
Improving Progressive Generation with Decomposable Flow Matching

Moayed Haji-Ali^{*,†,1,2}Willi Menapace^{*,2}Ivan Skorokhodov²Arpit Sahni²Sergey Tulyakov²Vicente Ordonez¹Aliaksandr Siarohin²¹Rice University²Snap IncProject Webpage: <https://snap-research.github.io/dfm>

Abstract

Generating high-dimensional visual modalities is a computationally intensive task. A common solution is *progressive generation*, where the outputs are synthesized in a coarse-to-fine spectral autoregressive manner. While diffusion models benefit from the coarse-to-fine nature of denoising, explicit multi-stage architectures are rarely adopted. These architectures have increased the complexity of the overall approach, introducing the need for a custom diffusion formulation, decomposition-dependent stage transitions, add-hoc samplers, or a model cascade. Our contribution, Decomposable Flow Matching (DFM), is a simple and effective framework for the progressive generation of visual media. DFM applies Flow Matching independently at each level of a user-defined multi-scale representation (such as Laplacian pyramid). As shown by our experiments, our approach improves visual quality for both images and videos, featuring superior results compared to prior multistage frameworks. On Imagenet-1k 512px, DFM achieves 35.2% improvements in FDD scores over the base architecture and 26.4% over the best-performing baseline, under the same training compute. When applied to finetuning of large models, such as FLUX, DFM shows faster convergence speed to the training distribution. Crucially, all these advantages are achieved with a single model, architectural simplicity, and minimal modifications to existing training pipelines.

1 Introduction

The high dimensionality of images and videos poses a challenge for generative modeling. An approach to make the problem more tractable involves decomposing the generative task into a sequence of simpler sub-problems. An increasing number of generative models adopt this practice by progressively modeling visual signals of increasing resolution [1, 11, 16, 18, 41, 42]. Autoregressive generative models, for instance, have demonstrated that switching from a next-token prediction pattern in a 1D sequence of flattened tokens [4, 6, 24, 31, 35, 49, 50] to a next-scale [42] or next-frequency [16] prediction pattern, yields substantial performance improvements. Recently, the success of diffusion models has been connected to a form of spectral autoregression enforced by the diffusion process which erases details progressively from the coarsest to the finest [30, 36, 38].

A variety of diffusion frameworks [1, 9, 13, 14, 18, 37, 41] have reformulated the diffusion process to enforce spectral autoregression, an approach we refer to as *progressive modeling*. This is typically

^{*}Equal Contribution.

[†]Work partially done during an internship at Snap Inc.

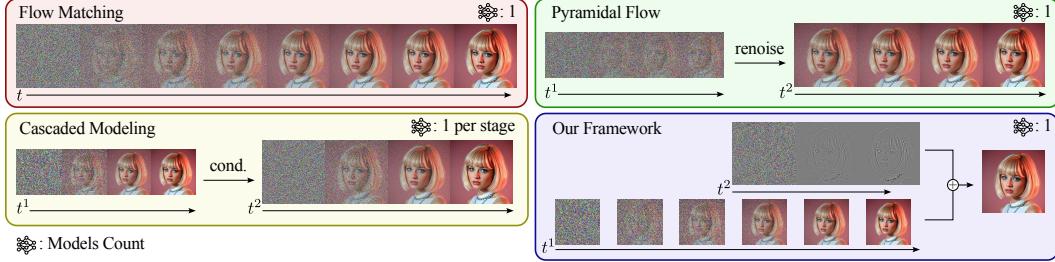


Figure 1: Comparison of different diffusion frameworks. Cascaded models employ a separate model per stage. Pyramidal Flow [18] progressively increases resolution through careful upsampling and renoising operations. Our framework models multiple stages independently from each other within the same model, enabling progressive generation with minimal modifications to Flow Matching [27]

realized by restructuring the diffusion process to include transition points where the signal resolution is altered, thereby defining a sequence of stages of progressively increasing resolution. Key progressive modeling paradigms are illustrated in Figure 1, where cascaded models [13, 14, 37] and Pyramidal Flow [18] are exemplified and contrasted with vanilla Flow Matching [26, 27]. However, while these frameworks demonstrate the feasibility of the progressive paradigm, they often operate directly in pixel space [1, 41], depend on multiple models [13, 14, 37, 41], or deviate from the original formulation of the diffusion process [1, 9, 18, 41]. Such deviations introduce complexities in managing inter-stage transitions, requiring careful consideration of operations such as resizing [1, 18, 41], renoising [1, 18, 41] or blurring [41] and, in some instances, require specialized samplers [9, 41].

To address these challenges, we propose Decomposable Flow Matching (DFM), a progressive modeling framework based on a simple extension of Flow Matching, depicted in Figure 2. At its core, a user-defined decomposition function is applied to the input, producing a multiscale input representation that defines each *stage* of the generation, and a Flow Matching process is independently applied to each stage, utilizing a set of per-stage flow timesteps. During inference, a standard sampler operates separately on each stage of the representation guided by a user-defined sampler schedule that dictates the progression of flow timesteps per each stage. Timesteps are evolved sequentially, starting from the coarsest stage; progression to a subsequent stage only begins once the preceding stage has reached a predefined threshold. During training, progressive generation is simulated by selecting a stage, sampling a noise level for it, and injecting the maximal noise level into subsequent stages of the representation and a small noise level to preceding ones. These capabilities are realized without the need for expert models, ad-hoc diffusion processes, or specialized samplers, all while offering the flexibility of user-defined input decomposition.

We conduct a thorough analysis of the design space for DFM, providing insights into its behavior across different training and sampling strategies, and model architectures. Based on these findings, we instantiate DFM and evaluate its performance on established image and video generation benchmarks, specifically ImageNet-1K [5] and Kinetics-700 [3]. Our results demonstrate uniform improvements over Flow Matching [27], cascaded models, and Pyramidal Flow [18]. Furthermore, we apply DFM to the finetuning of FLUX [23], demonstrating its efficacy in enhancing the performance of large-scale generative models. Compared with standard full finetuning, applying DFM for the same amount of finetuning iterations yields a 29.7% reduction in FID and 3.7% increase in CLIP score.

In summary, our work introduces Decomposable Flow Matching, a novel progressive generation framework that offers several key advantages: (i) It presents a simple formulation rooted in Flow Matching; (ii) it is agnostic to the choice of decompositions; (iii) it does not need multiple models or per-stage training; (iv) it includes a comprehensive analysis of critical training and sampling hyperparameters; (v) it outperforms existing state-of-the-art progressive generation frameworks on ImageNet-1K [5] and Kinetics-700 [3], and enhances FLUX [23] finetuning performance.

2 Related Work

2.1 Progressive Generation in Autoregressive Models

Autoregressive models for visual generation [4, 6, 24, 31, 35, 49, 50] formulate the generation of visual data as a next-token prediction problem. This is typically achieved by first obtaining a

discrete input representation, which is then flattened into a 1D sequence to enable autoregressive modeling. However, recent work [16, 42] has demonstrated that flattening undermines the intrinsic multidimensional structure of visual data. VAR [42] addresses this limitation by modeling image generation as a spectral autoregressive process. A multiscale tokenizer generates discrete tokens at different resolutions, and the model predicts each scale conditioned on the previous ones, yielding substantial improvements over baselines. NFIG [16] adopts a similar approach where frequency decomposition is used to define different scales that are predicted autoregressively. In our work, we explore spectral autoregression in the context of diffusion models.

2.2 Progressive Generation in Diffusion Models

Diffusion models have achieved state-of-the-art results across various visual generation tasks [7, 23, 28, 46]. Their effectiveness has been linked to an implicit form of *spectral autoregression*, whereby the diffusion process erases fine-grained details in a coarse-to-fine fashion [30, 36, 38].

Several *progressive generation* frameworks made spectral autoregression an explicit part of their design, generating their outputs stage-by-stage. Cascaded diffusion models [13, 14, 37] generate low-resolution outputs in an initial stage, followed by upsampling stages conditioned on previous outputs. A separate model is used for each stage. Other methods redefine the generative process to incorporate progressive signal transformations. f-DM [9] and RDM [41] introduce stages via downsampling. MDM [10] simultaneously denoises multiple resolutions. Edify Image [1] introduces frequency-specific attenuation in the diffusion process. Pyramidal Flow Matching [18] operates in the latent space over a hierarchy of increasing resolutions, combining upsampling and re-noising operations. Despite these advances, existing approaches share limitations. They often require complex stage-transition mechanisms [1, 9, 18, 41] or tailored samplers [9, 41], are tightly coupled to specific decomposition strategies [1, 9, 18, 41], and primarily focus on pixel-space generation [1, 10, 41]. In contrast, we propose a simple and generic framework for progressive generation in the latent space that supports user-defined decompositions without altering the core architecture.

2.3 Autoencoders with Spectral Latent Structure

Coarse-to-fine or *spectral autoregressive* modeling benefits from an autoencoder with a well-structured spectral representation where low-, mid-, and high-frequency components in the RGB space correspond to the respective low-, mid-, and high-frequency components in the latent representation. Modern image and video autoencoders typically lack such spectral structure [38], so recent works fine-tune them to enforce the desired spectral properties through scale equivariance regularization [17, 22, 38]. Another viable strategy is to train a Spectral Image Tokenizer [8]-like model from scratch, which encodes a wavelet-decomposed image representation into a causally ordered sequence, yielding spectrally structured latents by design. Our framework is agnostic to the choice of the autoencoder but benefits from such spectral latent structure, and we opt for the first strategy of fine-tuning with scale equivariance [22, 38] to stay closer to existing diffusion pipelines.

3 Method

This section details our progressive generation framework summarized in Figure 2. The method is based on Flow Matching [26, 27], which is introduced in Section 3.1. A Laplacian decomposition strategy, detailed in Section 3.2, is exemplified as a way of producing a multiscale input representation amenable to stage-by-stage progressive modeling. In Section 3.3, the Flow Matching framework is extended to account for the multiple stages by introducing per-stage flow timesteps which are leveraged to instantiate the progressive generation scheme by denoising stages one-by-one, starting from the coarsest. Finally, the DiT [33] architecture is extended in Section 3.4 to support the proposed generation scheme by introducing per-scale patchification and time embedding layers.

3.1 Flow Matching

We base our generative models on the Flow Matching framework [26, 27], which defines a generation of data $\mathbf{X}_1 \sim p_d$ as the progressive transformation of \mathbf{X}_0 following a path connecting samples from the two distributions. Often, the path is instantiated as a linear interpolation: $\mathbf{X}_t = t\mathbf{X}_1 + (1 - t)\mathbf{X}_0$, and $\mathbf{X}_0 \sim p_n = \mathcal{N}(0, \mathbb{I})$ originates from a noise distribution [27]. The velocity along this path is

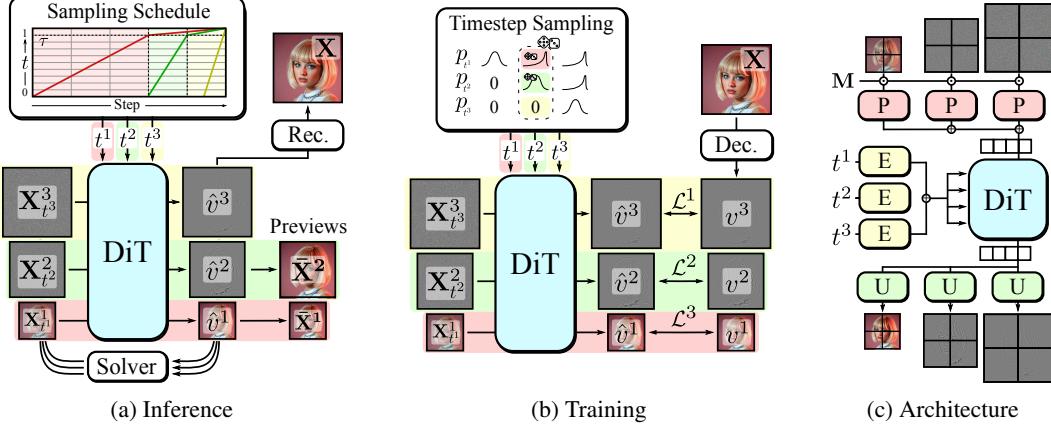


Figure 2: Overview of Decomposable Flow Matching (a) Sampling procedure. A sampling schedule is defined that dictates the timestep progression of each stage. The model predicts velocities for each input scale and a vanilla sampler denoises each scale according to its schedule. Intermediate stages can be decoded during inference to provide a low-resolution preview of the generation. (b) Training procedure. A stage index is sampled from a discrete probability distribution and defines the main stage. Timesteps for each stage are sampled accordingly. The model minimizes velocity prediction errors for each scale. (c) DiT [33] architecture for progressive generation. Patchification layers and time embedders are replicated for each scale.

given by $v_t = \frac{d\mathbf{X}_t}{dt} = \mathbf{X}_1 - \mathbf{X}_0$. A generator \mathcal{G} is trained to predict this velocity by minimizing:

$$\mathcal{L} = \mathbb{E}_{t \sim p_t, \mathbf{X}_1 \sim p_d, \mathbf{X}_0 \sim p_n} \|\mathcal{G}(\mathbf{X}_t, t) - v_t\|_2^2, \quad (1)$$

where p_t is a training-time distribution over t . At inference, samples are generated from noise using an ODE solver (e.g., Euler), integrating the learned velocity field to produce data samples.

3.2 Input Representations for Progressive Modeling

The high dimensionality of visual modalities makes their modeling challenging. This complexity can be reduced by factorizing their generation into simpler *stages*, instantiating a progressive generation framework which generates coarse details such as structure first, and finer details later.

To this end, each input \mathbf{X} is structured into a multiscale representation $\mathbf{P} = \{\mathbf{X}^s\}_{s=1}^S$ across S scales, each defining a generative *stage*, containing a progressively increasing level of detail, from the most coarse to the finest. Various decomposition techniques are viable, including DWT, DCT, Fourier transforms, or multiscale autoencoders [8]. We adopt the Laplacian pyramid due to its simplicity and flexibility, but note that, unlike existing methods, our framework is agnostic to the particular choice of decomposition. The pyramid levels are defined as:

$$\begin{cases} \mathbf{X}^s = \text{down}(\mathbf{X}, s) - \text{up}(\text{down}(\mathbf{X}, s-1), s) & \text{if } s > 1 \\ \mathbf{X}^1 = \text{down}(\mathbf{X}, 1) & \text{if } s = 1 \end{cases}, \quad (2)$$

where each scale s is associated with a user-defined resolution, and $\text{down}(\dots, s)$ and $\text{up}(\dots, s)$ respectively represent downsampling and upsampling to the resolution of s . A downsampled approximation of the input can be obtained as $\bar{\mathbf{X}}^s = \mathbf{X}^1 + \dots + \mathbf{X}^s$.

3.3 Decomposable Flow Matching

The multiscale input is amenable to progressive modeling, where each scale from the coarsest to the finest defines a generation *stage*. We extend Flow Matching to support multistage generation. To this end, we introduce an independent flow timesteps t^s for each generation stage, and formulate the forward process as:

$$\mathbf{P}_{t^1, \dots, t^S} = \{\mathbf{X}^s\}_{s=1}^S \quad \text{where} \quad \mathbf{X}^s = t^s \mathbf{X}_1^s + (1 - t^s) \mathbf{X}_0^s. \quad (3)$$

The factorization of flow timesteps enables control over stage generation by imposing conditional generation on previous stages and marginalization on successive stages by respectively applying low or full noise levels [2]. Consequently, rather than predicting a single velocity, the model predicts per-scale velocities v^s using a shared generator \mathcal{G} conditioned on the full set of noisy input scales and their timesteps, which is learned by minimizing:

$$\mathcal{L} = \mathbb{E}_{t^1, \dots, t^S \sim p_t, \mathbf{X}_1 \sim p_d, \mathbf{X}_0 \sim p_n} \sum_{s=1}^S \mathbf{M}^s \|\mathcal{G}(\mathbf{P}_{t^1, \dots, t^s}, t^1, \dots, t^S) - v^s\|_2^2, \quad (4)$$

where p_t represents the joint training distribution of scale timesteps.

During inference, a timestep schedule is defined that activates stages progressively as shown in Figure 2a. The sampler starts from the first stage and advances t^1 only, before introducing subsequent scales one at a time once the last introduced scale reaches a predefined threshold τ in its generation. Then, the last stage along with all previous stages continue the generation at different rates (see Sampling Schedule in Figure 2 (a)). Each time the generated scale reaches its threshold, the representation can undergo partial decoding up to the current stage, producing an output preview. At the end of the generation, all scales representations are summed to visualize the generated output.

The training timestep distribution p_t is central to effective learning of this progressive strategy, and training timesteps are chosen to simulate the progressive sampling through stages, as shown in Figure 2b. First, a stage index, representing the number of stages that are currently under generation, is sampled from a discrete probability distribution having density p_t^s for stage s . Given the number of sampled stages, we sample the timestep of the last stage according to a logit normal distribution [7]. For every preceding stage, we instead sample the respective timestep from a logit normal distribution shifted towards lower noise levels to simulate inference time behavior where the earlier stages are less noisy than the last stage currently being generated. Finally, the timesteps for each subsequent stage are set to zero. To prevent the model from spending its capacity in modeling such zeroed scales. We additionally introduce a masking term \mathbf{M}^s that ignores their loss contributions. This promotes progressive learning dynamics, where the model focuses on global structure before local detail, enabling high-quality progressive generation.

3.4 Architectural Adaptations

We choose to experiment on a DiT architecture [33] due to the pervasive adoption of its variants [7, 23, 25, 28, 46, 48], and we extend it to accommodate multiscale inputs and their associated flow timesteps (see Figure 2c). At their core, a DiT transforms the input visual modality into a sequence of tokens through a linear layer called *patchification* layer, where each token corresponds to a small portion of the input of size $k \times k$, denoted as *patch size*. An inverse operation is performed by the output patchification layer. A sequence of transformer blocks [45] processes the token sequence. To make the model aware of the input timestep, a *timestep embedder* MLP produces a respective conditioning embedding which is to the transformer blocks through modulation layers.

To accommodate multiscale inputs, rather than using single input and output patchification layers, dedicated input and output patchification layers are instantiated per scale. Patch sizes are chosen such that each input scale yields an equal number of patches, ensuring consistent spatial alignment across scales. For example, for two stages of resolution 256×256 and 512×512 , we use patch sizes of 1×1 and 2×2 , respectively. The resulting patch embeddings from all scales are summed and fed into the transformer backbone. Finally, a dedicated per-scale projection layer is used to output each scale velocity. Loss masks \mathbf{M} are integrated to zero out model inputs at masked scales. This prevents noisy inputs whose contribution to the loss is negated by the mask from harming model performance. To incorporate the per-scale timesteps, we use a dedicated timestep embedder for each level and sum their outputs to form a unified conditioning signal.

4 Experiments

4.1 Experimental setup

Latent space The FLUX [23] and Cogvideo [48] autoencoders are used, respectively, to produce the latent spaces for all image and video experiments. To maximize the benefits of coarse-to-fine modeling, low-, mid-, and high-frequency components in the latents should correspond to

Table 1: Ablation results on Imagenet-1K [5] using a DiT-B backbone.

Res.	FID _{10K} ↓	FDD _{10K} ↓	IS ↑		Res.	FID _{10K} ↓	FDD _{10K} ↓	IS ↑		Res.	FID _{10K} ↓	FDD _{10K} ↓	IS ↑					
(a) First stage samp. prob. p_t^0																		
0.7	512	36.02	676.0	32.2	baseline	512	29.56	571.0	41.0	↑ Batch size	512	31.95	615.0	39.9				
0.8	512	32.09	622.0	36.1	-masking	512	29.9	584.0	40.0	↓ Patch size	512	29.56	571.0	41.0				
0.9	512	29.56	571.0	41.0	-standardization	512	27.13	505.0	42.8									
0.95	512	27.5	540.0	43.7	(c) Architecture													
(b) Second stage lognorm location														(e) Compute Allocation				
0.0	512	29.83	567.0	40.4	None	512	28.37	564.0	41.5	512 → 1024	1024	58.03	899.0	20.3				
0.5	512	28.89	569.0	41.5	Mod.	512	28.95	568.0	41.4	256 → 512 → 1024	1024	50.61	827.0	24.5				
1.0	512	29.14	567.0	41.2	Proj.	512	29.56	571.0	41.0	256 → 1024	1024	41.06	704.0	30.9				
1.5	512	29.56	571.0	41.0	Cond.	512	28.88	565.0	40.0									
2.0	512	29.96	589.0	39.8	Attn.	512	28.99	564.0	40.7									
Tied	512	55.49	885.0	21.4	MLP	512	27.65	545.0	43.0									
					Full	512	26.63	526.0	45.5									
(d) Parameter Specialization														(f) Input Decomposition				
														512 → 1024	1024	58.03	899.0	20.3
														256 → 512 → 1024	1024	50.61	827.0	24.5
														256 → 1024	1024	41.06	704.0	30.9

the respective low-, mid-, and high-frequency RGB components once decoded, a property not typically found in modern autoencoders. We establish it for all autoencoders by scale-equivariant finetuning [38]. Namely, we fine-tune the FLUX autoencoder for 10,000 steps on our internal image dataset at 256^2 -resolution with $\times 2\text{-}4$ scale equivariance regularization [38] with a batch size of 64.

Training details We train our models on the ImageNet-1K [5] and Kinetics-700 [3] datasets. We run all ablations using DiT-B and main experiments on the DiT-XL architecture. All experiments use a batch size of 256, a learning rate of $1\text{e-}4$ with 10k steps warmup, gradient clipping of 1.0, the Adam [21] optimizer, and an EMA with $\beta = 0.9999$. We train ablations on ImageNet-1K for 600k steps, and the main experiment for 500k and 350k steps, respectively, for 512px and 1024px resolution. For Kinetics-700, we train the main experiments for 200k steps.

Evaluation metrics and protocol We report FID, and Inception Score (IS) for image experiments and frame-FID and FVD [43] for video experiments. We also include FDD (Frechet Distance computed against DINOv2 [32] features) as it was shown to be a more reliable metric than FID [20, 39]. We use 50k generated samples for ImageNet-1K 512px evaluation and 10k samples for the rest of the experiments. We report main experiment results with cfg [12] values of 1.0, 1.25, and 1.5 as [33].

4.2 Training Parameters Ablation

We conduct ablations to provide insights into the behavior of the main training hyperparameter. Unless otherwise specified, the ablations are run on ImageNet-1K [5] 512px using a DiT-B backbone. The default backbone employs a 2-stage Laplacian decomposition of 256px and 512px resolution, respectively, where each decomposed input scale is normalized to unit standard deviation. Results are presented in Table 1, and practical guidance on their optimal configuration is provided in Appendix Sec. A.5.

Training timestep distribution. The sampling distribution of timesteps p_t during training is crucial as it determines the implicit loss weighting for each stage (see Section 3.3). We parametrize its design across two dimensions. The first is the distribution over the stage index s , which corresponds to a number of scales being trained. Since we ablate over a 2-stage configuration, this can be expressed as the probability of sampling stage 0 (p_t^0). The second is the location for the logit normal distribution [7] of all preceding stages, representing the amount of bias towards lower noise levels for already-generated stages. We report the performance of various sampling strategies in Table 1 (a) and (b). Performance is positively impacted by large probabilities of sampling stage 0, indicating that allocating more model capacity towards predicting structural details benefits generation. This is aligned with previous cascaded model training strategies, where the first stage model is trained longer than the second stage model [29]. Performance is also positively impacted by larger shifts in the logit normal distribution of already-generated stages, as this better simulates the inference-time sampling schedule. Finally, we consider a strategy that ties the timesteps of every stage, enforcing the same value for them, and accordingly modify sampling. This variation presents degraded performance as it loses the benefits of progressive generation. We select $p_t^0 = 0.9$ and a location of 1.5 for our main experiments, as we speculate that larger models may afford spending more capacity for non-structural details and would benefit from a closer alignment to the inference-time schedule.

Architecture. We ablate our baseline by removing input masking (see Section 3.4) and removing the normalization of each stage to unit standard deviation before the application of the diffusion

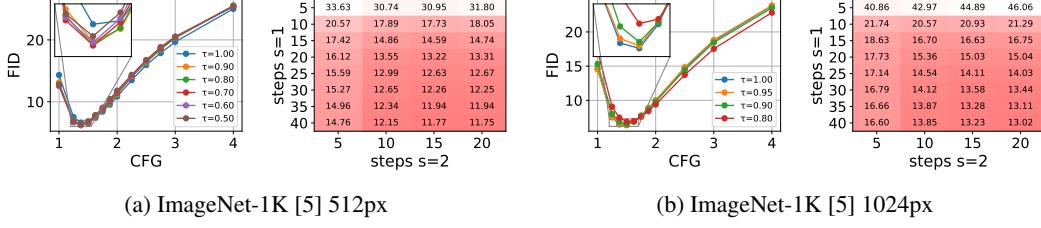


Figure 3: FID_{10K} results ablating sampling configuration, threshold, and per-stage steps.

process. We report the results in Table 1 (c). Removing input masking for successive stages degrades performance over all metrics due to the negative influence of injecting pure noise tensors into the model. However, we found that removing standardization improves performance. Removing normalization results in a lower magnitude for the input of the second stage. This affects the model in a similar way to increasing p_t^0 , giving more importance to stage 1 structural modeling and improving performance. Therefore, we adopt masking but exclude standardization for our main experiment.

Parameter Specialization. We leverage the factorization of generation into multiple stages by exploring model parameter specialization for each stage. Specifically, in addition to base model parameters, we create a set of per-stage model parameters acting as *stage experts*, akin to using distinct sets of model parameters in cascaded models. During the generation of stage s , the specialized version of the weights corresponding to s is averaged with the base model parameter, enabling scale-specific behavior without significant computational overhead. As shown in Table 1 (d), parameter specialization improves performance when applied to MLP layers and produces the best results when applied to every model parameter. Despite its performance benefit, we exclude this technique from our main experiments to keep the parameter count similar across baselines.

Compute allocation. Compared with a single-stage model, input decomposition entails a lower dimensionality for earlier stages which results in saved computation at similar settings. In the ablations, for example, the first stage corresponds to 256px inputs rather than 512px. Such computational savings can be leveraged by either increasing the training batch size and number of inference steps or reducing the DiT patch size so that each stage results in an equal number of tokens. We observe performance increases at a steady rate when reducing patch size, yet increasing the batch size quickly reaches a plateau, converging to lower performance as reported in Table 1 (e). We select the patch size reduction strategy for the final model.

Input Decomposition. Our framework is agnostic to the exact choice of input decomposition, and thus different number of stages or different resolutions at different stages can be seemingly applied. We explore different configurations of the Laplacian decomposition for 1024px ImageNet-1k generations and report the results in Table 1 (f). Each decomposition is denoted as the sequence of resolutions corresponding to each stage *e.g.* 256→1024 for a 2-stage decomposition with 4x downsampling between stages. The 512→1024 decomposition yields the worst performance due to the initial stage being performed in 512px resolution and containing a larger amount of non-structural components, reducing the effectiveness of the progressive framework. We find 256→1024 to yield the best result. Consequently, we adopt 2-stage decompositions of 256→512 and 256→1024, respectively, for 512px and 1024px experiments.

4.3 Sampling Parameters Ablation

This section analyzes the effects of sampling parameters on a DiT-XL trained on ImageNet-1k 512px and 1024px using the optimal training parameters discussed in Section 4.2. Results are given in Figure 3, and practical guidance on their optimal configuration is provided in the Appendix Sec. A.5.

Stage Timestep Thresholds. We analyze the optimal timestep threshold τ determining the switch between sampling stages. We choose a threshold of 0.7 for 512px as it produces the best FDD and second-best FID at the respective optimal cfg level and 0.95 for 1024px experiments.

Per-Stage Sampling Steps. We evaluate model performance using different numbers of steps allocated to the sampling of the first and second stages under a sampling threshold of 0.7 and 0.95 respectively for 512px and 1024px. Allocating a larger number of steps to the first stage gives overall

Table 2: Comparison to baselines on ImageNet-1K [5] and Kinetics-700 [3] using a DiT-XL model and training convergence plot for ImageNet-1K [5] 512px.

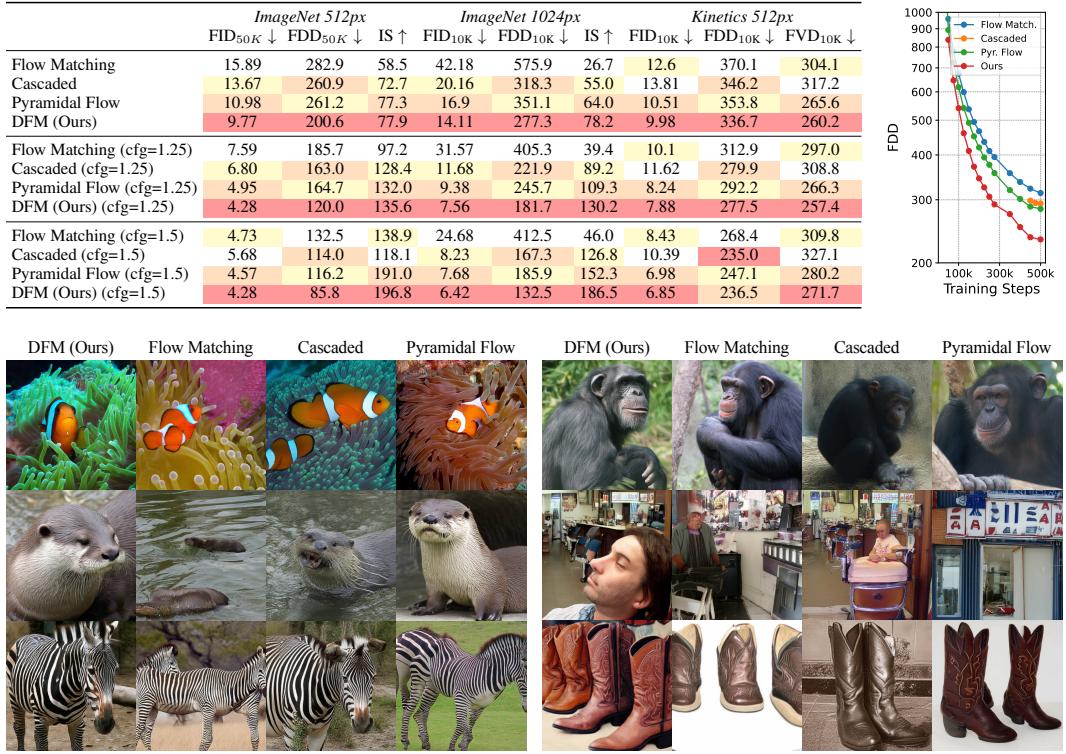


Figure 4: Comparison of DFM against baselines on ImageNet-1K 512px [5] on selected samples. Samples are generated with cfg 3.0

better performance, matching the intuition that the generation of structural details represents a more crucial component of sample quality than fine-grained details. We fix a total number of steps equal to 40 for our experiments and choose 30 and 10 sampling steps, respectively, for the two stages.

4.4 Comparison to baselines

Baselines selection and details. We select baselines corresponding to the main generation paradigms in Figure 1, which we train on the same underlying model backbone, ensuring a constant amount of training compute. The cascaded model baseline consists of separate models, one for each stage. Two stages are employed to avoid excessive error accumulation. We train the first stage model for 80% of the total training iterations and use it to initialize the second stage model for faster convergence. Pyramidal Flow [18] is chosen due to its state-of-the-art progressive generation performance and due to it operating in latent spaces while using a rectified flow formulation. We operate it in a 2 and 3-stage configuration for 512px and 1024px datasets, ensuring the same base resolution across experiments. All baselines are configured to ensure the same amount of training compute. Please refer to Appendix Sec. B for exact details on each baseline.

Results. As shown in Table 2, our method surpasses the baselines across all metrics and cfg settings with the exception of Kinetics-700 [3] FDD. Notably, DFM matches baselines best reported FDD in roughly half of the number of iterations, while using the same training compute (See Figure 4, right) Qualitative comparison (see Figure 4 and Appendix Sec. C) suggests that DFM generates better structural details, an observation which we attribute to its progressive formulation which focuses the model’s capacity on its lowest resolution stage.

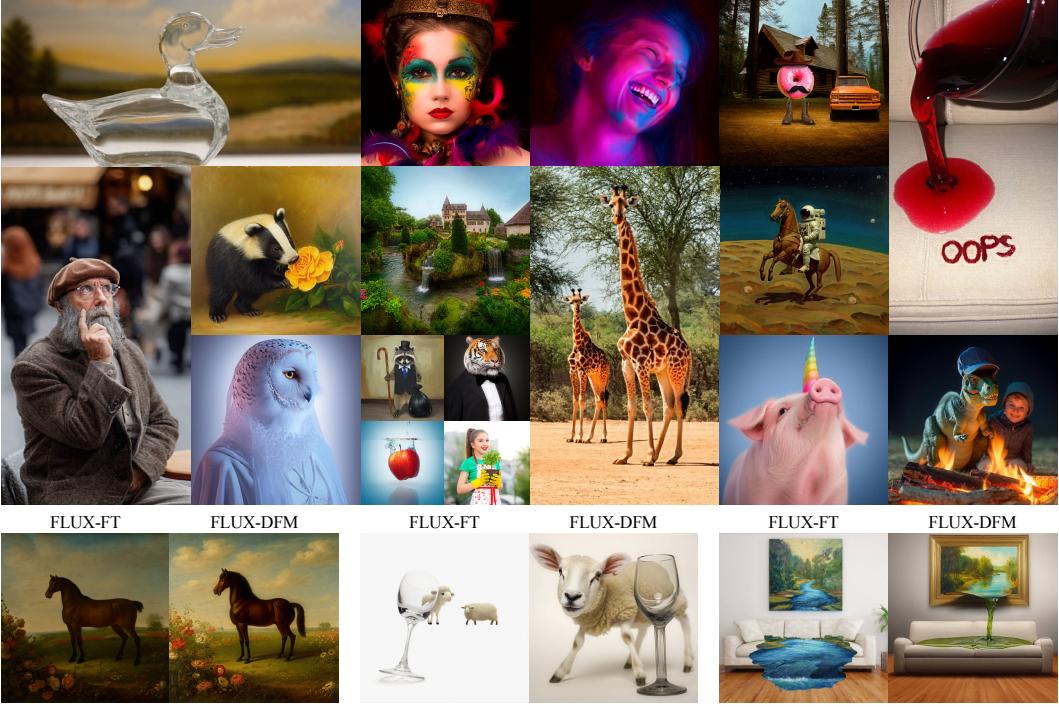


Figure 5: (Top) Qualitative results produced by FLUX [23] finetuned with DFM. (Bottom) a comparison of FLUX-DFM to the baseline (FLUX-FT).

4.5 Finetuning of large-scale models

To evaluate the effectiveness of DFM in large-scale generative models, we finetune FLUX-dev [23] on our internal image dataset using DFM and scale equivariant autoencoder. First, the original FLUX model is finetuned for 24k steps to adapt it to the scale equivariant autoencoder. Starting from this checkpoint, we branch two finetunings. The first continues regular finetuning to obtain the FLUX-FT checkpoint, the second finetunes for our framework to obtain the FLUX-DFM checkpoint. Both are finetuned for 32k steps under the same settings and training compute. We use FLUX-dev patchification and projection layers weights to initialize the first stage layers of FLUX-DFM, and zero-initialize the second stage layers. We train on multiple aspect ratios and use a total batchsize of 192 for the 1024px resolution. More details on the finetuning are included in the Appendix Sec. E. We compare the baselines by generating 10k samples from a validation split of our internal dataset and report FID, FDD, and CLIP [34] similarity (CLIPSIM) in Table 3. We include qualitative results and comparisons for FLUX-DFM in Figure 5 and Appendix Sec. C. FLUX-DFM achieves consistently better performance compared with standard full finetuning, improving FID and FDD by 28.7% and 24.3%, respectively. We report original FLUX-dev checkpoint numbers as a reference. Please note that in this experiment, we do not aim to improve FLUX-dev quality but rather show that finetuning with DFM achieves overall faster convergence in learning the finetuning dataset distribution. Thus, report FID and FDD as reliable metrics comparing FLUX-FT and FLUX-DFM.

5 Discussion

Limitations Similarly to other progressive generation frameworks [1, 9, 18, 41] our method introduces additional training and sampling hyperparameters. The framework’s most important hyperparameters balance between learning of low and high-frequency visual components, enabling users to pose more emphasis on structural details. When excessive importance is put on structural details, a decreased presence of high-frequency details is noticed. We provide extensive ablations, and practical guidance

Table 3: FLUX fine-tuning comparison.

Method	FID _{10K} ↓	FDD _{10K} ↓	CLIPSIM ↑
FLUX	17.79	172.7	0.3280
FLUX-FT	12.56	153.6	0.3261
FLUX-DFM	8.83	116.2	0.3381

for their setup, and show that the same parameters can be reused across different settings. We provide further ablations and guidance in *Appendix Sec. A* and discuss further our framework limitations and failed experiments in *Appendix Sec. F* and *Appendix Sec. G*

Conclusions We propose DFM, a novel framework for the progressive generation of images and videos. Differently from existing formulations, our framework is simple, agnostic to the choice of decompositions and does not necessitate multiple networks. We analyze it to provide insights into its main training and sampling hyperparameters and show that it outperforms both Flow Matching and the state-of-the-art progressive diffusion formulations on both ImageNet-1K [5] and Kinetics-700 [3]. We further validate the effectiveness of DFM on large-scale models, showing its faster convergence than standard full fine-tuning. A promising avenue for future work is to explore other decomposition paradigms such as discrete cosine, wavelet transforms, and multiscale autoencoders. Extending these findings to large-scale settings remains an exciting direction for future research.

Acknowledgments The authors thank Anil Kag and Dishani Lahiri for their engineering support.

References

- [1] Yuval Atzmon, Maciej Bala, Yogesh Balaji, Tiffany Cai, Yin Cui, Jiaojiao Fan, Yunhao Ge, Siddharth Gururani, Jacob Huffman, Ronald Isaac, et al. Edify image: High-quality image generation with pixel space laplacian diffusion models. *arXiv preprint arXiv:2411.07126*, 2024.
- [2] Fan Bao, Shen Nie, Kaiwen Xue, Chongxuan Li, Shi Pu, Yaole Wang, Gang Yue, Yue Cao, Hang Su, and Jun Zhu. One transformer fits all distributions in multi-modal diffusion at scale. In *International Conference on Machine Learning*, pages 1692–1717. PMLR, 2023.
- [3] Joao Carreira, Eric Noland, Chloe Hillier, and Andrew Zisserman. A short note on the kinetics-700 human action dataset. *arXiv preprint arXiv:1907.06987*, 2019.
- [4] Mark Chen, Alec Radford, Rewon Child, Jeff Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *ICML*. JMLR.org, 2020.
- [5] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2009.
- [6] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12873–12883, 2021.
- [7] Patrick Esser, Sumith Kulal, Andreas Blattmann, Rahim Entezari, Jonas Müller, Harry Saini, Yam Levi, Dominik Lorenz, Axel Sauer, Frederic Boesel, et al. Scaling rectified flow transformers for high-resolution image synthesis. In *ICML*, 2024.
- [8] Carlos Esteves, Mohammed Suhail, and Ameesh Makadia. Spectral image tokenizer. *arXiv preprint arXiv:2412.09607*, 2024.
- [9] Jiatao Gu, Shuangfei Zhai, Yizhe Zhang, Miguel Angel Bautista, and Josh Susskind. f-dm: A multi-stage diffusion model via progressive signal transformation. *ICLR (ICLR)*, 2023.
- [10] Jiatao Gu, Shuangfei Zhai, Yizhe Zhang, Joshua M Susskind, and Navdeep Jaitly. Matryoshka diffusion models. In *The Twelfth International Conference on Learning Representations*, 2023.
- [11] Moayed Haji-Ali, Willi Menapace, Aliaksandr Siarohin, Ivan Skorokhodov, Alper Canberk, Kwot Sin Lee, Vicente Ordonez, and Sergey Tulyakov. Av-link: Temporally-aligned diffusion features for cross-modal audio-video generation. *arXiv preprint arXiv:2412.15191*, 2024.
- [12] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2022.
- [13] Jonathan Ho, Chitwan Saharia, William Chan, David J Fleet, Mohammad Norouzi, and Tim Salimans. Cascaded diffusion models for high fidelity image generation. *Journal of Machine Learning Research*, 23(47):1–33, 2022.
- [14] Jonathan Ho, Tim Salimans, Alexey Gritsenko, William Chan, Mohammad Norouzi, and David J Fleet. Video diffusion models. *Advances in Neural Information Processing Systems*, 35:8633–8646, 2022.
- [15] Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. *ICLR*, 1(2):3, 2022.
- [16] Zhihao Huang, Xi Qiu, Yukuo Ma, Yifu Zhou, Junjie Chen, Hongyuan Zhang, Chi Zhang, and Xuelong Li. Nfig: Autoregressive image generation with next-frequency prediction. *arXiv preprint arXiv:2503.07076*, 2025.
- [17] Siyu Jiao, Gengwei Zhang, Yinlong Qian, Jiancheng Huang, Yao Zhao, Humphrey Shi, Lin Ma, Yunchao Wei, and Zequn Jie. Flexvar: Flexible visual autoregressive modeling without residual prediction. *arXiv preprint arXiv:2502.20313*, 2025.

- [18] Yang Jin, Zhicheng Sun, Ningyuan Li, Kun Xu, Hao Jiang, Nan Zhuang, Quzhe Huang, Yang Song, Yadong Mu, and Zhouchen Lin. Pyramidal flow matching for efficient video generative modeling. *arXiv preprint arXiv:2410.05954*, 2024.
- [19] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *Advances in neural information processing systems*, 35: 26565–26577, 2022.
- [20] Tero Karras, Miika Aittala, Jaakko Lehtinen, Janne Hellsten, Timo Aila, and Samuli Laine. Analyzing and improving the training dynamics of diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 24174–24184, 2024.
- [21] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [22] Theodoros Kouzelis, Ioannis Kakogeorgiou, Spyros Gidaris, and Nikos Komodakis. Eq-vae: Equivariance regularized latent space for improved generative image modeling. *arXiv preprint arXiv:2502.09509*, 2025.
- [23] Black Forest Labs. Flux, 2024. URL <https://bfl.ai/>.
- [24] Doyup Lee, Chiheon Kim, Saehoon Kim, Minsu Cho, and Wook-Shin Han. Autoregressive image generation using residual quantization. In *CVPR*, 2022.
- [25] Zhimin Li, Jianwei Zhang, Qin Lin, Jiangfeng Xiong, Yanxin Long, Xinchi Deng, Yingfang Zhang, Xingchao Liu, Minbin Huang, Zedong Xiao, Dayou Chen, Jiajun He, Jiahao Li, Wenyue Li, Chen Zhang, Rongwei Quan, Jianxiang Lu, Jiabin Huang, Xiaoyan Yuan, Xiaoxiao Zheng, Yixuan Li, Jihong Zhang, Chao Zhang, Meng Chen, Jie Liu, Zheng Fang, Weiyang Wang, Jinbao Xue, Yangyu Tao, Jianchen Zhu, Kai Liu, Sihuan Lin, Yifu Sun, Yun Li, Dongdong Wang, Mingtao Chen, Zhichao Hu, Xiao Xiao, Yan Chen, Yuhong Liu, Wei Liu, Di Wang, Yong Yang, Jie Jiang, and Qinglin Lu. Hunyuan-dit: A powerful multi-resolution diffusion transformer with fine-grained chinese understanding. *arXiv*, 2024.
- [26] Yaron Lipman, Ricky TQ Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. *arXiv preprint arXiv:2210.02747*, 2022.
- [27] Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *ICLR*, 2023.
- [28] Guoqing Ma, Haoyang Huang, Kun Yan, Liangyu Chen, Nan Duan, Shengming Yin, Changyi Wan, Ranchen Ming, Xiaoniu Song, Xing Chen, et al. Step-video-t2v technical report: The practice, challenges, and future of video foundation model. *arXiv preprint arXiv:2502.10248*, 2025.
- [29] Willi Menapace, Aliaksandr Siarohin, Ivan Skorokhodov, Ekaterina Deyneka, Tsai-Shien Chen, Anil Kag, Yuwei Fang, Aleksei Stolar, Elisa Ricci, Jian Ren, et al. Snap video: Scaled spatiotemporal transformers for text-to-video synthesis. In *CVPR*, 2024.
- [30] Mang Ning, Mingxiao Li, Jianlin Su, Haozhe Jia, Lanmiao Liu, Martin Beneš, Wenshuo Chen, Albert Ali Salah, and Itir Onal Ertugrul. Dctdiff: Intriguing properties of image generative modeling in the dct space. *arXiv preprint arXiv:2412.15032*, 2024.
- [31] Aäron van den Oord, Nal Kalchbrenner, Oriol Vinyals, Lasse Espeholt, Alex Graves, and Koray Kavukcuoglu. Conditional image generation with pixelcnn decoders. In *NIPS*, 2016.
- [32] Maxime Oquab, Timothée Darzet, Théo Moutakanni, Huy V. Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel HAZIZA, Francisco Massa, Alaaeldin El-Nouby, Mido Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Herve Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.
- [33] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

- [34] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *ICML*, 2021.
- [35] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *Advances in neural information processing systems*, 32, 2019.
- [36] Severi Rissanen, Markus Heinonen, and Arno Solin. Generative modelling with inverse heat dissipation. In *The Eleventh International Conference on Learning Representations*, 2023.
- [37] Chitwan Saharia, William Chan, Saurabh Saxena, Lala Li, Jay Whang, Emily L Denton, Kamyar Ghasemipour, Raphael Gontijo Lopes, Burcu Karagol Ayan, Tim Salimans, et al. Photorealistic text-to-image diffusion models with deep language understanding. In *NeurIPS*, 2022.
- [38] Ivan Skorokhodov, Sharath Girish, Benran Hu, Willi Menapace, Yanyu Li, Rameen Abdal, Sergey Tulyakov, and Aliaksandr Siarohin. Improving the diffusability of autoencoders. In *ICML*, 2025.
- [39] George Stein, Jesse Cresswell, Rasa Hosseinzadeh, Yi Sui, Brendan Ross, Valentin Villecroze, Zhaoyan Liu, Anthony L Caterini, Eric Taylor, and Gabriel Loaiza-Ganem. Exposing flaws of generative model evaluation metrics and their unfair treatment of diffusion models. *Advances in Neural Information Processing Systems*, 36:3732–3784, 2023.
- [40] Jianlin Su, Murtadha Ahmed, Yu Lu, Shengfeng Pan, Wen Bo, and Yunfeng Liu. Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 2024.
- [41] Jiayan Teng, Wendi Zheng, Ming Ding, Wenyi Hong, Jianqiao Wangni, Zhuoyi Yang, and Jie Tang. Relay diffusion: Unifying diffusion process across resolutions for image synthesis. *arXiv preprint arXiv:2309.03350*, 2023.
- [42] Keyu Tian, Yi Jiang, Zehuan Yuan, BINGYUE PENG, and Liwei Wang. Visual autoregressive modeling: Scalable image generation via next-scale prediction. In *NeurIPS*, 2024.
- [43] Thomas Unterthiner, Sjoerd Van Steenkiste, Karol Kurach, Raphael Marinier, Marcin Michalski, and Sylvain Gelly. Towards accurate generative models of video: A new metric & challenges. *arXiv preprint arXiv:1812.01717*, 2018.
- [44] Mahtab Alizadeh Vandchali, Anastasios Kyrillidis, et al. One rank at a time: Cascading error dynamics in sequential learning. *arXiv preprint arXiv:2505.22602*, 2025.
- [45] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *NeurIPS*, 2017.
- [46] Team Wan, Ang Wang, Baole Ai, Bin Wen, Chaojie Mao, Chen-Wei Xie, Di Chen, Feiwu Yu, Haiming Zhao, Jianxiao Yang, et al. Wan: Open and advanced large-scale video generative models. *arXiv preprint arXiv:2503.20314*, 2025.
- [47] Haiyang Wang, Yue Fan, Muhammad Ferjad Naeem, Yongqin Xian, Jan Eric Lenssen, Liwei Wang, Federico Tombari, and Bernt Schiele. Tokenformer: Rethinking transformer scaling with tokenized model parameters. *arXiv preprint arXiv:2410.23168*, 2024.
- [48] Zhuoyi Yang, Jiayan Teng, Wendi Zheng, Ming Ding, Shiyu Huang, Jiazheng Xu, Yuanming Yang, Wenyi Hong, Xiaohan Zhang, Guanyu Feng, et al. Cogvideox: Text-to-video diffusion models with an expert transformer. *arXiv preprint arXiv:2408.06072*, 2024.
- [49] Jiahui Yu, Xin Li, Jing Yu Koh, Han Zhang, Ruoming Pang, James Qin, Alexander Ku, Yuanzhong Xu, Jason Baldridge, and Yonghui Wu. Vector-quantized image modeling with improved VQGAN. In *ICLR*, 2022.
- [50] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfai Yang, Burcu Karagol Ayan, Ben Hutchinson, Wei Han, Zarana Parekh, Xin Li, Han Zhang, Jason Baldridge, and Yonghui Wu. Scaling autoregressive models for content-rich text-to-image generation. *Transactions on Machine Learning Research*, 2022.

Appendix

Appendix Contents

A Framework Details	14
A.1 DFM Implementation Details	14
A.2 DFM Training Details	14
A.3 DFM Dataset Details	15
A.4 Inference Details	15
A.5 Choosing Training and Inference Hyperparameter	15
B Baseline Details.	15
C Additional Evaluation Results and Details	16
D Autoencoders without Scale Equivariance	17
E Large Scale Finetuning	17
E.1 FLUX DFM Finetuning	18
F Failed Experiments	18
G Limitations	19

A Framework Details

A.1 DFM Implementation Details

We base our architecture on a modified version of DiT [33]. Specifically, we normalize the data across scales by applying scale-wise pre- and postconditioning following the scheme of [19], which ensures that each input and output distribution has unit variance in expectation. Additionally, we replace ViT frequency-based positional embeddings applied in DiT [33] with 2D rotary positional embeddings (RoPE) [40] due to its wide adaptation in recent models [23, 46]. Furthermore, we condition the model on the stage being generated by adding an embedding of the current stage index to the modulation signal. Finally, we drop the class label conditioning 10% of the time to enable classifier-free guidance.

To adapt the DiT [33] for video generation, we patchify the latent frames independently and apply tokenwise concatenation of the resulting tokens [46]. Furthermore, we replace the 2D RoPE with 3D RoPE to adapt the position information to the video input. Finally, our ablations on Kinetics-700 reveal that unlike the FLUX autoencoder (See Table. 1 (c)), Cogvideo latents benefit from standardizing the input to have unit variance before the application of the diffusion process. Therefore, we apply such standardization to all of our video experiments.

A.2 DFM Training Details

We train using the Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-8}$) and a base learning rate of 0.0001 with 10k linear warmup steps, weight decay of 0.01, and total batchsize of 256. The main experiments are trained with DiT-XL/2 on ImageNet-1K for 512px and 1024px, respectively, for 500k and 350k steps. For the main video experiments on Kinetics-700, we train for 200k steps. Ablations are trained with DiT-B/2 for 600k steps using the same hyperparameters as the main experiments.

ImageNet-1K 512px experiments are trained on a single node containing 8 H100 GPUs, whereas ImageNet-1K 1024px and Kinetics-700 512px experiments used 2 nodes of the same type. Ablations were trained on a single node.

A.3 DFM Dataset Details

Our main image experiments are trained and evaluated on ImageNet-1K [5] which has a research-only, non-commercial license. Our video experiments are trained and evaluated on Kinetics-700 [3], which is available under the Creative Commons Attribution 4.0 (CC BY 4.0) license.

A.4 Inference Details

For all of our experiments, unless otherwise specified, we use an Euler ODE sampler with 40 steps and a linear timestep scheduler.

A.5 Choosing Training and Inference Hyperparameter

DFM introduces several training and inference hyperparameters. In practice, a default configuration of such hyperparameters generalizes well across different datasets and settings. In the following, we discuss such optimal configurations and summarize the observed effects of varying the main hyperparameters.

- **Number of generation stages:** Using 2 stages works well for most experiments, where the first stage has a resolution of 256px, while the second stage has a resolution equal to the final resolution (see Table. 1). We find that a base resolution of 256px for the first stage contains an ideal amount of structural detail to support generation of the successive ones while not containing excessive amounts of fine-grained details.
- **First stage sampling probability p_t^0 :** We find that 0.9 generalizes well across different model sizes. The probability determines the amount of model capacity spent on structural detail modeling, so smaller models may benefit from higher values (see Table. 1)
- **Stage noise sampling parameters:** At training time, we use a logit normal distribution with parameter (*location*=0, *scale*=1.0) for the stage currently sampled for training, and (*location*=1.5, *scale*=1.0) for the previous stage. Larger location values for previous stages allow the model to leverage more structural details as conditioning during second stage generation, but expose the model more to train-inference mismatches if the first stage results are not generated with sufficient quality. Thus, larger location values are more beneficial for larger models (see Table. 1).
- **Sampling stage threshold τ :** 0.9 generalizes well across different settings (see Figure 6). As the parameter determines the amount of information in the first stage that the second stage can leverage as conditioning, in the presence of a high-quality first stage prediction, larger thresholds are desirable. While the optimal value varies depending on cfg, model, and dataset, we find the suggested value to be a reliable default.
- **Input standardization:** Before applying noise, we standardize each level in the decomposed input representation to have unit-variance for Kinetics-700 video experiments using the CogVideo autoencoder, while this behavior is disabled for ImageNet-1K image generation experiments with the FLUX autoencoder. Standardization has an impact on loss magnitude for each stage, thus it acts similarly to the first stage sampling probability p_t^0 in balancing the amount of model capacity allocated to modeling structural details or fine details. We found that the optimal parameters can vary depending on the autoencoder representation and we suggest ablating over this setting when adopting a new autoencoder.

B Baseline Details.

This section, includes details about the baselines training and inference. First, the base architecture is described, then specific details about the Cascaded and Pyramidal Flow baselines are detailed.

Table 4: DFM with original (orig) and scale-equivariant (SE) FLUX autoencoder. Experiments are performed on DiT-B and trained on ImageNet-1K [5] 512px for 400k steps.

Method	$\text{FID}_{50\text{K}} \downarrow$	$\text{FDD}_{50\text{K}} \downarrow$	$\text{IS} \uparrow$
Flow Matching (FLUX-ae-orig)	54.50	855.5	21.4
DFM (FLUX-ae-orig)	34.00	630.1	34.2
Flow Matching (FLUX-ae-SE)	43.16	728.85	26.6
DFM (FLUX-ae-SE)	32.89	626.5	35.0

Baselines architecture. All baselines use the hyperparameters and architecture of [33], with the modification to incorporate rotary positional embeddings (RoPE) [40] and Network Preconditioning [19]. Inference is performed with 40 steps using an Euler sampler and a linear sampling schedule.

Cascaded baseline. We initially train the stage 1 model following the spatial baseline for $\approx 80\%$ of the total training steps. To match the training and inference compute of the other baselines, we adopt a patch size of 1×1 , yielding the same number of tokens as the other baselines. Subsequently, we finetune the stage 1 model to obtain the stage 2 model, which upsamples the first stage output. For the stage 2 model, we introduce a dedicated pacification layer for the conditioning input from stage 1 and concatenate its output tokens with those in stage 2. We add a small amount of noise to the conditioning input during training to reduce exposure bias. The amount of noise is sampled from a logit normal distribution with scale parameters of 1.0 and location of 1.5. For inference, we perform a grid search over the best inference parameters and found that equally dividing the inference steps between stage 1 and stage 2 (*i. e.* 20 inference steps for each stage) and using a noise level of 0.025 yields the best results. Therefore, we use these settings for our evaluations.

Pyramidal Flow baseline. We follow Jin *et al* [18] in training the Pyramidal Flow baseline. For the three-stages experiments, we allocate twice the batch size for the second stage compared with the first and second stage following [18] and use a gamma parameter of 0.33. In the two-stages experiments, we allocate an equal batch size between the first and second stages. During inference, we use an equal number of inference stapes for each stage.

C Additional Evaluation Results and Details

Sampling parameters ablation We report quantitative results ablating the role of different sampling hyperparameters in Figure 6 and show qualitative results in Figure 18 and Figure 19 ablating, respectively, sampling steps allocated to each stage, and different cfg and sampling threshold configurations. Increasing the number of steps allocated to the first stage results in improved image structure, while increasing the number of steps allocated to the second stage results in reduced gains which are observable in areas with complex textures such as foliage, grass, or animal fur. The sampling threshold has a reduced impact on generation quality and is most visible at low cfg values.

Comparison to baselines Convergence behavior for DFM and baselines is shown in Figure 8, Figure 9, and Figure 10, respectively, for ImageNet-1K [5] 512px, ImageNet-1K [5] 1024px, and Kinetics-700 [3]. In addition, we show qualitative comparison on non-curated samples for DFM against baselines on ImageNet-1K [5] 512px (see Figure 11), ImageNet-1K [5] 1024px (see Figure 12), and Kinetics-700 [3] (see Figure 13, Figure 14, and Figure 15). Addtional qualitative results and videos are provided in the *Website*.

FLUX-DFM qualitative comparison with FLUX-FT. We provide additional qualitative comparison of FLUX finetuned with DFM (FLUX-DFM) against FLUX with standard finetuning applied (FLUX-FT) on 1024px text-to-image generation in Figure 16 and Figure 17.

Qualitative results. We provide qualitative results on ImageNet-1K [5] 512px of selected classes in Figure 20, Figure 21, and Figure 22. Addtionally, we provide fully uncurated samples in Figure 23, Figure 24, and Figure 25.

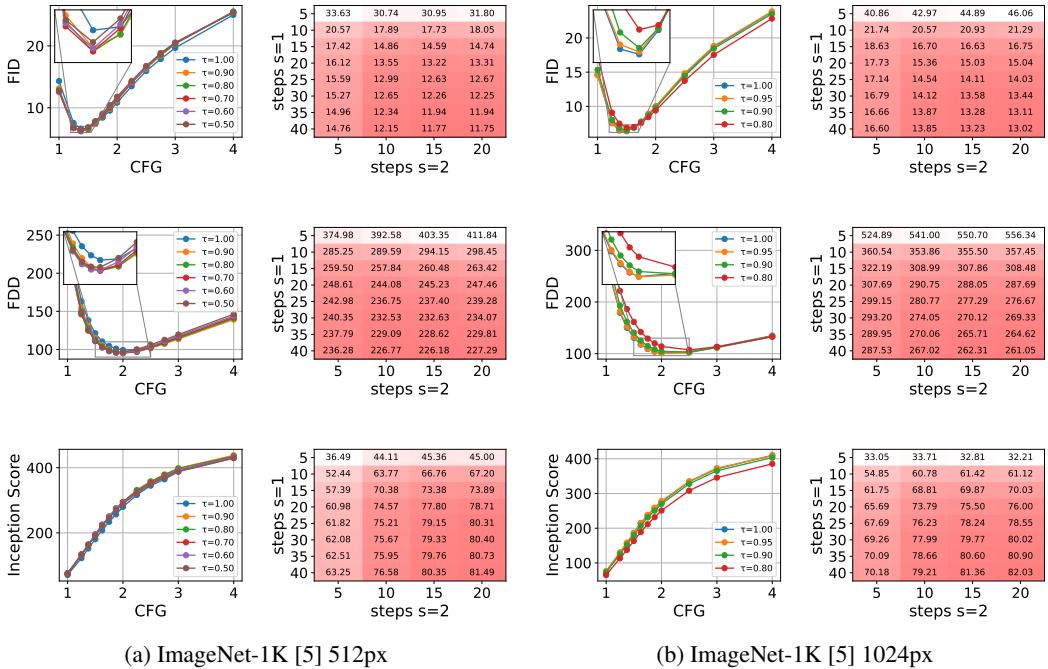


Figure 6: FID_{10K} , FDD_{10K} and IS abating sampling configuration, threshold, and per-stage steps on DiT-XL/2 trained with DFM on ImageNet-1K [5].

D Autoencoders without Scale Equivariance

DFM explicitly decouples low- and high-frequency content into two successive stages. The first stage diffuses coarse, low-frequency information. Its output then conditions the generation of high-frequency details in the second stage. Recent work shows that diffusion models follow this spectral autoregressive pattern implicitly [30, 36, 38] and proposed scale-equivariant autoencoders to improve this separation, making the produced latents easier to diffuse [38]. Although DFM is agnostic to the choice of autoencoder, we conduct our experiments with the autoencoders of [38] finetuned for scale equivariance to ensure desirable spectral properties both for baselines and DFM.

In this section, we explore the behavior of DFM on non-regularized autoencoders. Table 4 reports a comparison between Flow Matching and DFM on the original FLUX autoencoder and scale-equivariant FLUX autoencoder for a DiT-B architecture trained for 400k steps on ImageNet-1K [5]. When the scale-equivariant FLUX autoencoder is replaced with the original variant, Flow Matching suffers a substantial drop in performance, whereas DFM preserves a similar performance, thanks to its explicit spectral decomposition. In both cases, DFM outperforms Flow Matching.

E Large Scale Finetuning

This section explores the usage of DFM for fine-tuning large-scale models. Since the stage 1 input resembles a low-resolution version of the original input, we aim to preserve the pretrained model’s behavior as much as possible. DFM adds a per-stage patchification layer, timestep embedding, and output head. Accordingly, we reuse the pretrained patchification layer and timestep embedding for stage 1, while zero-initialising the corresponding stage 2 modules. Before training, the framework therefore replicates the pretrained model’s low-resolution predictions. After training, all weights (including patchification layers) are adapted to generate consistent stage 1 and stage 2 outputs. To adjust the model patchification layers weight to stage 1, which uses a smaller patch size, we upsample stage 1 input by a factor of 2 before adding noise, halving the effective patch size. We then downsample the model’s output to match stage 1 resolution.

E.1 FLUX DFM Finetuning

We choose to fine-tune FLUX-DEV [23] due to its strong performance. To avoid biases introduced by cfg distillation, the distillation-guidance factor is fixed at 1.0 throughout training, and all fine-tuning is carried on an internal dataset. We follow [38] to regularize the the FLUX autoencoder, yielding a *scale-equivariant* autoencoder.

Then, we perform full-finetuning of FLUX for 24k steps to adjust it to the new autoencoder, producing FLUX-SE. During the initial 4k training steps, we freeze all layers except the patchification layers. Then, we finetune FLUX-SE for 32k steps to obtain FLUX-DFM. As a baseline, we finetune FLUX-SE for the same 32k steps using standard full-finetuning. Finetuning uses 1024-px and 512-px images at variable aspect ratios. We use Adam optimizer ($\beta_1 = 0.9$, $\beta_2 = 0.99$, $\epsilon = 10^{-8}$) and a base learning rate of 0.00001 with 2k linear warmup steps, weight decay of 0.01, and total batch size of 192. We drop the text conditioning 10% of the time to enable classifier-free guidance.

Please note that since FLUX-DEV is distilled and post-trained on highly aesthetic images, its distribution differs from that of our internal data. Therefore, a direct comparison with the original FLUX-DEV would not be informative. Rather, we measure speed in learning the new training distribution with DFM compared with standard full-finetuning at an equal training cost.

Inference. We evaluate using a test split of 10k prompts from our internal dataset. We use 40 sampling steps and Euler ODE solver, cfg of 3.0, and a distillation guidance factor of 1.0 (equivalent to not applying cfg).

F Failed Experiments

DCT-Space DFM We apply DFM in the DCT space, where the input visual modality is represented in the frequency domain rather than in the spatial domain. We obtain the DCT decomposition by first dividing the visual input into 2D or 3D blocks of pixels of size 4 or 8 and applying DCT to each of them. Different stages are formulated as the progressive modeling of DCT components of increased frequency. We find, however, that DiT models provide reduced performance when working in the frequency rather than the spatial domain, a finding we speculate may originate from an increased difficulty of leveraging token similarities in attention operations. While experiments showed that DCT can be used as a way of producing the input decomposition, input to the DiT model should be converted to the spatial domain for optimal performance, and the Laplacian decomposition is preferred due to its simplicity.

Alternative parameter specialization methods As parameter specialization showed the capability of increasing the model’s performance (see Table. 1 (d)), we investigate alternative ways of specializing model parameters per each stage. As an alternative avenue, we investigate the recent Tokenformer [47] architecture due to its capacity to progressively integrate new parameters during training. In particular, we instantiate a shared set of parameters and a set of specialized parameters for each stage. The shared set of parameters is always active, while the specialized parameters are activated only when corresponding to the currently active stage. We find the Tokenformer [47] architecture to underperform the regular DiT design when given the same amount of computation, thus we abort this experiment. To reduce the number of parameters required for specialization, we wrap each linear layer in different LoRA wrappers [15, 44], one for each stage, and activate the respective wrapper when the corresponding stage is selected at training or inference. LoRA showed improved results over the baseline when utilizing high LoRA ranks which resulted in similar parameter counts to full parameter specialization.

Expert Models We train separate or *expert* models for each stage rather than a joint model. We then evaluate performance by mixing expert models trained for different numbers of steps. We find that performance improves steadily with more training of the first stage and improves marginally with more training with longer training of the second stage model, supporting the intuition that modeling of structural detail has a larger importance to sample quality. To reduce the computational burden of training separate models for each stage, we explore finetuning the second stage model starting from the first stage model with positive results. Despite positive results, the idea is not explored further due to the increased complexity of maintaining separate models for each stage. We note that, in principle, it is possible to obtain expert models by finetuning a jointly trained model separately for each stage with full or LoRA finetuning. We leave the exploration of this possibility as future work.



Figure 7: (left) Selected samples highlighting failure cases generated by our framework trained on ImageNet-1K [5] 512px on DiT-XL for 1.3M iterations. When generating high-frequency details such as fine structures, vegetation or cluttered environments, DFM may produce artifacts. (right) Artifacts can be mitigated by increasing the number of sampling steps for the second stage.

G Limitations

DFM improves modeling of visual inputs by decomposing them into different components. The framework’s hyperparameters control the amount of model capacity dedicated to modeling each of the components. As an example, a larger probability of sampling stage 0 (p_t^0) during training (see Table. 1 (a)) results in a larger emphasis on structural details. As shown in Figure 7, selected samples containing large amounts of high-frequency components such as vegetation, fur, thin structures, or cluttered environments may exhibit artifacts in such regions which manifest as a flattened appearance. Increasing the number of sampling steps for the second stage (see Figure 7 (right) and Figure 18) mitigates such artifacts. By acting on training sampling probabilities for each stage, and distribution of sampling steps between different stages, the framework allows for balance between structural and fine details modeling quality.

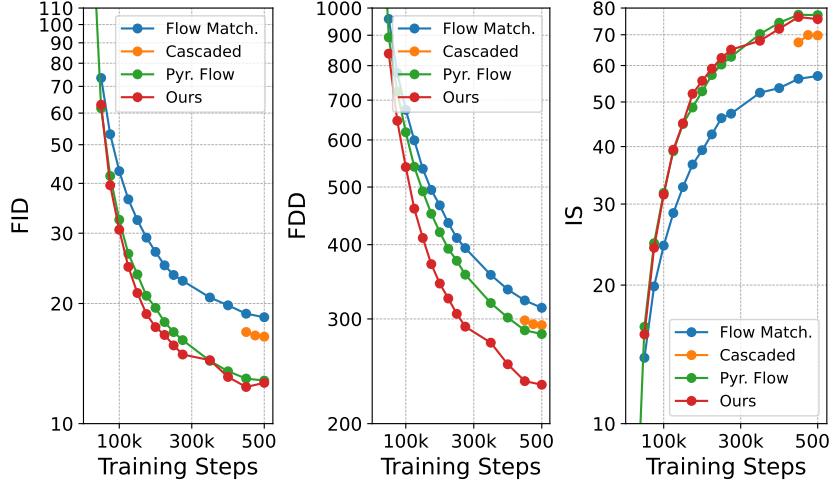


Figure 8: Convergence curves for different datasets and metrics comparing our framework to baselines on ImageNet-1K [5] 512px.

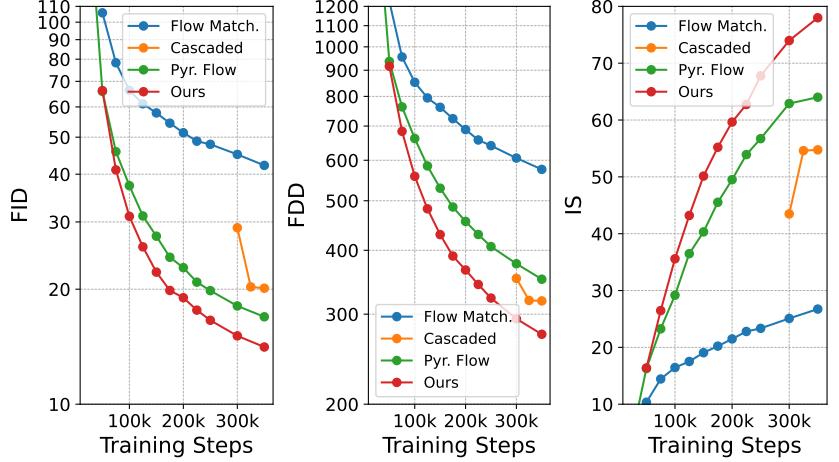


Figure 9: Convergence curves for different datasets and metrics comparing our framework to baselines on ImageNet-1K [5] 1024px.

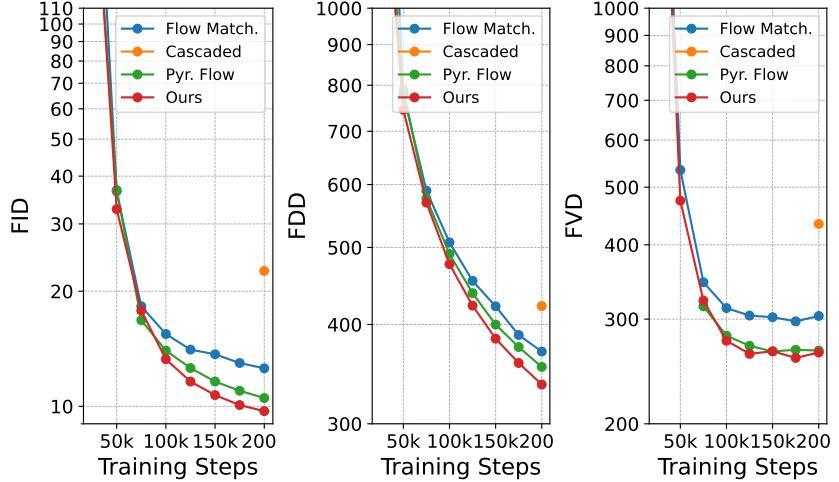


Figure 10: Convergence curves for different datasets and metrics comparing our framework to baselines on Kinetics-700 [3] 512px.



Figure 11: Comparison of DFM against baselines on DiT-XL trained on ImageNet-1K 512px [5] for 500k steps. Samples are fully uncurated and generated with cfg 3.0.



Figure 12: Comparison of DFM against baselines on DiT-XL trained on ImageNet-1K 1024px [5] for 350k steps. Samples are fully uncurated and generated with cfg 3.0.

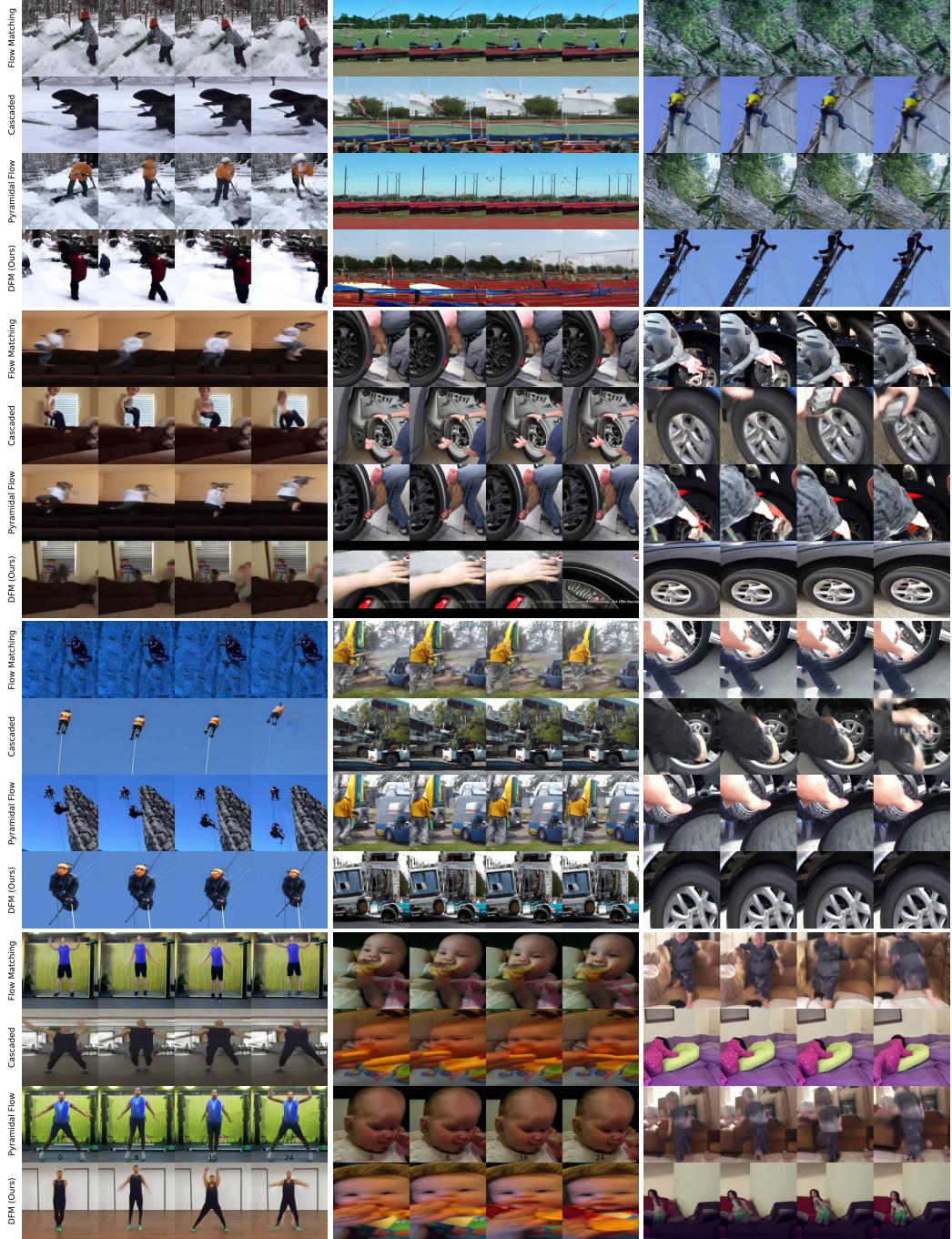


Figure 13: Comparison of DFM against baselines on DiT-XL trained on Kinetics-700 [3] 512px for 200k steps. Samples are fully uncurated and generated with cfg 3.0.

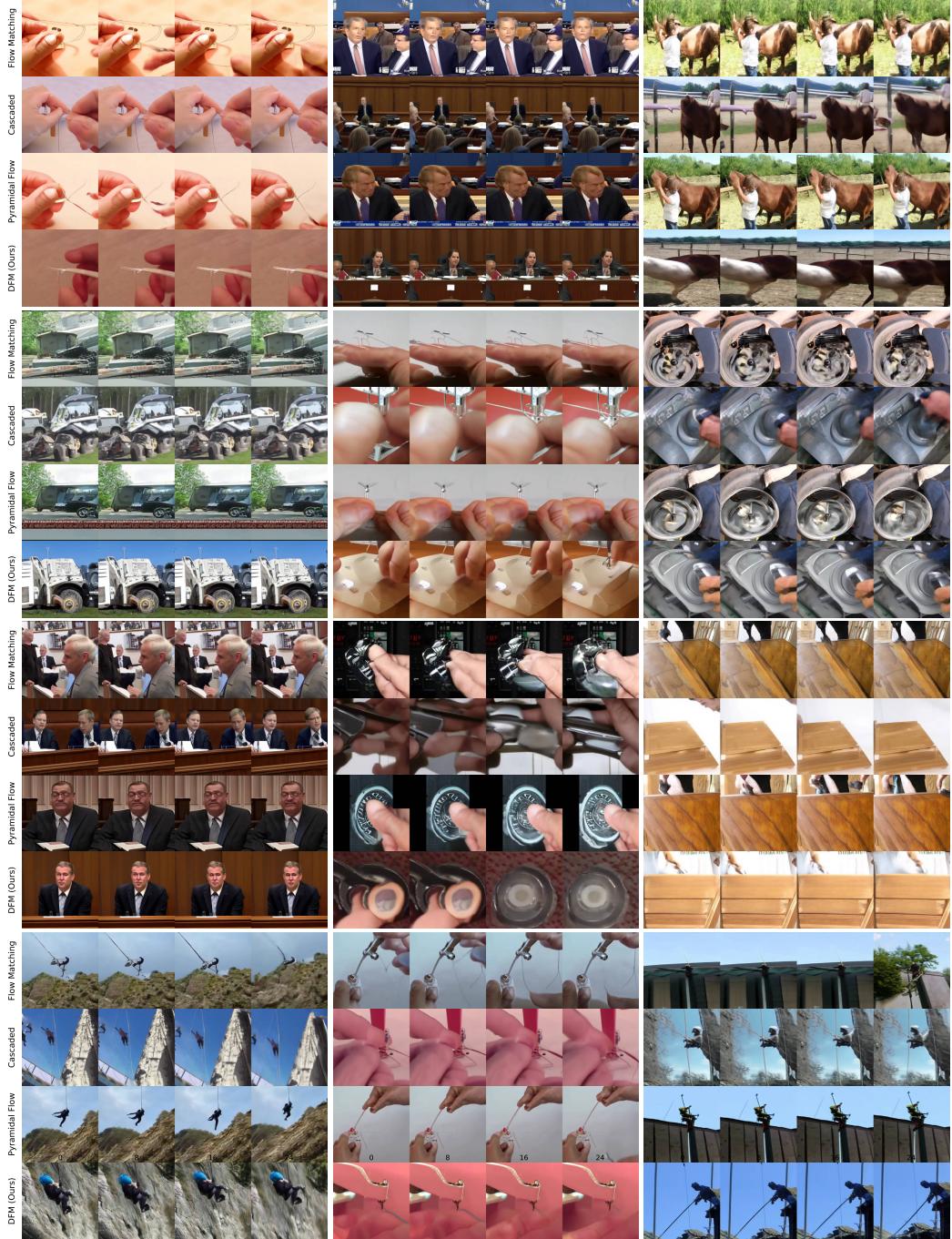


Figure 14: Comparison of DFM against baselines on DiT-XL trained on Kinetics-700 [3] 512px for 200k steps. Samples are fully uncurated and generated with cfg 3.0.

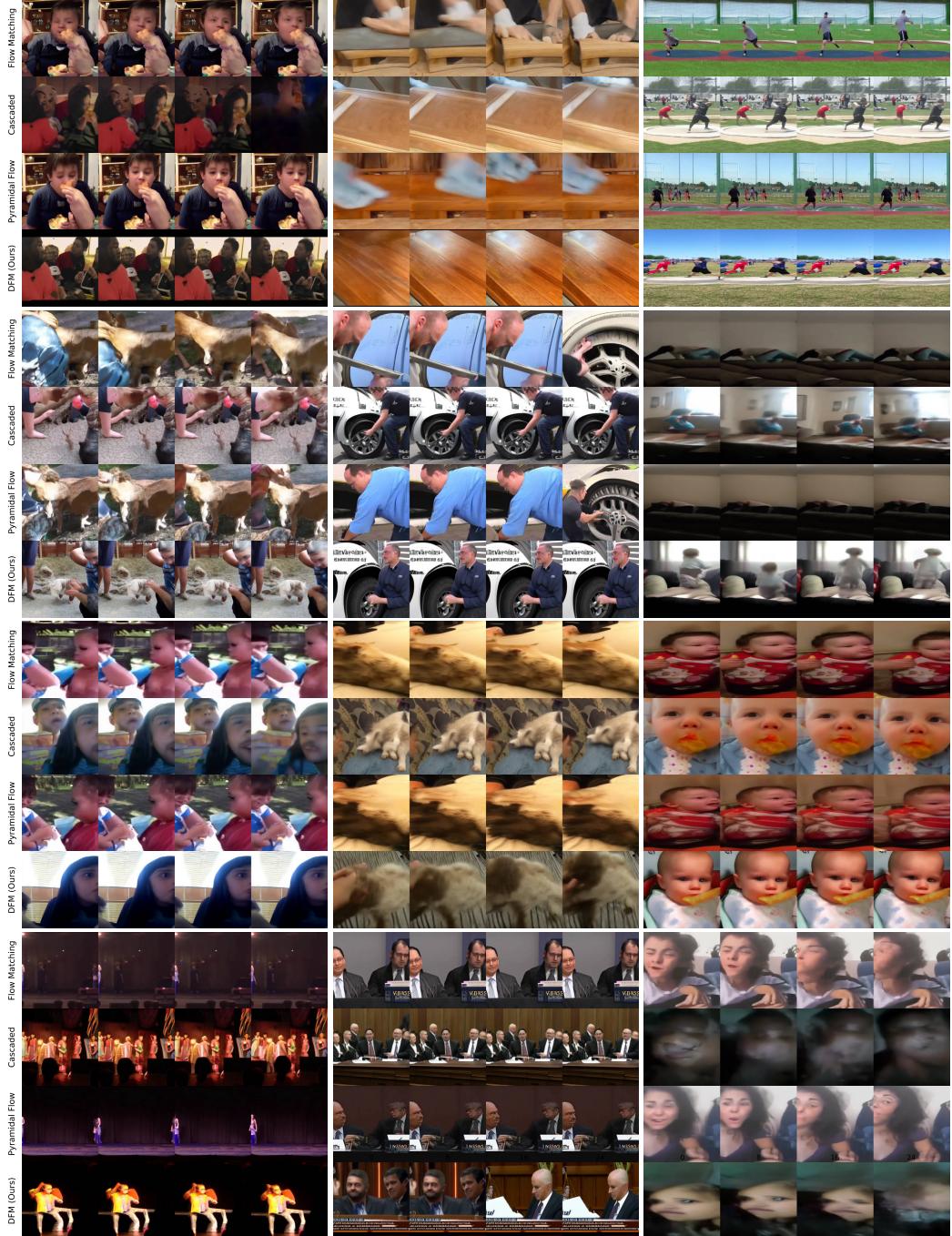


Figure 15: Comparison of DFM against baselines on DiT-XL trained on Kinetics-700 [3] 512px for 200k steps. Samples are fully uncurated and generated with cfg 3.0.

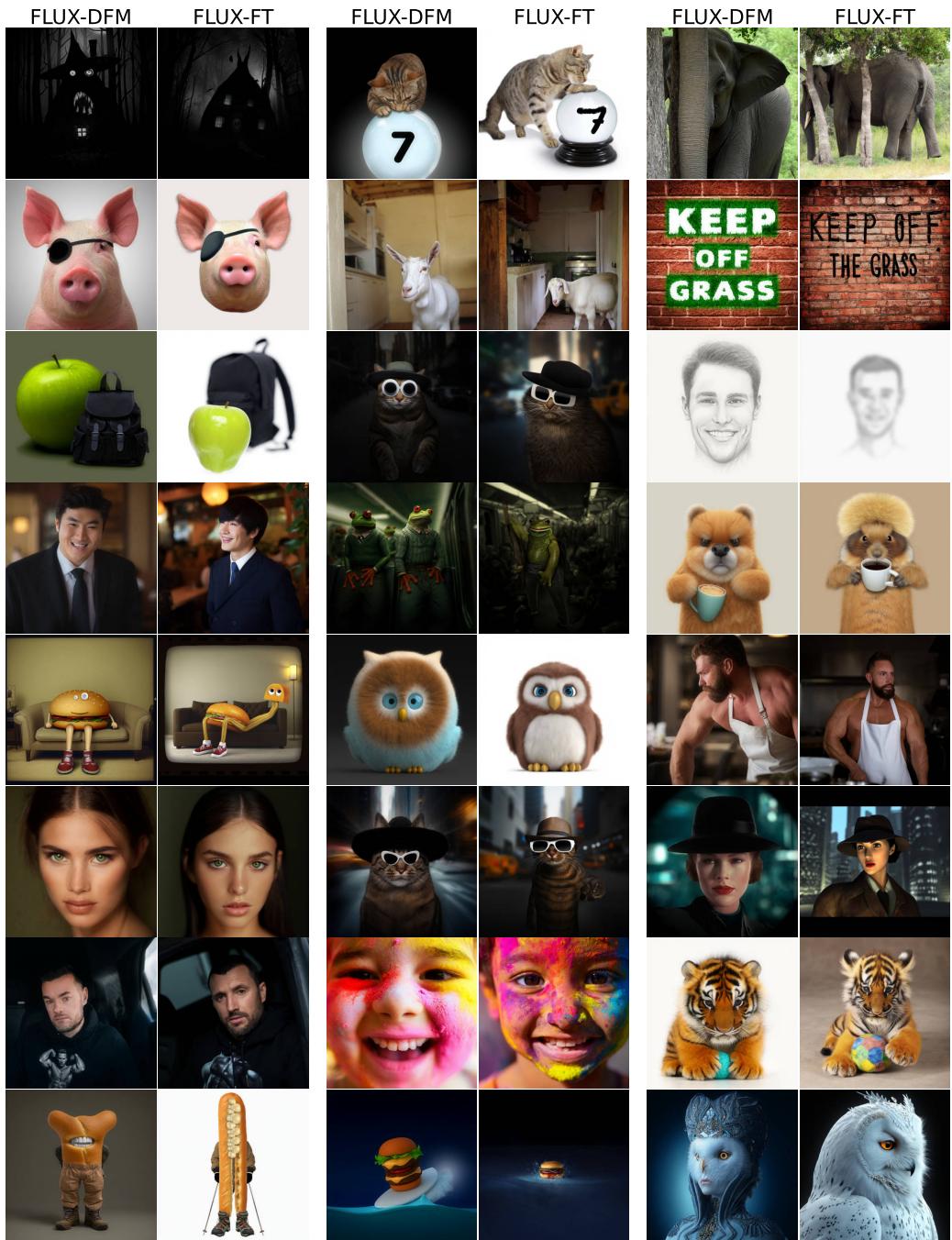


Figure 16: Comparison of finetuning FLUX-DEV with DFM against standard full finetuning trained finetuned for 24k steps. Samples are generated with cfg 4.5.

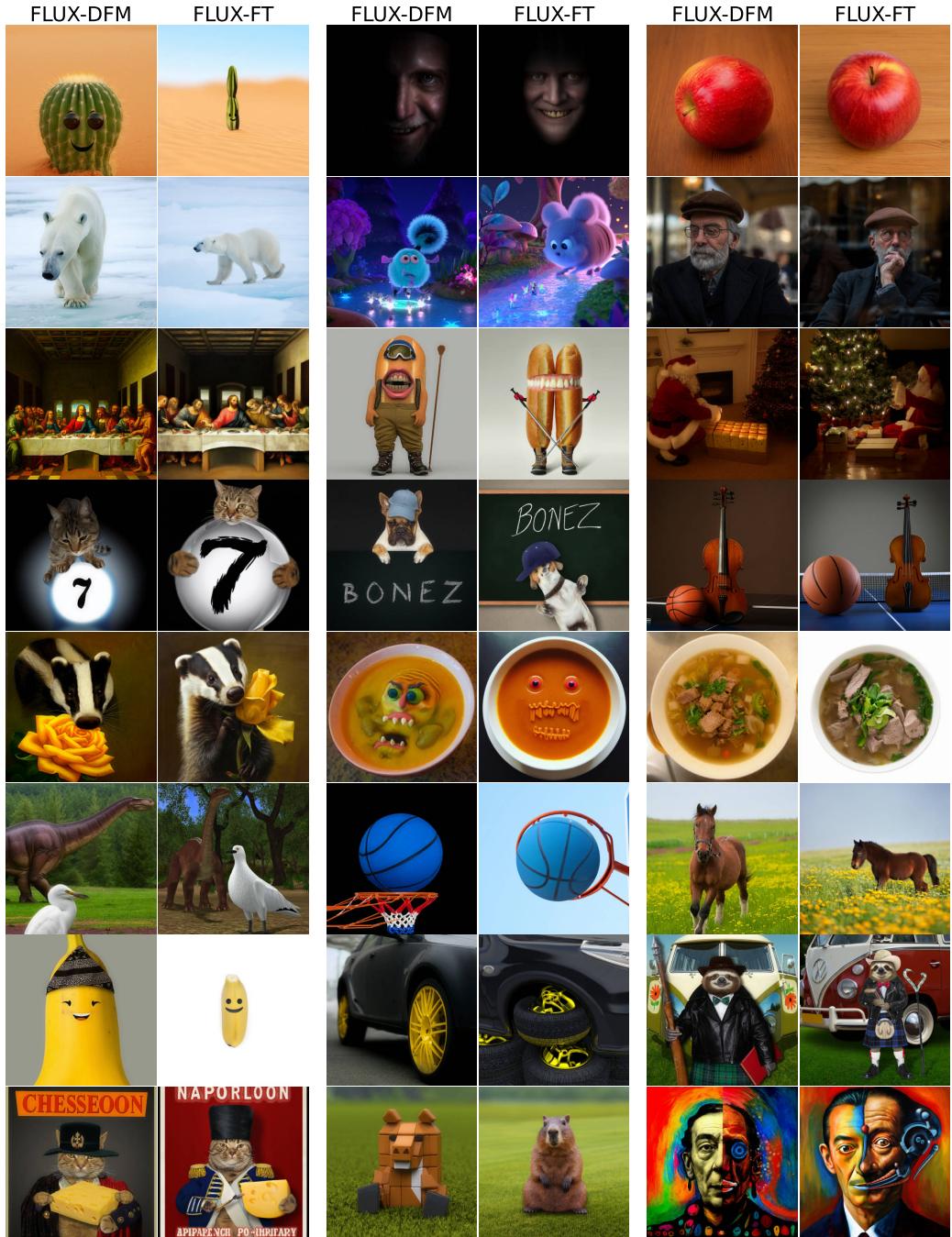


Figure 17: Comparison of finetuning FLUX-DEV with DFM against standard full finetuning trained finetuned for 24k steps. Samples are generated with cfg 4.5.

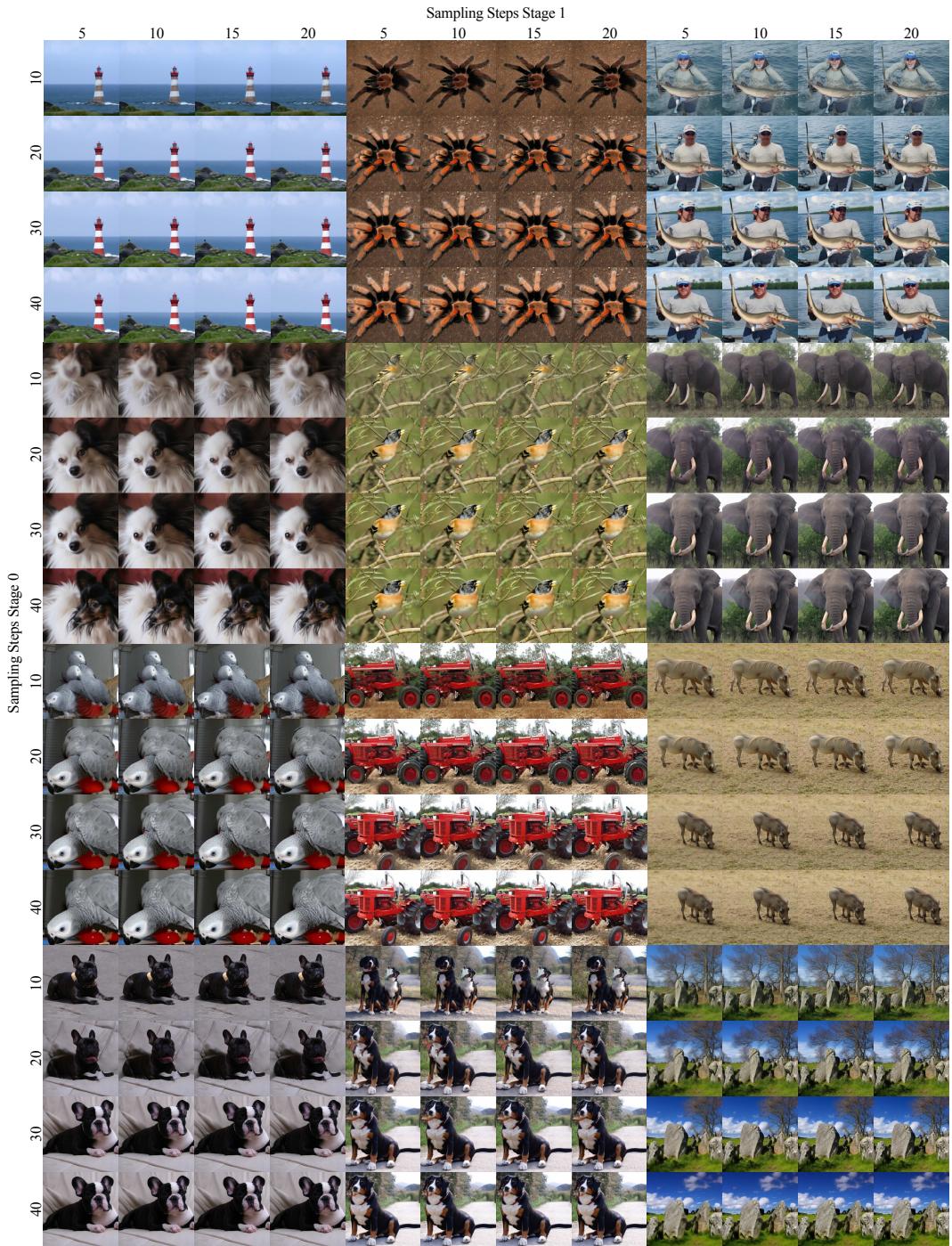


Figure 18: Ablation of per-stage sampling steps on DiT-XL trained on ImageNet-1K 512px [5] for 500k steps. Samples are selected to highlight the effects of varying sampling parameters. Samples are generated with cfg 3.0.



Figure 19: Ablation of the effect of cfg values and sampling threshold τ on DiT-XL trained on ImageNet-1K 512px [5] for 500k steps. Samples are selected to highlight the effects of varying sampling parameters. Samples are generated with cfg 3.0.



Figure 20: Qualitative results from selected classes on ImageNet-1K [5] 512px produced by DiT-XL trained with DFM for 1.3M steps. We use 30 and 10 sampling steps respectively for the first and second stages and a cfg value of 3.0.



Figure 21: Qualitative results from selected classes on ImageNet-1K [5] 512px produced by DiT-XL trained with DFM for 1.3M steps. We use 30 and 10 sampling steps respectively for the first and second stages and a cfg value of 3.0.



Figure 22: Qualitative results from selected classes on ImageNet-1K [5] 512px produced by DiT-XL trained with DFM for 1.3M steps. We use 30 and 10 sampling steps respectively for the first and second stages and a cfg value of 3.0.

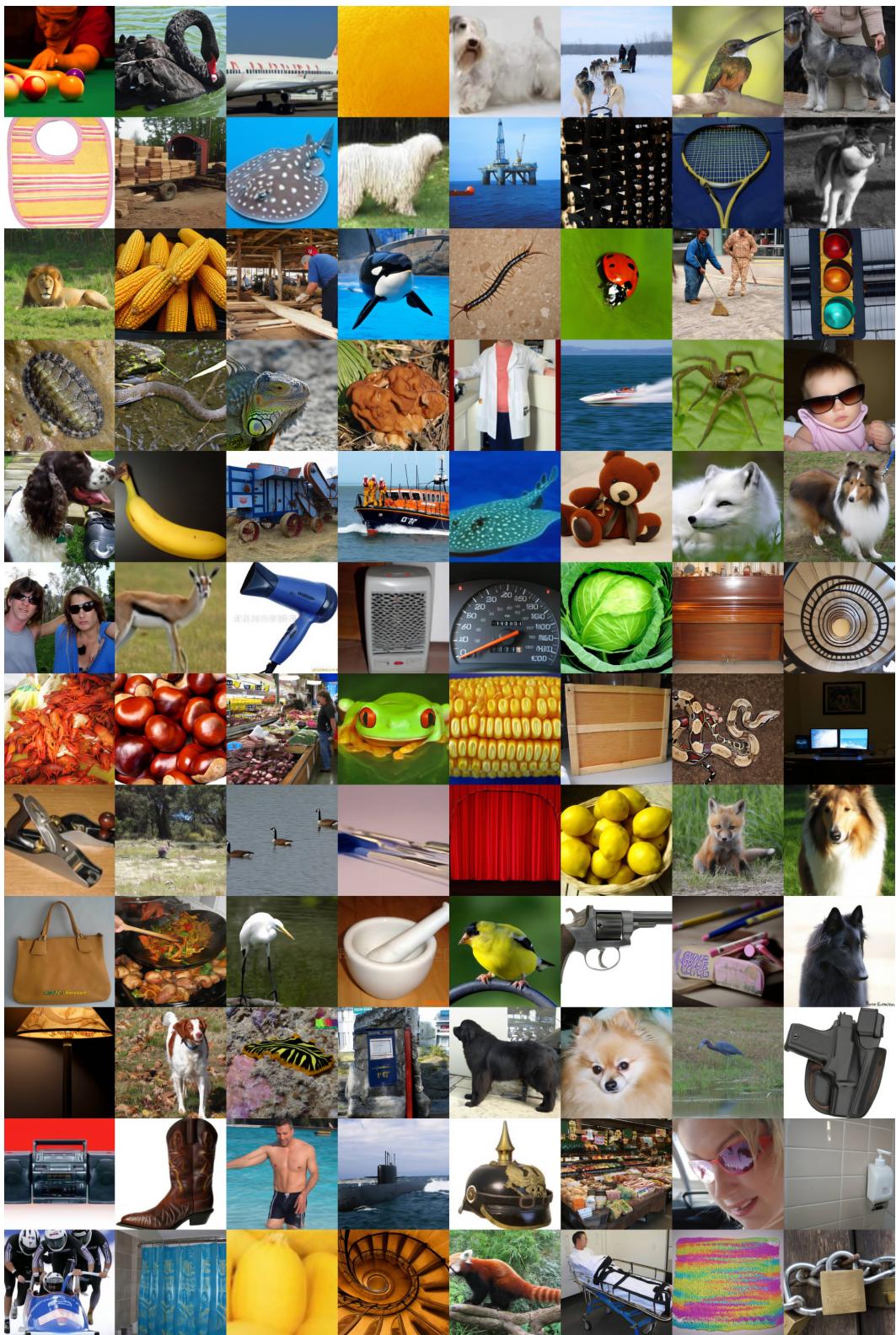


Figure 23: Fully uncurated samples from ImageNet-1K [5] 512px produced by DiT-XL trained with DFM for 1.3M steps. We use 30 and 10 sampling steps respectively for the first and second stages and a cfg value of 3.0.

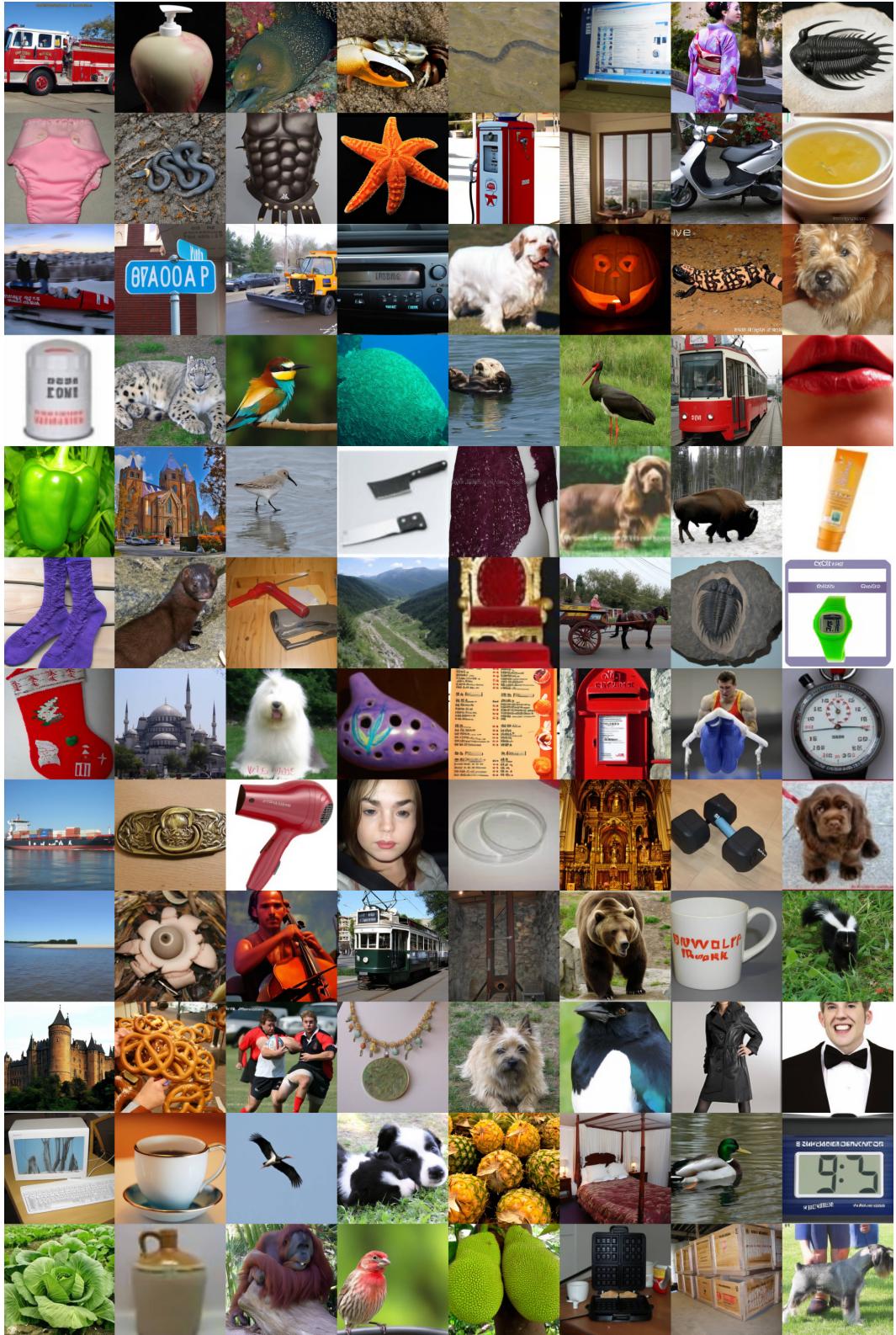


Figure 24: Fully uncurated samples from ImageNet-1K [5] 512px produced by DiT-XL trained with DFM for 1.3M steps. We use 30 and 10 sampling steps respectively for the first and second stages and a cfg value of 3.0.

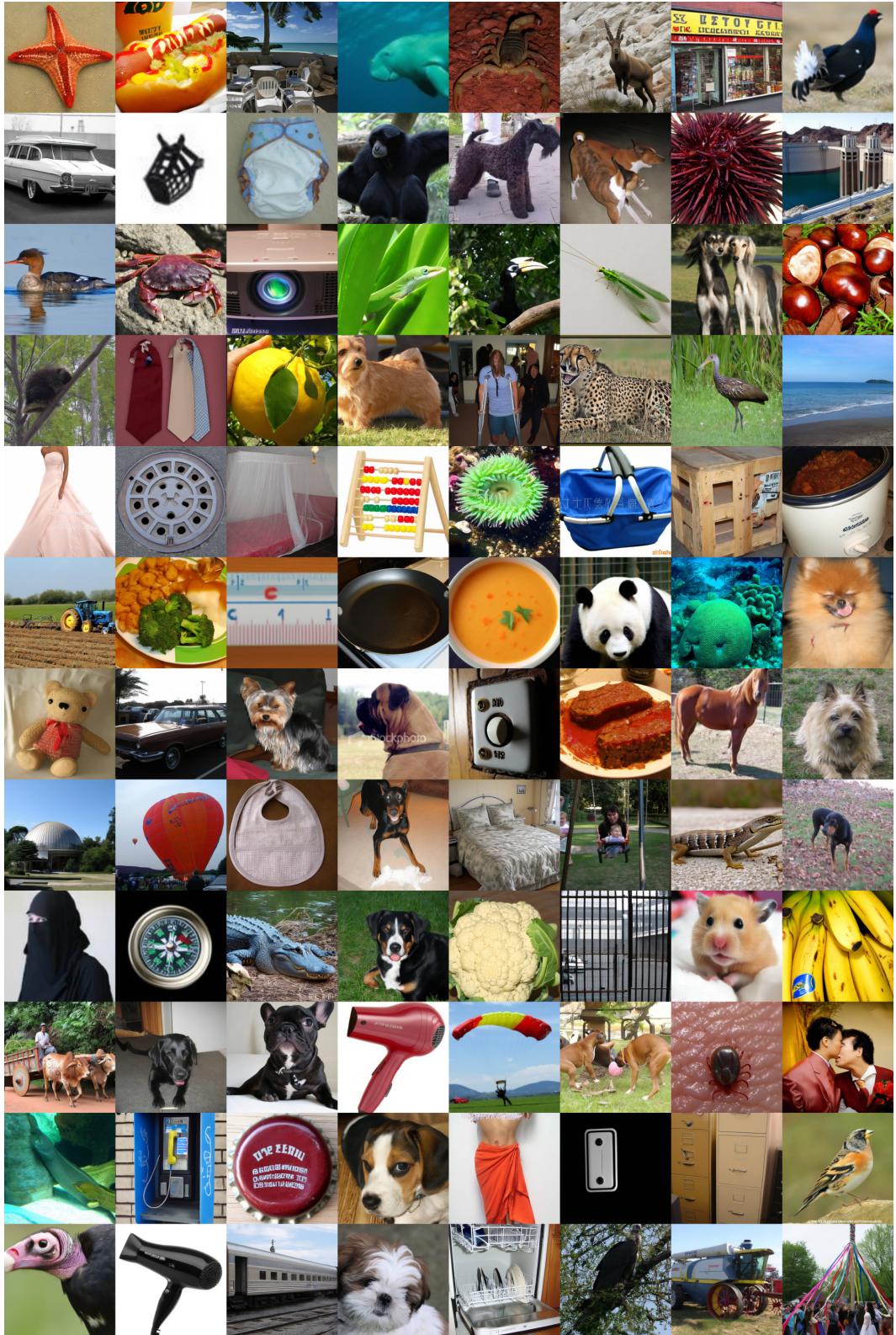


Figure 25: Fully uncurated samples from ImageNet-1K [5] 512px produced by DiT-XL trained with DFM for 1.3M steps. We use 30 and 10 sampling steps respectively for the first and second stages and a cfg value of 3.0.