proposed by Tensor Programs [67; 68; 71; 69; 70; 72; 74], which is a theoretical series mainly developed to express and analyze the infinite-width limits of neural networks both at initialization and during training. The latest and most general version among them is the NE⊗OR⊤ Program [72], which is formed by inductively generating scalars and vectors, starting from initial matrices/vectors/scalars (typically the initial weights of the neural network) and applying defined operations, including vector averaging (Avg), matrix multiplication (MatMul), and nonlinear outer product transformations (OuterNonlin). We introduce these operators, where n is the model width tending to be infinite.

Table 6: μP for (diffusion) Transformers with Adam/AdamW optimizer. We use purple text to highlight the differences between μP and standard parameterization (SP) in practice (e.g., Kaiming initialization [25]), and gray text to indicate the SP settings. We do not contain the scalar weights in the main text because they do not exist in DiT.

	Input weights	Hidden weights	Output weights	Scalar weights
a_W	0	0	1 (0)	0
b_W	0	$1/2$	0 ($1/2$)	0
c_W	0	1 (0)	0	0

Avg We can choose a existing vector $\boldsymbol{x} \in \mathbb{R}^n$ and append to the program a scalar

$$\frac{1}{n} \sum_{\alpha=1}^n x_\alpha \in \mathbb{R}.$$

MatMul We can choose a matrix $\boldsymbol{W} \in \mathbb{R}^{n \times n}$ and vector $\boldsymbol{x} \in \mathbb{R}^n$ existing in the program, and append to the program a vector

$$\boldsymbol{W}\boldsymbol{x} \in \mathbb{R}^n \quad \text{or} \quad \boldsymbol{W}^\top \boldsymbol{x} \in \mathbb{R}^n.$$

OuterNonlin For any integer $k, l \geq 0$, we can aggregate k existing vectors and l existing scalars to $\boldsymbol{X} \in \mathbb{R}^{n \times k}$ and $\boldsymbol{c} \in \mathbb{R}^l$, respectively. With an integer $r \geq 0$ and a pseudo-Lipschitz function $\psi : \mathbb{R}^{k(r+1)+l} \to \mathbb{R}$ (e.g., SiLU, GeLU), we append to the program a vector

$$\boldsymbol{y} \in \mathbb{R}^n, \quad y_\alpha = \frac{1}{n^r} \sum_{\beta_1,\ldots,\beta_r=1}^n \psi(\boldsymbol{X}_{\alpha:}; \boldsymbol{X}_{\beta_1:}; \ldots; \boldsymbol{X}_{\beta_r:}; \boldsymbol{c}^\top),$$

where the $r + 1$ is called the *order* of ψ.

While the above operators do not directly instruct to transform scalars into scalars, previous work [72] has proved that this can be done by combining the above operators. This can be formed as follows.

Lemma B.1 (Scalars-to-scalars transformation is representable by the NE⊗OR⊤ Program, Lemma 2.6.2 in [72]). *If $\psi : \mathbb{R}^l \to \mathbb{R}$ is a pseudo-Lipschitz function and $\boldsymbol{c}$ are the collected scalars in a NE⊗OR⊤ program, then $\psi(\boldsymbol{c}) \in \mathbb{R}$ can be introduced as a new scalar in the program.*

B.3 μP of Any Representable Archtecture

The seminal work of Yang and Littwin [72] systematically studies the dynamics of common neural architectures trained with adaptive optimizers (e.g., Adam [32], AdamW [47]), where the forward pass of neural network can be expressed as a NE⊗OR⊤ program (e.g., MLPs, CNNs [36], RNNs [27], standard Transformers [61]). In this setting, Yang and Littwin [72] prove the existence of a unique, optimal scaling rule—Maximal Update Parametrization (μP)—under which features in each layer evolve independently of width and at maximal strength. Otherwise, feature evolution in some layers will either explode or vanish in the infinite-width limit, rendering large-scale pretraining ineffective.

Concretely, μP is implemented by analytically adjusting HPs with model width, typically involving the weight multiplier, initialization variance, and learning rate considered in abc-parameterization. The μP formulation of any architecture whose forward pass can be expressed as a NE⊗OR⊤ program follows the same scaling rules presented in Table 6 (equivalent to Table 1 in the main text). Specifically, it adjusts the abc-parameterization of each weight according to its type, where the weights are categorized into four types: input weights, hidden weights, output weights, and scalar weights. The definitions are as follows: input weights satisfy $d_{\text{in}} = \Theta(1)$ and $d_{\text{out}} = \Theta(n)$; hidden weights satisfy $d_{\text{in}} = \Theta(n)$ and $d_{\text{out}} = \Theta(n)$; output weights satisfy $d_{\text{in}} = \Theta(n)$ and $d_{\text{out}} = \Theta(1)$; and scalar weights satisfy $d_{\text{in}} = \Theta(1)$ and $d_{\text{out}} = \Theta(1)$, where $n, d_{\text{in}}, d_{\text{out}}$ denotes the model width, fan-in dimension and fan-out dimension, respectively. The μP result is formally summarized in the following lemma.

Lemma B.2 (μP of representable architecture, Definition 2.9.12 in [72]). *If the forward pass of an architecture can be represented by a NE⊗OR⊤ program, its μP formulation follows the scaling rules in Table 6 (equivalent to Table 1 in the main text).*

B.4 Extensions to Other HPs

The μP principle can be extended to other HPs, such as common optimization-related HPs (e.g., warm-up iteration). We refer the reader to [73; 72] for a comprehensive discussion. We mainly focus on the HPs that occur in the main text.

- Gradient clip: The clip value should be held constant with respect to width.
- Warm-up iteration: The warm-up iteration should be held constant with respect to width.
- Weights of different losses (e.g., REPA loss, noise schedule): The weights of different losses should be held constant with respect to width.
- Weight decay: For coupled weight decay (e.g., Adam [32]), the weight decay value can not be transferred. For decoupled weight decay (e.g., AdamW [47]), the weight decay value should be held constant with respect to width.

Appendix C   Proof of Theorem 3.1

Elementary notations. We use lightface (e.g., a, A), lowercase boldface (e.g., $\boldsymbol{a}$), and uppercase boldface letters (e.g., $\boldsymbol{A}$) to denote scalars, vectors, and matrices, respectively. For a vector $\boldsymbol{a}$, we denote its i-th element as a_i. For a matrix $\boldsymbol{A}$, we use $\boldsymbol{A}_{k:}$, $\boldsymbol{A}_{:k}$ and A_{ij} to denote its k-th row, k-th column and (i, j)-th element, respectively. We define $[n] = \{1, 2, \ldots, n\}$.

For the reader's convenience, we restate Theorem 3.1 as follows.

Theorem C.1. *The forward passes of mainstream diffusion Transformers (U-ViT [1], DiT [52], Pixart-α [8], and MMDiT [16]) can be represented within the $\text{NE}\otimes\text{OR}\top$ Program. Therefore, their μP matches the standard μP presented in Table 6 (equivalent to Table 1 in the main text).*

Following the standard practice in the literature [70; 72; 74; 24] to simplify the derivation, we adopt the following assumption in all of our proofs.

Assumption C.1. *We assume that constant dimensions (fixed during scaling the width of neural networks) are 1, which includes data dimension, label dimension, patch size, frequency embedding size of time embedding, dimension of attention head, and so on. We also assume the width of different layers is the same n and do not consider bias parameters here.*

In principle, the proof technique naturally extends to any finite fixed dimension (≥ 1) and to hidden layers with different widths (non-uniform infinite-width setting) [67; 72; 24]. We refer the readers to Section 2.9 in Yang and Littwin [72], which provides a detailed discussion of both these generalizations and shows how the tensor program framework naturally accommodates them.

C.1   Proof of DiT

Proof. By Lemma B.2, we can prove the theorem by proving the forward pass of DiT can be represented by the $\text{NE}\otimes\text{OR}\top$ Program.

The DiT architecture includes an embedder for the input latent $x \in \mathbb{R}$, an embedder for the diffusion timestep $t \in \mathbb{R}$, an embedder for the label $y \in \mathbb{R}$, Transformer blocks with adaLN, and a final layer with adaLN. In the following, we use the $\text{NE}\otimes\text{OR}\top$ Program to represent the forward computation of these modules in sequence.

Embedder for the input latent x**.** The embedder for the x is a one-layer CNN, which is known to be representable by the $\text{NE}\otimes\text{OR}\top$ Program (e.g., see Program 7 in [67] for general CNNs with pooling). We write the program for clarity here. In our case, the one-layer CNN's parameters can be denoted by $\boldsymbol{w}^{\text{CNN}} \in \mathbb{R}^n$, and the operation can be implemented by

$$x_\alpha^{\text{embed}} := \psi(w_\alpha^{\text{CNN}}; x), \quad (\texttt{OuterNonlin})$$

where $\psi(w_\alpha^{\text{CNN}}; x) = w_\alpha^{\text{CNN}} x$.

Positional embedding. Given a number of patches (1 in our simplified case), the positional embedding is initialized with the sine-cosine version and fixed during the training. We use the $\boldsymbol{x}^{\text{pos}}$ to denote the positional embedding, and adding it to the $\boldsymbol{x}^{\text{embed}}$ can be written as

$$x_\alpha^{\text{embed}} := \psi([\boldsymbol{x}^{\text{embed}}, \boldsymbol{x}^{\text{pos}}]_{\alpha:}), \quad (\texttt{OuterNonlin})$$

where $\psi([\boldsymbol{x}^{\mathrm{embed}}, \boldsymbol{x}^{\mathrm{pos}}]_{\alpha:}) = x_\alpha^{\mathrm{embed}} + x_\alpha^{\mathrm{pos}}$. Here, we reuse the notation of $\boldsymbol{x}^{\mathrm{embed}}$ of the final embedding for the input latent x without confusion.

Embedder for the diffusion timestep t. The embedder for the diffusion timestep t is a frequency embedding (one dimension here) followed by a two-layer MLP. First, for the frequency embedding, we can represent it as

$$t^{\mathrm{freq}} := \psi(t). \quad \text{(Lemma B.1)}$$

Second, the MLP is known to be representable by the $\mathrm{N_E \otimes OR \top}$ Program (e.g., Program 1 in [67]). We write it here for completeness. We use $\boldsymbol{w}^{(1)} \in \mathbb{R}^n$ and $\boldsymbol{W}^{(2)} \in \mathbb{R}^{n \times n}$ to denote the weights in the first layer and the second layer of the MLP, respectively. We can derive

$$
\begin{aligned}
t_\alpha^{(1)} &:= \psi_1(w_\alpha^{(1)}; t^{\mathrm{freq}}), & \text{(OuterNonlin)} \\
h_\alpha^{(1)} &:= \psi_2(t_\alpha^{(1)}), & \text{(OuterNonlin)} \\
\boldsymbol{t}^{\mathrm{embed}} &:= \boldsymbol{W}^{(2)} \boldsymbol{h}^{(1)}, & \text{(MatMul)}
\end{aligned}
$$

where $\psi_1(w_\alpha^{(1)}; t^{\mathrm{freq}}) = w_\alpha^{(1)} t^{\mathrm{freq}}$ and $\psi_2(t_\alpha^{(1)}) = \mathrm{SiLU}(t_\alpha^{(1)})$.

Embedder for the label y. Denoting the weights of embedding as $\boldsymbol{w}^{\mathrm{embed}} \in \mathbb{R}^n$, then its computation can be written directly as

$$y_\alpha^{\mathrm{embed}} := \psi(w_\alpha^{\mathrm{embed}}; y), \quad \text{(OuterNonlin)}$$

where $\psi(w_\alpha^{\mathrm{embed}}; y) = w_\alpha^{\mathrm{embed}} y$.

Merging the embeddings of t and y. In DiT, we sum the embeddings of t and y as the condition $\boldsymbol{c}$ for the following calculation, which can be written as

$$c_\alpha := \psi([\boldsymbol{t}^{\mathrm{embed}}, \boldsymbol{y}^{\mathrm{embed}}]_{\alpha:}), \quad \text{(OuterNonlin)}$$

where $\psi([\boldsymbol{t}^{\mathrm{embed}}, \boldsymbol{y}^{\mathrm{embed}}]_{\alpha:}) = t_\alpha^{\mathrm{embed}} + y_\alpha^{\mathrm{embed}}$.

Transformer block with adaLN. First, the adaLN block maps the condition $\boldsymbol{c}$ to the gate, shift, and scale values $(\boldsymbol{\theta}^{\mathrm{attn}}, \boldsymbol{\beta}^{\mathrm{attn}}, \boldsymbol{\gamma}^{\mathrm{attn}}, \boldsymbol{\theta}^{\mathrm{mlp}}, \boldsymbol{\beta}^{\mathrm{mlp}}, \boldsymbol{\gamma}^{\mathrm{mlp}})$ for attention layer and MLP layer in the Transformer block. We denote the parameters of the adaLN block by $(\boldsymbol{W}^{\theta-\mathrm{attn}}, \boldsymbol{W}^{\beta-\mathrm{attn}}, \boldsymbol{W}^{\gamma-\mathrm{attn}}, \boldsymbol{W}^{\theta-\mathrm{mlp}}, \boldsymbol{W}^{\beta-\mathrm{mlp}}, \boldsymbol{W}^{\gamma-\mathrm{mlp}})$, each of them is in $\mathbb{R}^{n \times n}$. Then the forward pass of adaLN block can be written as

$$
\begin{aligned}
\widetilde{c}_\alpha &:= \psi(c_\alpha), & \text{(OuterNonlin)} \\
\boldsymbol{\theta}^{\mathrm{attn}} &:= \boldsymbol{W}^{\theta-\mathrm{attn}} \widetilde{\boldsymbol{c}}, & \text{(MatMul)} \\
\boldsymbol{\beta}^{\mathrm{attn}} &:= \boldsymbol{W}^{\beta-\mathrm{attn}} \widetilde{\boldsymbol{c}}, & \text{(MatMul)} \\
\boldsymbol{\gamma}^{\mathrm{attn}} &:= \boldsymbol{W}^{\gamma-\mathrm{attn}} \widetilde{\boldsymbol{c}}, & \text{(MatMul)} \\
\boldsymbol{\theta}^{\mathrm{mlp}} &:= \boldsymbol{W}^{\theta-\mathrm{mlp}} \widetilde{\boldsymbol{c}}, & \text{(MatMul)} \\
\boldsymbol{\beta}^{\mathrm{mlp}} &:= \boldsymbol{W}^{\beta-\mathrm{mlp}} \widetilde{\boldsymbol{c}}, & \text{(MatMul)} \\
\boldsymbol{\gamma}^{\mathrm{mlp}} &:= \boldsymbol{W}^{\gamma-\mathrm{mlp}} \widetilde{\boldsymbol{c}}, & \text{(MatMul)}
\end{aligned}
$$

where ψ is the SiLU activation.

Now, we denote the input feature of the Transformer block as $\boldsymbol{x} \in \mathbb{R}^n$. A layer norm without learnable parameters first normalizes it, which is also known to be representable by the $\mathrm{N_E \otimes OR \top}$ Program (e.g., Program 9 in [67]). We rewrite it in our simplified case.

$$
\begin{aligned}
\mathbb{E}[\boldsymbol{x}] &:= \frac{1}{n} \sum_{\alpha=1}^n x_\alpha, & \text{(Avg)} \\
\bar{x}_\alpha^2 &:= \psi_1(x_\alpha; \mathbb{E}[\boldsymbol{x}]), & \text{(OuterNonlin)} \\
\mathbb{V}[\boldsymbol{x}] &:= \frac{1}{n} \sum_{\alpha=1}^n \bar{x}_\alpha^2, & \text{(Avg)} \\
x_\alpha^{\mathrm{norm}} &:= \psi_2(x_\alpha; [\mathbb{E}[\boldsymbol{x}], \mathbb{V}[\boldsymbol{x}]]), & \text{(OuterNonlin)}
\end{aligned}
$$

where $\psi_1(x_\alpha; \mathbb{E}[\boldsymbol{x}]) = (x_\alpha - \mathbb{E}[\boldsymbol{x}])^2$, $\psi_2(x_\alpha; [\mathbb{E}[\boldsymbol{x}], \mathbb{V}[\boldsymbol{x}]]) = (x_\alpha - \mathbb{E}[\boldsymbol{x}])/\sqrt{\mathbb{V}[\boldsymbol{x}] + \epsilon}$.

After that, the normalized feature interacts with the condition information via the output of adaLN ($\boldsymbol{\beta}^{\text{attn}}, \boldsymbol{\gamma}^{\text{attn}}$), which can be represented as follows.

$$\widetilde{x}_\alpha^{\text{norm}} := \psi([\boldsymbol{x}^{\text{norm}}, \boldsymbol{\beta}^{\text{attn}}, \boldsymbol{\gamma}^{\text{attn}}]_{\alpha:}), \quad (\texttt{OuterNonlin})$$

where $\psi([\boldsymbol{x}^{\text{norm}}, \boldsymbol{\beta}^{\text{attn}}, \boldsymbol{\gamma}^{\text{attn}}]_{\alpha:}) = \gamma_\alpha^{\text{attn}} x_\alpha^{\text{norm}} + \beta_\alpha^{\text{attn}}$.

Now, the processed feature $\widetilde{\boldsymbol{x}}^{\text{norm}}$ is put to the multi-head attention layer, which is known to be representable by the NE$\otimes$OR$\top$ Program (e.g., see Program 10 in [67]). We write it for completeness here. We denote $\boldsymbol{W}^K, \boldsymbol{W}^Q, \boldsymbol{W}^V \in \mathbb{R}^{n \times n}$ the attention layer's key, query, and value matrix, each row representing a head with dimension 1. The forward pass can be written as follows.

$$\boldsymbol{k} := \boldsymbol{W}^K \widetilde{\boldsymbol{x}}^{\text{norm}}, \qquad\qquad (\texttt{MatMul})$$
$$\boldsymbol{q} := \boldsymbol{W}^Q \widetilde{\boldsymbol{x}}^{\text{norm}}, \qquad\qquad (\texttt{MatMul})$$
$$\boldsymbol{v} := \boldsymbol{W}^V \widetilde{\boldsymbol{x}}^{\text{norm}}, \qquad\qquad (\texttt{MatMul})$$
$$x_\alpha^{\text{attn}} := \psi([\boldsymbol{k}, \boldsymbol{q}, \boldsymbol{v}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

where $\psi([\boldsymbol{k}, \boldsymbol{q}, \boldsymbol{v}]_{\alpha:}) = k_\alpha q_\alpha v_\alpha$.

Before discussing the MLP layer, we still have a residual connection and interaction from the gate value $\boldsymbol{\theta}^{\text{attn}}$, which can be represented by

$$h_\alpha := \psi([\boldsymbol{x}^{\text{attn}}, \boldsymbol{\theta}^{\text{attn}}, \boldsymbol{x}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

where $\psi([\boldsymbol{x}^{\text{attn}}, \boldsymbol{\theta}^{\text{attn}}, \boldsymbol{x}]_{\alpha:}) = x_\alpha + \theta_\alpha^{\text{attn}} x_\alpha^{\text{attn}}$.

The remaining part is very similar to the above, which includes the layer norm, interaction with the outputs of adaLN, the MLP layer, and the residual connection. We write them in sequence for clarity.

For the layer norm, we can write

$$\mathbb{E}[\boldsymbol{h}] := \frac{1}{n} \sum_{\alpha=1}^n h_\alpha, \qquad\qquad (\texttt{Avg})$$
$$\bar{h}_\alpha^2 := \psi_1(h_\alpha; \mathbb{E}[\boldsymbol{h}]), \qquad\qquad (\texttt{OuterNonlin})$$
$$\mathbb{V}[\boldsymbol{h}] := \frac{1}{n} \sum_{\alpha=1}^n \bar{h}_\alpha^2, \qquad\qquad (\texttt{Avg})$$
$$h_\alpha^{\text{norm}} := \psi_2(h_\alpha; [\mathbb{E}[\boldsymbol{h}], \mathbb{V}[\boldsymbol{h}]]), \qquad\qquad (\texttt{OuterNonlin})$$

where $\psi_1(h_\alpha; \mathbb{E}[\boldsymbol{h}]) = (h_\alpha - \mathbb{E}[\boldsymbol{h}])^2$, $\psi_2(h_\alpha; [\mathbb{E}[\boldsymbol{h}], \mathbb{V}[\boldsymbol{h}]]) = (h_\alpha - \mathbb{E}[\boldsymbol{h}])/\sqrt{\mathbb{V}[\boldsymbol{h}] + \epsilon}$.

For the interaction with scale and shift value from adaLN, we have

$$\widetilde{h}_\alpha^{\text{norm}} := \psi([\boldsymbol{h}^{\text{norm}}, \boldsymbol{\beta}^{\text{mlp}}, \boldsymbol{\gamma}^{\text{mlp}}]_{\alpha:}), \quad (\texttt{OuterNonlin})$$

where $\psi([\boldsymbol{h}^{\text{norm}}, \boldsymbol{\beta}^{\text{mlp}}, \boldsymbol{\gamma}^{\text{mlp}}]_{\alpha:}) := \gamma_\alpha^{\text{mlp}} h_\alpha^{\text{norm}} + \beta_\alpha^{\text{mlp}}$.

For the two-layer MLP in the Transformer block, we use $\boldsymbol{W}^{(1)}, \boldsymbol{W}^{(2)} \in \mathbb{R}^{n \times n}$ to denote the weights in its first layer and the second layer, respectively. We have

$$h_\alpha^{(1)} := \boldsymbol{W}^{(1)} \widetilde{\boldsymbol{h}}^{\text{norm}}, \qquad\qquad (\texttt{MatMul})$$
$$\widetilde{h}_\alpha^{(1)} := \psi(h_\alpha^{(1)}), \qquad\qquad (\texttt{OuterNonlin})$$
$$\boldsymbol{h}^{\text{mlp}} := \boldsymbol{W}^{(2)} \widetilde{\boldsymbol{h}}^{(1)}, \qquad\qquad (\texttt{MatMul})$$

where the ψ is the GeLU activation function.

For the final residual connection, it can be represented by

$$h_\alpha^{\text{Tblock}} := \psi([\boldsymbol{h}^{\text{mlp}}, \boldsymbol{\theta}^{\text{mlp}}, \boldsymbol{h}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

where $\psi([\boldsymbol{h}^{\mathrm{mlp}}, \boldsymbol{\theta}^{\mathrm{mlp}}, \boldsymbol{h}]_{\alpha:}) = h_\alpha + \theta_\alpha^{\mathrm{mlp}} h_\alpha^{\mathrm{mlp}}$.

Final layer with adaLN. The final layer includes the adaLN, layer norm, and a linear projection. We discuss them in sequence.

The adaLN block has parameters $(\boldsymbol{W}^{\beta-\mathrm{final}}, \boldsymbol{W}^{\gamma-\mathrm{final}})$ and receive the condition $\boldsymbol{c}$. Its forward pass can be written as follows.

$$\widetilde{c}_\alpha := \psi(c_\alpha), \qquad\qquad\qquad (\texttt{OuterNonlin})$$

$$\boldsymbol{\beta}^{\mathrm{final}} := \boldsymbol{W}^{\beta-\mathrm{final}} \widetilde{\boldsymbol{c}}, \qquad\qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{\gamma}^{\mathrm{final}} := \boldsymbol{W}^{\gamma-\mathrm{final}} \widetilde{\boldsymbol{c}}, \qquad\qquad\qquad (\texttt{MatMul})$$

where ψ is the SiLU activation.

The layer norm layer normalizes the output of the transformer block $\boldsymbol{h}^{\mathrm{Tblock}}$, which is the same as the above layer norm. We omit the derivation here for simplicity. We denote the output of the layer norm by $\boldsymbol{z}^{\mathrm{norm}} \in \mathbb{R}^n$ and the parameter of the linear projection by $\boldsymbol{w}^{\mathrm{final}}$, then the remains calculation can be derived as

$$\widetilde{z}_\alpha^{\mathrm{norm}} := \psi_1([\boldsymbol{z}^{\mathrm{norm}}, \boldsymbol{\beta}^{\mathrm{final}}, \boldsymbol{\gamma}^{\mathrm{final}}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

$$\widetilde{z}_\alpha^{\mathrm{final}} := \psi_2([\boldsymbol{w}^{\mathrm{final}}, \widetilde{\boldsymbol{z}}^{\mathrm{norm}}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

$$z^{\mathrm{final}} = \frac{1}{n} \sum_{\alpha=1}^n \widetilde{z}_\alpha^{\mathrm{final}}, \qquad\qquad\qquad (\texttt{Avg})$$

where $\psi_1([\boldsymbol{z}^{\mathrm{norm}}, \boldsymbol{\beta}^{\mathrm{final}}, \boldsymbol{\gamma}^{\mathrm{final}}]_{\alpha:}) := \gamma_\alpha^{\mathrm{final}} z_\alpha^{\mathrm{norm}} + \beta_\alpha^{\mathrm{final}}$ and $\psi_2([\boldsymbol{w}^{\mathrm{final}}, \widetilde{\boldsymbol{z}}^{\mathrm{norm}}]_{\alpha:}) = w_\alpha^{\mathrm{final}} \widetilde{z}_\alpha^{\mathrm{norm}}$. Therefore, the forward pass from the inputs (x, y, t) to output z^{final} can be represented by the $\mathrm{NE}{\otimes}\mathrm{OR}{\top}$ Program, which means that The μP of DiT is the same as the standard μP in Table 6. The proof is completed.

$\square$

C.2 Proof of PixArt-α

Proof. Since most parts of the PixArt-α are the same as those in DiT, we only discuss the different modules from DiT for simplicity. These modules are mainly induced from the condition process.

Embedder for the diffusion timestep t. The embedder for the diffusion timestep t not only outputs $\boldsymbol{t}^{\mathrm{embed}}$ but also gate, shift, and scale values based on $\boldsymbol{t}^{\mathrm{embed}}$ (named adaLN-single in [8]), which are shared by the following Transformer block. We denote the additional parameters by $(\boldsymbol{W}^{\theta-\mathrm{tattn}}, \boldsymbol{W}^{\beta-\mathrm{tattn}}, \boldsymbol{W}^{\gamma-\mathrm{tattn}}, \boldsymbol{W}^{\theta-\mathrm{tmlp}}, \boldsymbol{W}^{\beta-\mathrm{tmlp}}, \boldsymbol{W}^{\gamma-\mathrm{tmlp}})$, each of them is in $\mathbb{R}^{n\times n}$. Then its forward pass can be written as

$$\widetilde{t}_\alpha := \psi(t_\alpha^{\mathrm{embed}}), \qquad\qquad\qquad (\texttt{OuterNonlin})$$

$$\boldsymbol{\theta}^{\mathrm{tattn}} := \boldsymbol{W}^{\theta-\mathrm{tattn}} \widetilde{\boldsymbol{t}}, \qquad\qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{\beta}^{\mathrm{tattn}} := \boldsymbol{W}^{\beta-\mathrm{tattn}} \widetilde{\boldsymbol{t}}, \qquad\qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{\gamma}^{\mathrm{tattn}} := \boldsymbol{W}^{\gamma-\mathrm{tattn}} \widetilde{\boldsymbol{t}}, \qquad\qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{\theta}^{\mathrm{tmlp}} := \boldsymbol{W}^{\theta-\mathrm{tmlp}} \widetilde{\boldsymbol{t}}, \qquad\qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{\beta}^{\mathrm{tmlp}} := \boldsymbol{W}^{\beta-\mathrm{tmlp}} \widetilde{\boldsymbol{t}}, \qquad\qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{\gamma}^{\mathrm{tmlp}} := \boldsymbol{W}^{\gamma-\mathrm{tmlp}} \widetilde{\boldsymbol{t}}, \qquad\qquad\qquad (\texttt{MatMul})$$

where ψ is the SiLU activation.

Transformer block with gate-shift-scale table. PixArt-α replaces the adaLN block in DiT block with the gate-shift-scale table, which parameters are denoted by $\boldsymbol{\theta}^{\mathrm{xattn}}, \boldsymbol{\beta}^{\mathrm{xattn}}, \boldsymbol{\gamma}^{\mathrm{xattn}}, \boldsymbol{\theta}^{\mathrm{xmlp}}, \boldsymbol{\beta}^{\mathrm{xmlp}}, \boldsymbol{\gamma}^{\mathrm{xmlp}} \in \mathbb{R}^n$. The gate-shift-scale table interacts with the shared embeddings of t in the following way.

$$\theta_\alpha^{\mathrm{attn}} := \psi([\boldsymbol{\theta}^{\mathrm{xattn}}, \boldsymbol{\theta}^{\mathrm{tattn}}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

$$\beta_\alpha^{\mathrm{attn}} := \psi([\boldsymbol{\beta}^{\mathrm{xattn}}, \boldsymbol{\beta}^{\mathrm{tattn}}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

$$\gamma_\alpha^{\text{attn}} := \psi([\boldsymbol{\gamma}^{\text{xattn}}, \boldsymbol{\gamma}^{\text{tattn}}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

$$\theta_\alpha^{\text{mlp}} := \psi([\boldsymbol{\theta}^{\text{xmlp}}, \boldsymbol{\theta}^{\text{tmlp}}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

$$\beta_\alpha^{\text{mlp}} := \psi([\boldsymbol{\beta}^{\text{xmlp}}, \boldsymbol{\beta}^{\text{tmlp}}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

$$\gamma_\alpha^{\text{mlp}} := \psi([\boldsymbol{\gamma}^{\text{xmlp}}, \boldsymbol{\gamma}^{\text{tmlp}}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

where $\psi(a, b) = a + b$. The above outputted gate, shift, and scale values are used in the following attention layer and MLP layer, just like DiT.

One more difference between the Transformer block of PixArt-α and DiT is the existence of the additional cross-attention layer, which integrates the output of the self-attention layer $\boldsymbol{h}$ and the text embedding $\boldsymbol{y}^{\text{embed}}$. We denote its parameters by $\boldsymbol{W}^{cK}, \boldsymbol{W}^{cQ}, \boldsymbol{W}^{cV}, \boldsymbol{W}^{\text{proj}} \in \mathbb{R}^{n \times n}$ It can be written as follows.

$$\boldsymbol{k} := \boldsymbol{W}^{cK} \boldsymbol{y}^{\text{embed}}, \qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{q} := \boldsymbol{W}^{cQ} \boldsymbol{h}, \qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{v} := \boldsymbol{W}^{cV} \boldsymbol{y}^{\text{embed}}, \qquad\qquad (\texttt{MatMul})$$

$$h_\alpha^v := \psi([\boldsymbol{k}, \boldsymbol{q}, \boldsymbol{v}]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

$$\boldsymbol{h}^{\text{cross}} := \boldsymbol{W}^{\text{proj}} \boldsymbol{h}^v, \qquad\qquad (\texttt{MatMul})$$

where $\psi([\boldsymbol{k}, \boldsymbol{q}, \boldsymbol{v}]_{\alpha:}) = k_\alpha q_\alpha v_\alpha$. The following pre-LN MLP with gate-shift-scale table is a trivial extension of the pre-LN MLP layer in the DiT block, like what we did above, so we omit them here.

Final layer with shift-scale table. The only difference is that we replace the adaLN block with a shift-scale table, just like what we did for the attention layer, so we omit the derivation here.

$\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\square$

C.3 Proof of U-ViT

Proof. We only discuss the modules that do not occur in DiT and PixArt-α. We find that the only new operator in U-ViT is the long skip connection, which concatenates the current main branch $\boldsymbol{h}^m \in \mathbb{R}^n$ and the long skip branch $\boldsymbol{h}^s \in \mathbb{R}^n$, and then performs a linear projection to reduce the dimension. We denote the parameters of the linear layer by $\boldsymbol{W} = [\boldsymbol{W}^m, \boldsymbol{W}^s]$, where $\boldsymbol{W}^m, \boldsymbol{W}^s \in \mathbb{R}^{n \times n}$, then the long skip connection can be represented by

$$\widetilde{\boldsymbol{h}}^m = \boldsymbol{W}^m \boldsymbol{h}^m, \qquad\qquad (\texttt{MatMul})$$

$$\widetilde{\boldsymbol{h}}^s = \boldsymbol{W}^s \boldsymbol{h}^s, \qquad\qquad (\texttt{MatMul})$$

$$h_\alpha = \psi([\widetilde{\boldsymbol{h}}^m, \widetilde{\boldsymbol{h}}^s]_{\alpha:}), \qquad\qquad (\texttt{OuterNonlin})$$

where $\psi([\widetilde{\boldsymbol{h}}^m, \widetilde{\boldsymbol{h}}^s]_{\alpha:}) = \widetilde{h}_\alpha^m + \widetilde{h}_\alpha^s$. Because other operators in U-ViT (CNN embeddings/decoder, layer norm, attention layer, MLP layer, residual connection) have been studied in DiT, we finish the proof. $\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\quad\square$

C.4 Proof of MMDiT

Proof. For simplicity, we only discuss the modules that do not occur in the above models. The new module that emerges in the MMDiT is the joint attention with the QK-Norm for the latent feature $\boldsymbol{x}$ and text condition $\boldsymbol{c}$. For the latent feature, we denote the key/query/value parameters by $\boldsymbol{W}^{Kx}, \boldsymbol{W}^{Qx}, \boldsymbol{W}^{Vx} \in \mathbb{R}^{n \times n}$, the parameters of QK-norm by $\boldsymbol{\gamma}^{Kx}, \boldsymbol{\gamma}^{Qx} \in \mathbb{R}^n$, and the parameters of linear projection by $\boldsymbol{W}^{Ox} \in \mathbb{R}^{n \times n}$. We can define the similar parameters for the text condition $\boldsymbol{c}$ as $\boldsymbol{W}^{Kc}, \boldsymbol{W}^{Qc}, \boldsymbol{W}^{Vc}, \boldsymbol{\gamma}^{Kc}, \boldsymbol{\gamma}^{Qc}, \boldsymbol{W}^{Oc}$. The forward pass of the joint attention layer can be written as follows.

$$\boldsymbol{k}^x := \boldsymbol{W}^{Kx} \boldsymbol{x}, \qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{q}^x := \boldsymbol{W}^{Qx} \boldsymbol{x}, \qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{v}^x := \boldsymbol{W}^{Vx} \boldsymbol{x}, \qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{k}^c := \boldsymbol{W}^{Kc} \boldsymbol{c}, \qquad\qquad (\texttt{MatMul})$$

$$\boldsymbol{q}^c := \boldsymbol{W}^{Qc}\boldsymbol{c}, \qquad\qquad\qquad\qquad\qquad \text{(MatMul)}$$

$$\boldsymbol{v}^c := \boldsymbol{W}^{Vc}\boldsymbol{c}, \qquad\qquad\qquad\qquad\qquad \text{(MatMul)}$$

$$k_\alpha^{x\mathrm{norm}} := \psi_1([\boldsymbol{k}^x, \boldsymbol{\gamma}^{Kx}]_{\alpha:}), \qquad\qquad\qquad \text{(OuterNonlin)}$$

$$q_\alpha^{x\mathrm{norm}} := \psi_1([\boldsymbol{q}^x, \boldsymbol{\gamma}^{Qx}]_{\alpha:}), \qquad\qquad\qquad \text{(OuterNonlin)}$$

$$k_\alpha^{c\mathrm{norm}} := \psi_1([\boldsymbol{k}^c, \boldsymbol{\gamma}^{Kc}]_{\alpha:}), \qquad\qquad\qquad \text{(OuterNonlin)}$$

$$q_\alpha^{c\mathrm{norm}} := \psi_1([\boldsymbol{q}^c, \boldsymbol{\gamma}^{Qc}]_{\alpha:}), \qquad\qquad\qquad \text{(OuterNonlin)}$$

$$\widetilde{\boldsymbol{v}}^x := \psi_2([\boldsymbol{q}^{x\mathrm{norm}}, \boldsymbol{k}^{x\mathrm{norm}}, \boldsymbol{v}^x, \boldsymbol{k}^{c\mathrm{norm}}, \boldsymbol{v}^c]_{\alpha:}), \qquad \text{(OuterNonlin)}$$

$$\widetilde{\boldsymbol{v}}^c := \psi_2([\boldsymbol{q}^{c\mathrm{norm}}, \boldsymbol{k}^{x\mathrm{norm}}, \boldsymbol{v}^x, \boldsymbol{k}^{c\mathrm{norm}}, \boldsymbol{v}^c]_{\alpha:}), \qquad \text{(OuterNonlin)}$$

$$\boldsymbol{v}^{x\mathrm{out}} := \boldsymbol{W}^{Ox}\widetilde{\boldsymbol{v}}^x, \qquad\qquad\qquad\qquad \text{(MatMul)}$$

$$\boldsymbol{v}^{c\mathrm{out}} := \boldsymbol{W}^{Oc}\widetilde{\boldsymbol{v}}^c, \qquad\qquad\qquad\qquad \text{(MatMul)}$$

where $\psi_1(a,b) = ab/|a|$ and $\psi_2(a,b,c,d,e) = abc + ade$. The proof is completed. $\qquad\square$

Appendix D Additional Details for Section 3

In this section, we summarize the methodology for verifying base HP transferability across widths, batch sizes, and training steps in Algorithms 1, 2, and 3, respectively. We also present the μTransfer algorithm [73] in Algorithm 4 for the completeness.

Algorithm 1 Validate the base HP Transferability of μP across widths

---

1: **Input:** set of widths $\{n_i\}_{i=1}^P$, set of base HP to validate $\{\lambda_j\}_{j=1}^R$, other fixed base HPs $\boldsymbol{\theta}$, batch size B, training iteration T, base width n_{base}
2: **Output:** whether model with μP enjoy base HP transferability across widths
3: **for** $i = 1$ **to** P **do**
4:      **for** $j = 1$ **to** R **do**
5:          $\lambda_{ij} \leftarrow \mu\mathrm{P}(\lambda_j, n_{\mathrm{base}}, n_i)$ $\triangleright$ actual HP for n_i-width model
6:          $\boldsymbol{\theta}_i \leftarrow \mu\mathrm{P}(\boldsymbol{\theta}, n_{\mathrm{base}}, n_i)$ $\triangleright$ actual HP for n_i-width model
7:          $\mathcal{M}_{ij} \leftarrow$ train n_i-width model with b.s. B, iter. T, HPs $\lambda_{ij}, \boldsymbol{\theta}_i$
8:          $s_{ij} \leftarrow$ evaluate score for $\mathcal{M}_{ij}$
9:      **end for**
10:    $\lambda_i^* \leftarrow \arg\max_{\lambda_j}(\{s_{ij}\}_{j=1}^R)$ $\triangleright$ Obtain the optimal base HP for n_i-width model
11: **end for**
12: **return** whether λ_i^*s are approximately same

Algorithm 2 Validate the base HP Transferability of μP across batch sizes

---

1: **Input:** set of batch sizes $\{B_i\}_{i=1}^P$, set of base HP to validate $\{\lambda_j\}_{j=1}^R$, other fixed base HPs $\boldsymbol{\theta}$, width n, training iteration T, base width n_{base}
2: **Output:** whether model with μP enjoy base HP transferability across batch sizes
3: **for** $i = 1$ **to** P **do**
4:      **for** $j = 1$ **to** R **do**
5:          $\lambda_{ij} \leftarrow \mu\mathrm{P}(\lambda_j, n_{\mathrm{base}}, n)$ $\triangleright$ actual HP for n-width model
6:          $\boldsymbol{\theta}_i \leftarrow \mu\mathrm{P}(\boldsymbol{\theta}, n_{\mathrm{base}}, n)$ $\triangleright$ actual HP for n-width model
7:          $\mathcal{M}_{ij} \leftarrow$ train n-width model with b.s. B_i, iter. T, HPs $\lambda_{ij}, \boldsymbol{\theta}_i$
8:          $s_{ij} \leftarrow$ evaluate score for $\mathcal{M}_{ij}$
9:      **end for**
10:    $\lambda_i^* \leftarrow \arg\max_{\lambda_j}(\{s_{ij}\}_{j=1}^R)$ $\triangleright$ Obtain the optimal base HP for n-width model
11: **end for**
12: **return** whether λ_i^*s are approximately same

Algorithm 3 Validate the base HP Transferability of μP across training steps

---

1: **Input:** set of training steps $\{T_i\}_{i=1}^P$, set of base HP to validate $\{\lambda_j\}_{j=1}^R$, other fixed base HPs $\boldsymbol{\theta}$, batch size B, width n, base width n_{base}
2: **Output:** whether model with μP enjoy base HP transferability across training steps
3: **for** $i = 1$ **to** P **do**
4:     **for** $j = 1$ **to** R **do**
5:         $\lambda_{ij} \leftarrow \mu\text{P}(\lambda_j, n_{\text{base}}, n)$ $\triangleright$ actual HP for n-width model
6:         $\boldsymbol{\theta}_i \leftarrow \mu\text{P}(\boldsymbol{\theta}, n_{\text{base}}, n)$ $\triangleright$ actual HP for n-width model
7:         $\mathcal{M}_{ij} \leftarrow$ train n-width model with b.s. B, iter. T_i, HPs $\lambda_{ij}, \boldsymbol{\theta}_i$
8:         $s_{ij} \leftarrow$ evaluate score for $\mathcal{M}_{ij}$
9:     **end for**
10:    $\lambda_i^* \leftarrow \arg\max_{\lambda_j}(\{s_{ij}\}_{j=1}^R)$ $\triangleright$ Obtain the optimal base HP for n-width model
11: **end for**
12: **return** whether λ_i^*s are approximately same

Algorithm 4 μTransfer (Algorithm 1 in [73])

1: **Input:** base/proxy/target width $n_{\text{base}}/n_{\text{proxy}}/n_{\text{target}}$, proxy/target batch size $B_{\text{proxy}}/B_{\text{target}}$, proxy/target training steps $T_{\text{proxy}}/T_{\text{target}}$, set of base HPs to search $\{\boldsymbol{\theta}_i\}_{j=1}^R$
2: **Output:** Trained target model $\mathcal{M}_{\text{target}}$
3: **for** $j = 1$ **to** R **do**
4:    $\boldsymbol{\theta}_{\text{proxy},j} \leftarrow \mu\text{P}(\boldsymbol{\theta}_j, n_{\text{base}}, n_{\text{proxy}})$ $\triangleright$ actual HPs of proxy model
5:    $\mathcal{M}_j \leftarrow$ train n_{proxy}-width model with batch size B_{proxy}, training steps T_{proxy}, HPs $\boldsymbol{\theta}_{\text{proxy},j}$
6:    $s_j \leftarrow$ evaluate score for $\mathcal{M}_j$
7: **end for**
8: $\boldsymbol{\theta}^* \leftarrow \arg\max_{\boldsymbol{\theta}_j}(\{s_j\}_{j=1}^R)$
9: $\boldsymbol{\theta}_{\text{target}} \leftarrow \mu\text{P}(\boldsymbol{\theta}^*, n_{\text{base}}, n_{\text{target}})$ $\triangleright$ actual HPs of target model
10: $\mathcal{M}_{\text{target}} \leftarrow$ train n_{target}-width model with batch size B_{target}, training steps T_{target}, HPs $\boldsymbol{\theta}_{\text{target}}$
11: **return** $\mathcal{M}_{\text{target}}$

Appendix E   Additional Experimental Details and Results

In this section, we present the additional experimental details and results which are neglected by the main text.

E.1   Assets and Licenses

All used assets (datasets and codes) and their licenses are listed in Table 7.

Table 7: **The used assets and licenses.**

URL	Citation	License
https://www.image-net.org/	[13]	non-commercial
https://ai.meta.com/datasets/segment-anything/	[33]	Apache-2.0
http://images.cocodataset.org/zips/val2014.zip	[45]	Commons Attribution 4.0
https://huggingface.co/datasets/playgroundai/MJHQ-30K	[39]	-
https://github.com/djghosh13/geneval	[21]	MIT
https://github.com/facebookresearch/DiT	[52]	Link
https://github.com/openai/guided-diffusion	[15]	MIT
https://github.com/PixArt-alpha/PixArt-alpha	[8]	Apache-2.0
https://github.com/microsoft/mup	[73]	MIT

E.2   Additional Details of DiT Experiments

E.2.1   Evaluation Implementation

Following the Table 4 in original DiT paper [52], we use the codebase of ADM [15] to obtain the results of FID, sFID, IS, precision, and recall without cfg.

E.2.2   Additional Results of Base HP transferability

Base learning rate.   Comprehensive results of base learning rate transferability across widths, batch sizes, and training steps are presented in Tables 8, 9, and 10, respectively.

Table 8: **DiT-μP enjoys base learning rate transferability across widths.** We use a batch size of 256 and 200K training iterations. NaN data points indicate training instability, where the loss explodes. Under μP, the base learning rate can be transferred across model widths.

Width	$\log_2(\text{lr})$	FID-50K ↓	sFID-50K ↓	Inception Score ↑	Precision ↑	Recall ↑
144	-9	NaN	NaN	NaN	NaN	NaN
144	-10	**88.89**	**16.19**	**14.24**	**0.30**	**0.437**
144	-11	91.88	16.79	13.73	0.29	0.431
144	-12	93.61	17.94	13.41	0.28	0.406
144	-13	102.99	21.26	11.93	0.25	0.366
288	-9	NaN	NaN	NaN	NaN	NaN
288	-10	**61.65**	**9.06**	**20.60**	**0.41**	**0.563**
288	-11	63.85	10.32	20.59	0.40	0.561
288	-12	65.99	10.79	19.73	0.38	0.544
288	-13	79.17	13.04	15.92	0.33	0.503
576	-9	NaN	NaN	NaN	NaN	NaN
576	-10	**43.73**	**6.65**	**28.82**	**0.51**	**0.611**
576	-11	45.00	7.22	28.61	0.50	0.606
576	-12	50.30	7.87	25.67	0.47	0.602
576	-13	66.20	9.67	18.91	0.39	0.562

Base output multiplier.   Likewise that for the base learning rate, we also sweep the base multiplier of the output weight over the set $\{2^{-6}, 2^{-4}, 2^{-2}, 2^{0}, 2^{2}, 2^{4}, 2^{6}\}$ across various widths, batch sizes and training steps. FID-50K results are shown in Figure 7, and comprehensive results for other metrics are presented in Tables 11, 12, and 13, respectively. As presented in Figure 7, the optimal base output multiplier is approximately transferable across different widths, batch sizes, and training iterations.

(a) HP transfer across widths.   (b) HP transfer across batch sizes.   (c) HP transfer across iterations.
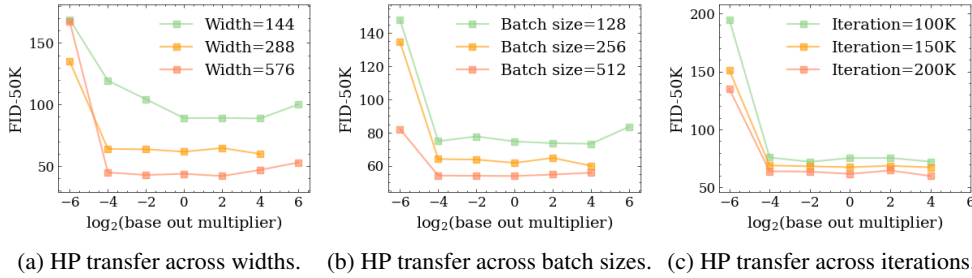
Figure 7: **DiT-μP enjoys base HP transferability**. Unless otherwise specified, we use a model width of 288, a batch size of 256, and a training iteration of 200K. Missing data points indicate training instability. Under μP, the base output multiplier can be transferred across model widths, batch sizes, and steps.

Table 9: **DiT-μP enjoys base learning rate transferability across batch sizes.** We use a width of 288 and a training iteration of 200K. NaN data points indicate training instability, where the loss explodes. We **bold** the best result and underline the second-best result. Under μP, the base learning rate can be (approximately) transferred across batch sizes.

Batch size	$\log_2(\text{lr})$	FID-50K ↓	sFID-50K ↓	Inception Score ↑	Precision ↑	Recall ↑
128	-9	NaN	NaN	NaN	NaN	NaN
128	-10	<u>74.54</u>	**10.32**	<u>16.70</u>	<u>0.358</u>	<u>0.513</u>
128	-11	**73.47**	10.33	**16.83**	**0.366**	**0.523**
128	-12	76.98	12.12	16.50	0.343	0.498
128	-13	88.79	14.63	14.10	0.293	0.453
256	-9	NaN	NaN	NaN	NaN	NaN
256	-10	**61.65**	**9.06**	**20.60**	**0.407**	**0.563**
256	-11	<u>63.85</u>	<u>10.32</u>	<u>20.59</u>	<u>0.398</u>	<u>0.561</u>
256	-12	65.99	10.79	19.73	0.383	0.544
256	-13	79.17	13.04	15.92	0.327	0.503
512	-9	<u>53.90</u>	**8.52**	24.18	**0.452**	0.585
512	-10	**53.72**	<u>9.27</u>	<u>24.66</u>	<u>0.448</u>	**0.592**
512	-11	54.36	9.33	**24.96**	0.442	<u>0.587</u>
512	-12	57.60	10.13	23.37	0.420	0.577
512	-13	72.09	12.15	17.81	0.356	0.537

Table 10: **DiT-μP enjoys base learning rate transferability across training steps.** We use a width of 288 and a batch size of 256. NaN data points indicate training instability, where the loss explodes. Under μP, the base learning rate can be transferred across training steps.

Iteration	$\log_2(\text{lr})$	FID-50K ↓	sFID-50K ↓	Inception Score ↑	Precision ↑	Recall ↑
100K	-9	**74.40**	11.37	**16.75**	0.363	0.504
100K	-10	75.35	**10.67**	16.61	**0.363**	**0.514**
100K	-11	78.57	11.69	16.17	0.338	0.495
100K	-12	84.74	12.97	14.78	0.315	0.474
100K	-13	102.79	18.37	11.74	0.243	0.363
150K	-9	70.44	10.07	18.03	**0.396**	0.534
150K	-10	**67.48**	**9.45**	**18.83**	0.387	**0.553**
150K	-11	69.61	10.51	18.45	0.380	0.533
150K	-12	72.72	11.33	17.52	0.357	0.515
150K	-13	88.05	14.07	14.34	0.296	0.455
200K	-9	NaN	NaN	NaN	NaN	NaN
200K	-10	**61.65**	**9.06**	**20.60**	**0.407**	**0.563**
200K	-11	63.85	10.32	20.59	0.398	0.561
200K	-12	65.99	10.79	19.73	0.383	0.544
200K	-13	79.17	13.04	15.92	0.327	0.503

Table 11: **DiT-μP enjoys base output multiplier transferability across widths.** We use a batch size of 256 and a training iteration of 200K. NaN data points indicate training instability, where the loss explodes. Under μP, the base output multiplier can be (approximately) transferred across model widths.

Width	$\log_2(\phi_{\text{out}})$	FID-50K $\downarrow$	sFID-50K $\downarrow$	Inception Score $\uparrow$	Precision $\uparrow$	Recall $\uparrow$
144	-6	168.81	62.70	6.30	0.157	0.061
144	-4	119.38	27.05	10.77	0.185	0.300
144	-2	104.26	15.65	12.17	0.262	0.419
144	0	88.89	16.19	14.24	0.297	0.437
144	2	89.01	16.83	13.97	0.294	0.433
144	4	88.68	15.71	14.19	0.303	0.441
144	6	100.03	19.33	12.48	0.262	0.386
288	-6	134.82	41.52	9.44	0.184	0.186
288	-4	63.93	8.98	20.31	0.395	0.552
288	-2	63.65	9.15	20.23	0.401	0.563
288	0	61.65	9.06	20.60	0.407	0.563
288	2	64.64	9.44	19.30	0.398	0.555
288	4	59.85	9.39	20.95	0.415	0.566
288	6	NaN	NaN	NaN	NaN	NaN
576	-6	167.14	69.32	5.04	0.130	0.051
576	-4	44.82	6.66	29.47	0.510	0.592
576	-2	42.75	6.68	30.73	0.526	0.600
576	0	43.73	6.65	28.82	0.509	0.611
576	2	41.93	6.68	29.45	0.537	0.600
576	4	46.87	6.74	26.78	0.501	0.594
576	6	52.93	7.30	24.07	0.455	0.584

Table 12: **DiT-μP enjoys base output multiplier transferability across batch sizes.** We use a width of 288 and a training iteration of 200K. NaN data points indicate training instability, where the loss explodes. Under μP, the base output multiplier can be (approximately) transferred across batch sizes.

Batch Size	$\log_2(\phi_{\text{out}})$	FID-50K ↓	sFID-50K ↓	Inception Score ↑	Precision ↑	Recall ↑
128	-6	148.06	51.95	7.76	0.170	0.126
128	-4	74.74	10.21	17.23	0.357	0.513
128	-2	77.62	11.55	16.20	0.346	0.511
128	0	74.54	10.32	16.70	0.358	0.513
128	2	73.55	10.24	16.68	0.365	0.514
128	4	73.19	9.93	16.93	0.375	0.514
128	6	83.48	11.79	14.92	0.323	0.464
256	-6	134.82	41.52	9.44	0.184	0.186
256	-4	63.93	8.98	20.31	0.395	0.552
256	-2	63.65	9.15	20.23	0.401	0.563
256	0	61.65	9.06	20.60	0.407	0.563
256	2	64.64	9.44	19.30	0.398	0.555
256	4	59.85	9.39	20.95	0.415	0.566
256	6	NaN	NaN	NaN	NaN	NaN
512	-6	82.00	14.83	16.13	0.325	0.468
512	-4	53.99	8.83	24.49	0.447	0.585
512	-2	53.85	9.11	24.88	0.443	0.584
512	0	53.72	9.27	24.66	0.448	0.592
512	2	54.62	9.14	23.97	0.449	0.593
512	4	55.68	8.49	23.29	0.441	0.597
512	6	NaN	NaN	NaN	NaN	NaN

Table 13: **DiT-μP enjoys base output multiplier transferability across training steps.** We use a width of 288 and a batch size of 256. NaN data points indicate training instability, where the loss explodes. Under μP, the base output multiplier can be (approximately) transferred across training steps.

Iteration	$\log_2(\phi_{\text{out}})$	FID-50K $\downarrow$	sFID-50K $\downarrow$	Inception Score $\uparrow$	Precision $\uparrow$	Recall $\uparrow$
100K	-6	194.59	87.88	4.38	0.084	0.016
100K	-4	75.87	10.28	16.64	0.350	0.508
100K	-2	72.09	9.43	17.15	0.352	0.523
100K	0	75.35	10.67	16.61	0.363	0.514
100K	2	75.43	9.87	16.57	0.361	0.529
100K	4	72.26	9.47	17.08	0.378	0.514
100K	6	NaN	NaN	NaN	NaN	NaN
150K	-6	150.86	48.06	8.17	0.163	0.143
150K	-4	68.94	9.26	18.57	0.377	0.541
150K	-2	68.24	9.77	18.78	0.386	0.535
150K	0	67.48	9.45	18.83	0.387	0.553
150K	2	68.67	9.29	17.96	0.383	0.545
150K	4	67.32	9.08	18.17	0.397	0.539
150K	6	NaN	NaN	NaN	NaN	NaN
200K	-6	134.82	41.52	9.44	0.184	0.186
200K	-4	63.93	8.98	20.31	0.395	0.552
200K	-2	63.65	9.15	20.23	0.401	0.563
200K	0	61.65	9.06	20.60	0.407	0.563
200K	2	64.64	9.44	19.30	0.398	0.555
200K	4	59.85	9.39	20.95	0.415	0.566
200K	6	NaN	NaN	NaN	NaN	NaN

E.2.3 Additional Results of DiT-XL-2 Pretraining

Complete benchmark results of DiT-XL-2 and DiT-XL-2-μP throughout the training are provided in Table 14. DiT-XL-2-μP, using a base learning rate transferred from small proxy models, consistently outperforms the original DiT-XL-2 throughout the training process.

Table 14: **Benchmark results of DiT-XL-2 and DiT-XL-2-μP without classifier-free guidance.** Both models are trained on the ImageNet 256×256 dataset. Results with * are reported in the original paper [52], and others are reproduced by us. DiT-XL-2-μP, using a base learning rate transferred from small proxy models, consistently outperforms the original baseline throughout the training process.

Iteration	Method	FID-50K ↓	sFID-50K ↓	Inception Score ↑	Precision ↑	Recall ↑
0.1M	DiT-XL-2 [52]	48.17	7.33	26.61	0.49	**0.595**
	DiT-XL-2-μP	**44.15**	**6.72**	**28.54**	**0.54**	0.575
0.4M	DiT-XL-2 [52]	20.38	6.37	65.02	0.63	**0.635**
	DiT-XL-2-μP	**18.63**	**5.49**	**67.74**	**0.66**	0.606
0.8M	DiT-XL-2 [52]	14.73	6.37	85.03	0.66	**0.638**
	DiT-XL-2-μP	**12.91**	**5.48**	**89.46**	**0.68**	0.634
1.2M	DiT-XL-2 [52]	12.78	6.41	95.13	0.66	**0.644**
	DiT-XL-2-μP	**11.25**	**5.56**	**99.89**	**0.68**	0.638
1.6M	DiT-XL-2 [52]	11.79	6.49	100.88	0.67	**0.651**
	DiT-XL-2-μP	**10.16**	**5.59**	**107.58**	**0.69**	0.649
2M	DiT-XL-2 [52]	10.97	6.45	105.86	0.67	**0.657**
	DiT-XL-2-μP	**9.72**	**5.62**	**111.23**	**0.69**	0.651
2.4M	DiT-XL-2 [52]	10.75	6.59	108.23	0.67	**0.665**
7M	DiT-XL-2 [52]	9.62*	-	-	-	-
2.4M	DiT-XL-2-μP	**9.44**	**5.66**	**112.98**	**0.68**	0.653

E.2.4 Computational Cost

It takes 104 (13×8) A100-80GB hours to train the DiT-μP with a width of 288, a batch size of 256, and a train iteration of 200K steps. The computational cost of other DiT-μP proxy models can be inferred based on this situation. It takes around 224 (32×7) A100-80GB days to reproduce the pretraining of DiT-XL-2-μP.

E.3 Additional Details of PixArt-α Experiments

E.3.1 FLOPs ratio of HP Search on Proxy Models

Based on Equation (1) in the main text, we can derive that

$$\texttt{ratio} = \frac{RS_{\text{proxy}}E_{\text{proxy}}}{S_{\text{target}}E_{\text{target}}} = \frac{5 \times 0.04B \times 5}{0.61B \times 30} \approx 5.5\%.$$

Therefore, the HP tuning requires only 5.5% FLOPs of a single training run for PixArt-α.

E.3.2 Additional Results of Base Learning Rate Search

We present the detailed GenEval results [21] of different base learning rates in Table 15. Overall, the base learning rate 2^{-10} achieves the best GenEval performance.

Table 15: **GenEval results of base learning rate search on PixArt-α-μP proxy models.** 0.04B models with different learning rates are trained for 5 epochs on the SAM dataset. Overall, the base learning rate 2^{-10} achieves the best GenEval performance.

$\log_2(\text{lr})$	Overall ↑	Single	Two	Counting	Colors	Position	Color Attribution
-9							
-10	**0.083**	**30.63**	4.04	**2.5**	**12.23**	0.5	0
-11	0.078	29.38	**5.05**	1.25	9.57	**1.5**	**0.25**
-12	0.030	12.81	0	0.31	4.79	0.25	0
-13	0.051	20.62	3.54	0.31	5.59	0.25	0

E.3.3 Additional Results of PixArt-α Pretraining

We present the complete comparison between PixArt-α and PixArt-α-μP throughout training in Table 16. The results demonstrate that PixArt-α-μP consistently outperforms PixArt-α across all evaluation metrics during training.

Table 16: **Benchmark comparison between PixArt-α-μP and PixArt-α.** Both models are trained on the SAM dataset for 30 epochs. PixArt-α-μP with transferred base learning rate from the optimal 0.04B proxy model consistently outperforms the original baseline throughout the training process.

Epoch	Method	GenEval ↑	MJHQ		MS-COCO	
			FID-30K ↓	CLIP Score ↑	FID-30K ↓	CLIP Score ↑
6	PixArt-α [8]	0.14	43.19	25.16	42.25	27.02
	PixArt-α-μP (Ours)	**0.17**	**34.24**	**25.99**	**32.62**	**28.16**
10	PixArt-α [8]	0.19	38.36	25.78	34.58	28.12
	PixArt-α-μP (Ours)	**0.20**	**33.35**	**26.25**	**29.68**	**28.87**
16	PixArt-α [8]	0.22	36.19	26.21	33.71	28.35
	PixArt-α-μP (Ours)	**0.23**	**32.28**	**26.67**	**28.03**	**29.45**
20	PixArt-α [8]	0.20	35.68	26.54	30.13	28.81
	PixArt-α-μP	**0.23**	**33.42**	**26.83**	**29.05**	**29.53**
26	PixArt-α [8]	0.18	38.39	26.44	34.98	29.17
	PixArt-α-μP (Ours)	**0.28**	**32.34**	**27.17**	**27.59**	**29.83**
30	PixArt-α [8]	0.15	42.71	26.25	37.61	28.91
	PixArt-α-μP (Ours)	**0.26**	**29.96**	**27.13**	**25.84**	**29.58**

E.4 Additional Details of MMDiT Experiments

E.4.1 HPs Tuned by Human Experts

Algorithmic experts take roughly 5 times the cost of full pretraining to tune HPs based on their experience. The best HPs are a learning rate of 2E-4, a gradient clip of 0.1, a REPA loss weight of 0.5 and a warm-up iteration of 1K.

E.4.2 Additional Details of Base HP Search

Based on the optimal base HPs searched from 80 trials trained for 30K steps, we further evaluate the HP transferability across iterations. We conducted an additional 5 proxy training runs (100K steps each) using the searched optimal HPs and five different base learning rates, as detailed in Table 17.

Table 17: **Iteration of 30K is enough for proxy task.** The base learning rate of 2.5E-4 is optimal when proxy models are trained for 100K steps, consistent with the results observed at 30K steps.

Base learning rate	1E-4	2E-4	2.5E-4	4E-4	8E-4
Training loss	0.185028	0.184505	**0.184423**	0.184436	0.185562

E.4.3 FLOPs ratio of HP Search on Proxy Models

Based on Equation (1) in the main text, we can derive the ratio of tuning cost to single pretraining cost as

$$\texttt{ratio} = \frac{RS_{\text{proxy}}E_{\text{proxy}}}{S_{\text{target}}E_{\text{target}}} = \frac{80 \times 0.18B \times 30K + 5 \times 0.18B \times 100K}{18B \times 200K} = 14.5\%.$$

Besides, as algorithmic experts take roughly 5 times the cost of full pretraining, the ratio of μP tuning cost to standard human tuning cost is approximately 3%.

E.4.4 Additional Results of Training Loss Comparison

We present the complete comparison between the training loss of MMDiT-18B and MMDiT-μP-18B in Figure 8. MMDiT-μP-18B achieves consistently lower training loss than the baseline after 15K steps, and the advantage is gradually increasing.
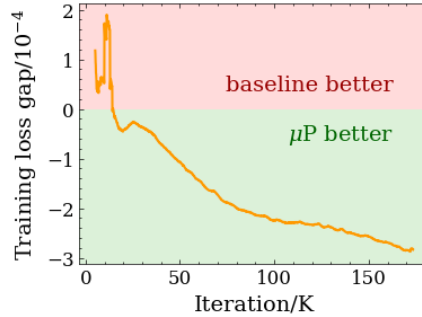

Figure 8: **Comparision between training loss.** We present the training loss of MMDiT-μP-18B minus that of MMDiT-18B. The training loss gap less than 0 means that MMDiT-μP-18B is better. MMDiT-μP-18B achieves consistently lower training loss than the baseline after 15K steps, and the advantage is gradually expanding.

E.4.5 Additional Details of Human Evaluation

In this section, we provide the details about the user study in Section 5.2. To evaluate the text-image alignment, we created an internal benchmark comprising 150 prompts, where each prompt includes an average of five binary alignment checks (yes or no), covering a wide range of concepts such as nouns, verbs, adjectives (e.g., size, color, style), and relational terms (e.g., spatial relationships, size comparisons). Ten human experts conducted the evaluation, with each prompt generating three

images, resulting in a total of 22,500 alignment tests. The final score is computed as the average correctness across all test points. We present two examples in Figure 9.
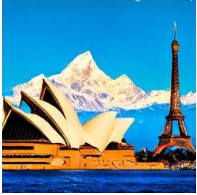
Prompt	Image	Alignment Evaluation
One wine glass, two bottles of wine, three cans of beer		1. Wine glass: ✅ 2. One (wine glass): ✅ 3. Bottles of wine: ✅ 4. Two (bottles of wine): ✅ 5. Cans of beer: ✅ 6. Three (cans of beer): ❌
The Sydney Opera House with the Eiffel tower sitting on the right, and Mount Everest rising above		1. Sydney Opera House: ✅ 2. Eiffel tower: ✅ 3. Right: ✅ 4. Mount Everest: ✅ 5. Rising above: ✅

Figure 9: **Examples of text-image alignment test evaluated by human.** Ten human experts conducted the evaluation, with each prompt containing an average of five test points and generating three images, resulting in 22,500 alignment test points. The final score is computed as the average correctness across all test points.

E.4.6 Additional Results of Visualization Comparison

In this section, we provide visual comparisons between MMDiT-μP-18B and MMDiT-18B baseline in Figure 10.
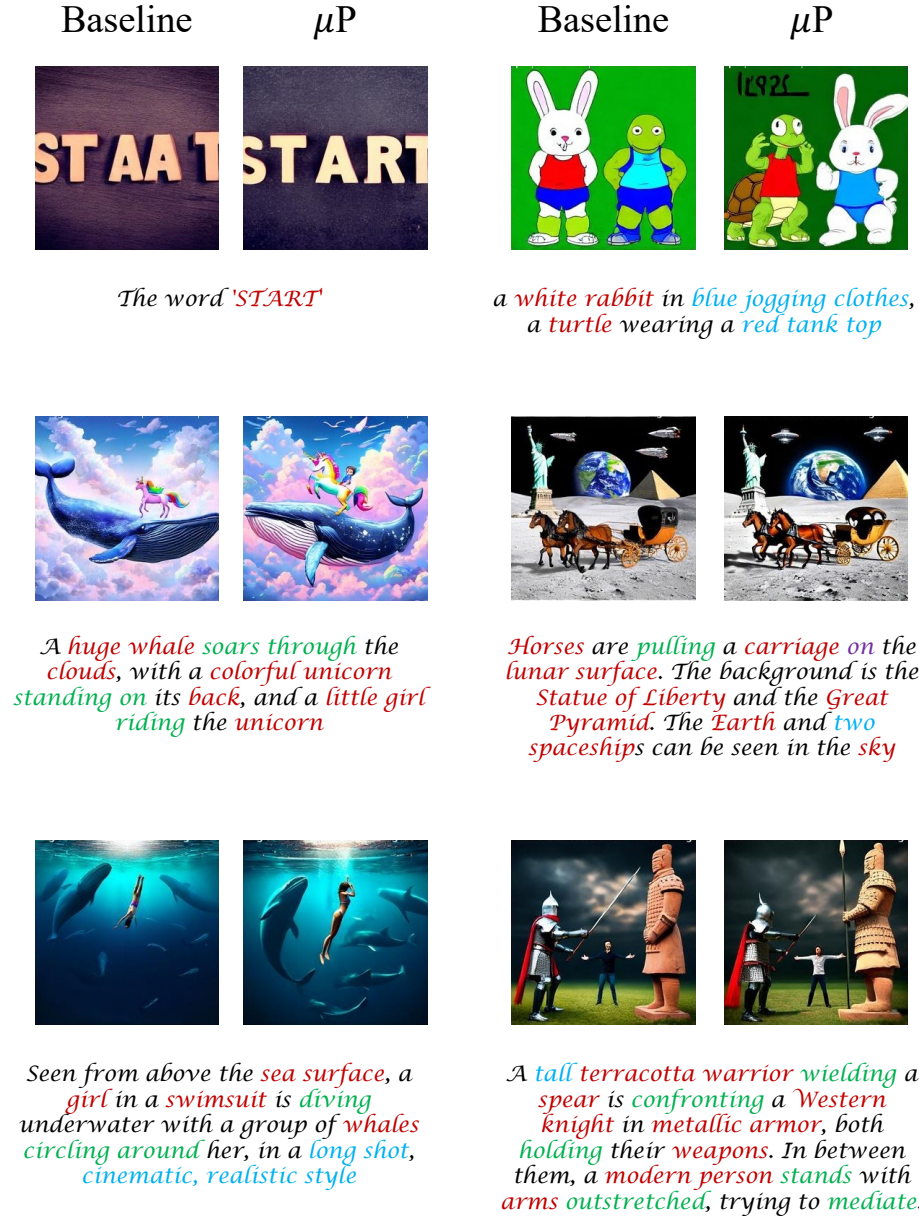
Figure 10: **Visual comparison between MMDiT-μP-18B and MMDiT-18B baseline.** The configurations for generating images (e.g., random seed, cfg value) of two models are the same. MMDiT-μP-18B shows better text-image alignment than the baseline model.