A. Detailed Related Work

Efficiency in Vision Transformers [26] falls primarily into two broad categories: reducing the computation per token or the number of tokens overall.

Reducing the amount of computation per token. Reducing the amount of computation per token can be achieved by reducing the model size when training via distillation [8] or pruning the network after training [47, 114]. These methods again though typically work with a static inference protocol. As in [112], we compare in Fig. 10 our class-conditioned *ImageNet FlexiDiT* model, against popular based pruning techniques, based on Diff pruning [31], Taylor, magnitude or random pruning [48]. As one can see, our dynamic scheduler outperforms these baselines. *We note that our method can also be*

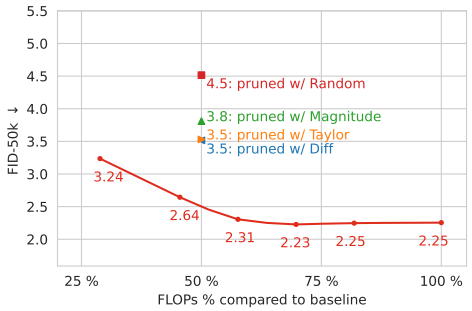


Figure 10. We compare our dynamic scheduler with more baselines.

applied in conjunction with pruning techniques to achieve even higher efficiency gains, offering potential orthogonal benefits. In concurrent work, [112] propose to adjust, in a dynamic way, the model during inference in different steps. In our work, we do not train separate models for each target FLOPs ratio like they do, meaning that we can train a single model and decide how many FLOPs we want to invest during inference. This makes our approach more versatile. Additionally, our training is very stable, and no specific training tricks are required to converge successfully. That is how we were able to extend experiments to high-resolution image and video generation, achieving significantly better speed-ups than the ones they reported. We also outperform their results when the compute budget is very small. Nonetheless, this work could inspire adaptive per-sample schedulers, that could open new future directions.

Given the large computational requirements of the attention operation, many methods nowadays focus on that to reduce the overhead imposed. These methods commonly use some form of hierarchical attention [34, 66], skip (usually the first) attention layers altogether [102], or reduce the number of the attended keys [16, 30, 108], by commonly aggregating keys in a spatial neighborhood or applying some form of windowed attention.

Reducing number of tokens. Our method primarily falls within the second category of reducing the overall number of tokens. Previous work here, typically relied on filtering [64, 82, 101], merging [12, 44, 69] or dropping [2]. Although merging works well for applications that eventually lead to some pooling operations (like classification tasks or for the task of creating an image-specific embedding), it works significantly less well for applications that require dense (token-level) predictions, where some un-merging operation has to be defined. In other concurrent work, [97] reduces the number of representative tokens to calculate the attention over. Our approach resembles most [9], where vision Transformers are trained to handle inputs with varying patch resolution. By applying less compute for some steps, we can reduce computational complexity significantly, without a drop in performance.

Image generation. In the context of image generation, diffusion has been largely established as the method for attaining state-of-the-art results. There have been previous works that try to take advantage of potential correlations between successive denoising step predictions [86], by either caching intermediate results in the activations [71, 100], or in the attention [113]. Caching has the advantage of a training-free method. Nonetheless, potential benefits are lower. Similar to our work, [5] use different experts for different denoising steps. Instead of using different experts that require separate training and separate deployment, we show how a single model can be easily formed into a flexible one that can be instantiated in different modes, with each of its modes corresponding essentially to a different expert. Similar in-spirit approaches have been proposed that rely on the smaller compute requirements for lower resolution image generation [54, 111]. [50] also adapt the computation per step, by projecting into smaller subspaces. We instead, keep the dimension of the latent space and the characteristics

of it the same across diffusion steps. Orthogonal gains to our approach are also possible through methods such as guidance distillation [56, 74] and consistency models [89]. Our approach is also largely agnostic to the diffusion process and can be applied out of the box for flow matching methods [61]. We point the interested reader to [73] for a survey for further efficiency in diffusion models. Finally, compared to other established techniques [35, 63] we do not fundamentally change the architecture, which allows us to apply our framework effortlessly for numerous pre-trained models across different modalities.

Video generation. Our approach can be easily extended for video generation, and in principle for the generation of any modality where some inductive bias (spatial, temporal, etc) is employed in the diffusion (latent) space. In video generation, typically, latent video tokens are processed in parallel [11, 65, 79]. Training-free methods [51, 62, 113] have been proposed in this case to accelerate video generation. Benefits with training-free methods are nonetheless minimal before performance degradation kicks in (see Table 1 in [51], where one can typically save less than 30%).

An interesting direction for future work involves adapting the inference scheduler, i.e. what patch size we are using for each denoising step, based on the requirements of each sample. It is natural to assume that when generating more static videos, increasing the temporal patch size, and thus decreasing the amount of compute along the temporal dimension, will result in smaller drops in performance. The same holds for the spatial patch sizes when generating images or videos that require less high-frequency details.

B. Additional Experiments and Details

We provide additional experiments, complementary to the main text.

B.1. Exposure Bias and Accumulation of Error

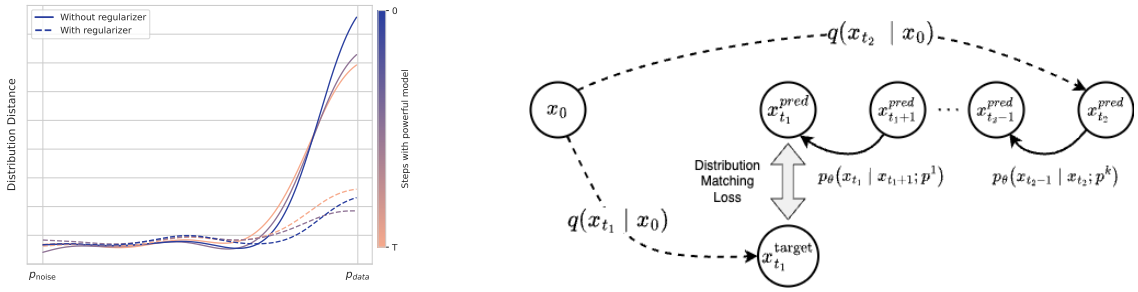


Figure 11. **Left:** We use maximum mean discrepancy to estimate the distribution mismatch between $p_\theta(\mathbf{x}_t|\mathbf{x}_{T:t+1})$ and $q(\mathbf{x}_t|\mathbf{x}_0)$. **Right:** The proposed bootstrapped loss. During training, we perform a few denoising steps with a weak model followed by a few denoising steps with a powerful model (reminiscent of the scheduler during inference) and apply a distribution matching loss on the resulting samples.

Inference with diffusion suffers from *exposure bias*, due to the discrepancy of the input distribution during training and inference [21, 57, 58, 76]. Briefly, models are trained to denoise images sampled from the $q(\mathbf{x}_t|\mathbf{x}_0)$ distribution [40]. Inference on the other hand is characterized by repeated model evaluations $p_\theta(\mathbf{x}_t|\mathbf{x}_{t+1})$ and any distribution mismatch between $p_\theta(\mathbf{x}_t|\mathbf{x}_{T:t+1})$ and $q(\mathbf{x}_t|\mathbf{x}_0)$ accumulates, as also shown in Fig. 11 (left). The error at each iteration depends on the model, with a perfect model resulting in 0 error and thus no error accumulated. In our case, the accumulation of error is exacerbated by the characteristics of our model, where weak models could lead to higher, but also specific in nature, kinds of errors. Training with the standard denoising objective, where real samples are randomly noised for some t, does not make the more powerful model aware of the nature of the mistakes made by the weak model, rendering it unable to potentially correct them. We propose to mitigate this issue by introducing a bootstrapped distribution matching loss [93], as illustrated in Fig. 11 (right). The loss is applied in a patch size-dependent manner, according to the desired inference protocol (from weak to powerful model calls during inference).

Given natural images $\mathbf{x}_0, \tilde{\mathbf{x}}_0 \sim q(\mathbf{x}_0)$, we sample two time points $t_1 > t_2$ and corrupt the images with noise $\mathbf{x}_{t_1}^{\text{target}} \sim q(\mathbf{x}_{t_1}|\mathbf{x}_0), \tilde{\mathbf{x}}_{t_2}^{\text{pred}} \sim q(\tilde{\mathbf{x}}_{t_2}|\tilde{\mathbf{x}}_0)$. We then apply a chain of denoising steps $\epsilon_\theta(\tilde{\mathbf{x}}_{t-1}^{\text{pred}}|\tilde{\mathbf{x}}_t^{\text{pred}}; p)$ for $t \in (t_1, t_2]$ and a patch size p. Ultimately, we wish for the distributions of $\mathbf{x}_{t_1}^{\text{target}}$ and $\tilde{\mathbf{x}}_{t_1}^{\text{pred}}$ to match, for which we employ the maximum mean discrepancy (MMD) [33]. To let the powerful model learn and potentially correct mistakes of the weak model and to simulate how our inference patch size scheduler works, we perform the first of these denoising steps with the weak model, followed by denoising steps with the powerful model. Given a set of patch sizes $\{p^i\}_{i=1}^k$ where $p^1 < p^2 < \cdots < p^k$ and a number of

denoising steps to perform with each $s^1, s^2, \ldots, s^k$, where $\sum_{j=1}^{k} s^j = t_2 - t_1$, we denoise with a given patch size p^i for all t's in $(t_1 + \sum_{j=i+1}^{k} s^j, t_1 + \sum_{j=i}^{k} s^j]$. An illustration of this process can also be seen in Fig. 11 (right). When sampling a time step t_1, we bias our sampling similar to [85]. The proposed distribution matching loss, inspired by the notion of consistency [88, 89], provides a principled way to correct the errors accumulated during inference. We note that we are optimizing a simple distribution matching loss, instead of over-optimizing according to desired downstream metrics (namely FID), thus not violating Goodhart's law[6]. Different distribution matching losses (including discriminator-based losses) can also be used. Note that we only correct this exposure bias for the class-conditioned image generation experiments, as this is the only case where we fine-tune the powerful pre-trained model.

B.2. Inference with Packing

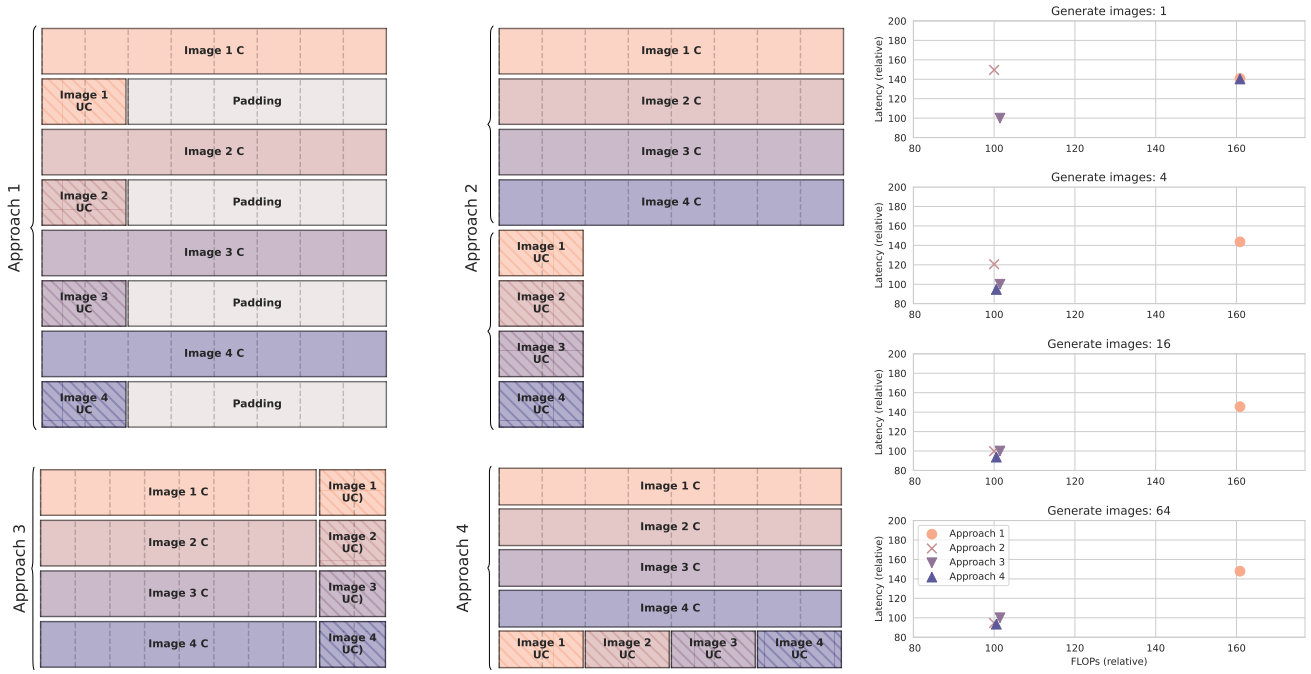


Figure 12. Different approaches can be employed to perform forward passes with CFG when the conditional (C) and unconditional (UC) predictions use different patch sizes. Here, each row corresponds to a sequence of tokens propagated through the DiT, and each bracket corresponds to a batch of sequences for a single NFE. Generally 'Approach 2' leads to the smallest amount of FLOPs, but for batch size 1, inference can be memory bound for low-resolution image generation. 'Approach 4' mostly leads to the smallest latency, as long as the number of generated images is larger than 4, i.e. the ratio of the sequence lengths between the powerful and the weak model. On the right, we plot FLOPs and Latency from the four different approaches of performing inference, for a different number of generated images. Batch size plays a role here (class-conditioned image generation experiments) as generated images are of lower resolution, namely 256×256, and thus sequence lengths through the Transformer are smaller. Normalized FLOPs are determined based on 'Approach 2' and normalized latency based on 'Approach 3'. We use *torch.compile* with *fullgraph=True* and *mode = 'reduce-overhead'*.

We provide more details in Fig. 12, on how to perform inference with CFG when the conditional and unconditional predictions employ a different patch size. We show this for our class-conditioned model, but results easily generalize for all our *FlexiDiT* models. Performing CFG entails NFEs with double the batch size (or 2 distinct NFEs), for the conditional and unconditional input, respectively. Performing the conditional and unconditional calls with different patch sizes leads to propagating sequences of different lengths through the DiT. Depending on how these sequences are 'packed' together, and for lack of a hardware-specific implementation of masked attention, more FLOPs can be traded for better latency. Our weak model additionally leads to memory benefits, which can be traded for a bigger batch size when serving the model. Notice that current state-of-the-art image generation models in practice require much longer sequences compared to the 256×256 images generated here (see also Section 4.4) and so generation is compute-bound even when generating with batch size equal to 1.

[6]Goodhart's law states that: 'When a measure becomes a target, it ceases to be a good measure'.

B.3. What does the Model Learn?

Transformers are composed of a series of channel mixing components — feed-forward layers with shared weight applied to all tokens in the sequence — and token mixing components — attention applied to tokens in the sequence. By coercing the model to learn the denoising objective when applied to images processed with different patch sizes, we are enforcing inductive bias in its weights and helping it better understand global structures in the image [27, 81, 102]. We test this hypothesis and evaluate what the model is learning in the following ways in Fig. 13. (left) We visualize using t-SNE, centered kernel alignment (CKA) between feature maps across layers when performing NFEs with different patch sizes. Activations across layers exhibit similar transformations [96], except the early layers, where features are lower level, i.e. more patch specific. (right) We visualize the Jensen–Shannon divergence (JSD) between attention maps (interpolated to the same image space) when performing NFEs with different patch sizes. We compare using our *FlexiDiT* model with different patch sizes (Flexible) versus using two static models trained with different patch sizes (namely *DiT-XL/2* and a trained from scratch *DiT-XL/4*). Our flexible model showcases lower JSD, demonstrating better knowledge transfer between the different patch sizes. We believe that this "transfer" of knowledge is crucial to (1) confirm that parameter sharing across patch sizes is valid and (2) ensure that fine-tuning can be fast and sample efficient.

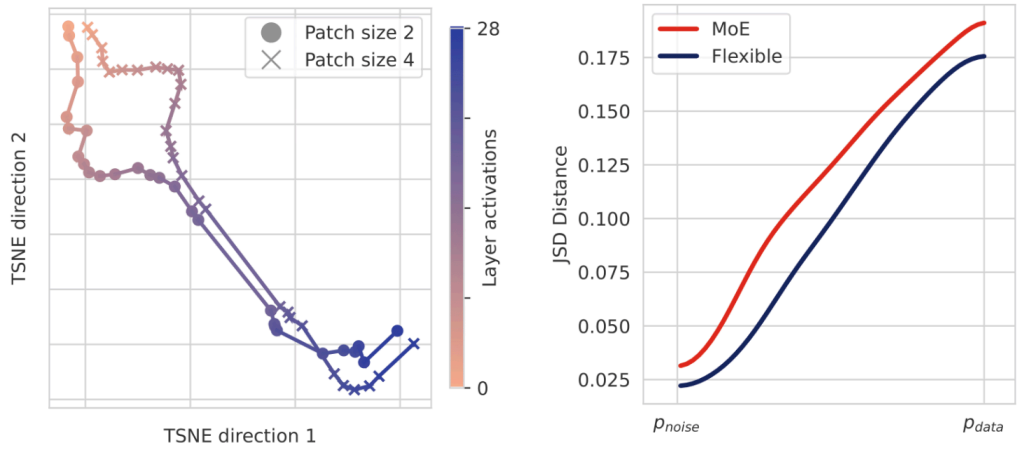


Figure 13. Interpretability of the model activations and attention scores, when propagating samples tokenized with different patch sizes.

B.4. Class-Conditioned Image Generation

Additional metrics. For class-conditioned experiments on the main text we focused on the *DiT-XL/2* [78] and FID as a metric. Here, we report more metrics apart from FID, namely Inception Score (IS), sFID, and Precision/Recall. Results are presented in Fig. 14 for our flexible *DiT-XL/2* model. We remind that for class-conditioned models, we fine-tune models using our distribution matching loss. As a result, the powerful model that we get after fine-tuning is different to the pre-trained checkpoints we start from. To verify that our weak model does not lead to less diverse samples, we embark on a small experimental study to guarantee the diversity of generated images. We follow [92] and generate images from the same label map. We then calculate pairwise similarity/distance between these images and average across all similarities/distances and all label maps. We use MS-SSIM [98], LPIPS [110] and plot results in Fig. 15. Results indicate very similar values in terms of the diversity of the generated images. We also provide some sample images demonstrating diversity from the baseline model in Fig. 16 and our tuned model in Fig. 17. Note that the diversity of the generated images is in general high and not affected much by using the weak model for more of the initial denoising steps.

Caching distance. In the main text, we have shown how weak and powerful models generate more similar predictions during the early steps of the denoising process. Previous papers to accelerate diffusion have largely relied on caching [17, 71, 100, 108, 113] previous activations, by taking advantage of the similarity in activations between successive steps. For completeness, we also plot the caching distance between activations of the same layer between successive generation steps in Fig. 18. In this paper, we do not employ caching but focus on an orthogonal approach. We advocate that all steps are important for high-quality image generation, as demonstrated by our experiments on reducing the overall number of generation steps.

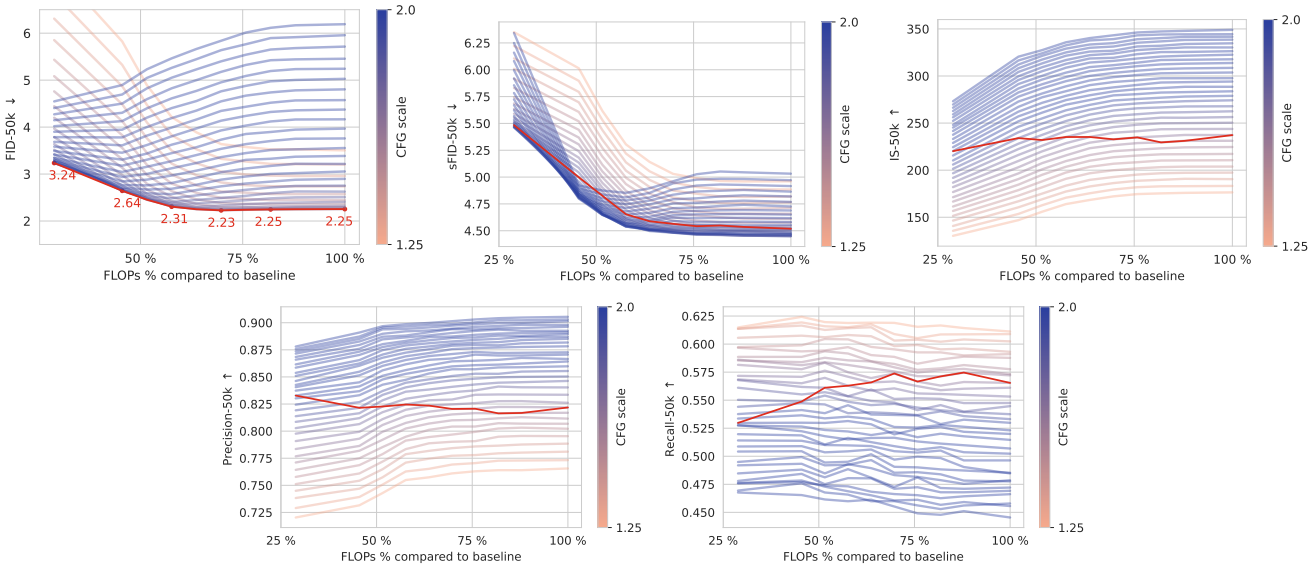


Figure 14. More metrics for our *FlexiDiT* based on the *DiT-XL/2* model for class-conditioned generation on *ImageNet*. We plot (a) FID (b) sFID, (c) inception score, (d) precision, and (e) recall when generating $50,000$ samples with 250 steps of the DDPM schedule for various values of the CFG scales. Red lines correspond to the values that lead to the optimum FID scores for each compute level.

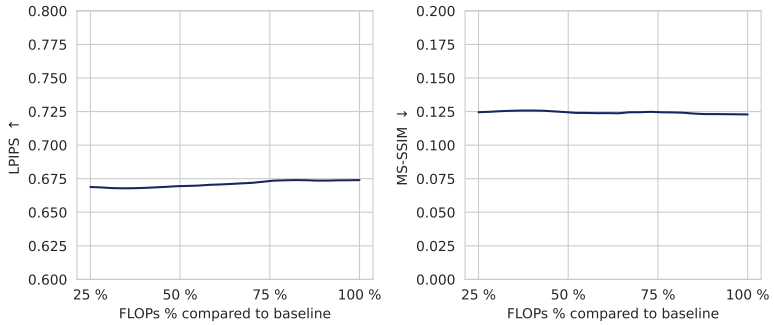


Figure 15. Average distance/similarity of images generated from the same label map. Both metrics take values between 0 and 1.

Instead of completely skipping steps, we simply invest less compute for them, and let the model decide how to allocate this compute.

Additional schedulers. Based on the results on activation distance between successive denoising steps (Fig. 18), one could argue that first denoising steps are also important and thus a better inference scheduler would deploy the powerful model for these as well. In practice, we found no benefit from deploying a scheduler that works like that. Notice though how activation distance is high for the first denoising steps only for some of the layers. We additionally experimented with dynamic schedulers that choose the patch size of each denoising step based on the activation distance of different layers between successive denoising steps. We did not find additional potential benefits.

In this paper, we are training a single model that can denoise images with any patch size for any denoising step. Given a fixed desired inference scheduler — i.e. if we know exactly which t's to run with the powerful and the weak model —, one can train a model specifically based on that, leading to undoubtedly better quality images for the same compute. Similar techniques are regularly applied in consistency models [89]. Finally, we compare our scheduler — performing the first T_{weak} denoising steps with a weak model — versus the opposite scheduler, i.e. performing the last T_{weak} denoising steps with a weak model. Results in Fig. 19 indicate that, as expected, using the weak model in the last diffusion steps is suboptimal, leading to a loss in fine-grained details. We also provide qualitative examples of how these different schedulers affect image quality in Fig. 20.

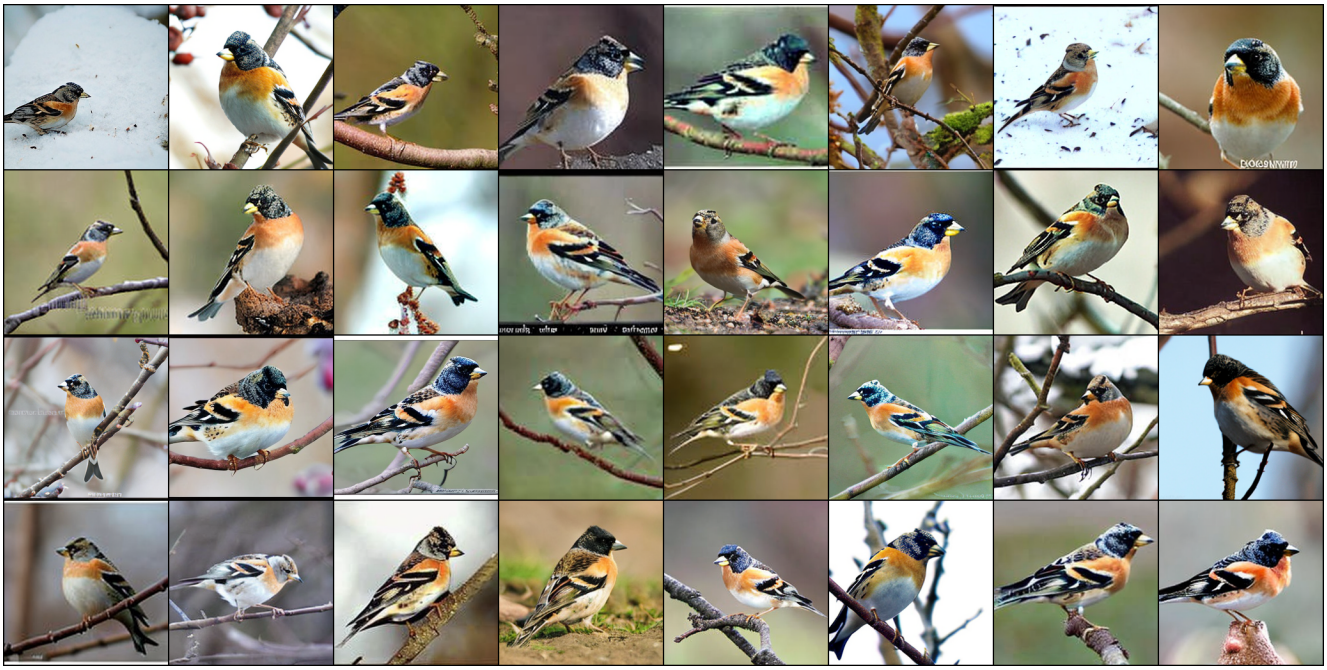


Figure 16. Sample images generated with the baseline *DiT-XL/2* for the *ImageNet* category 'Brambling'.

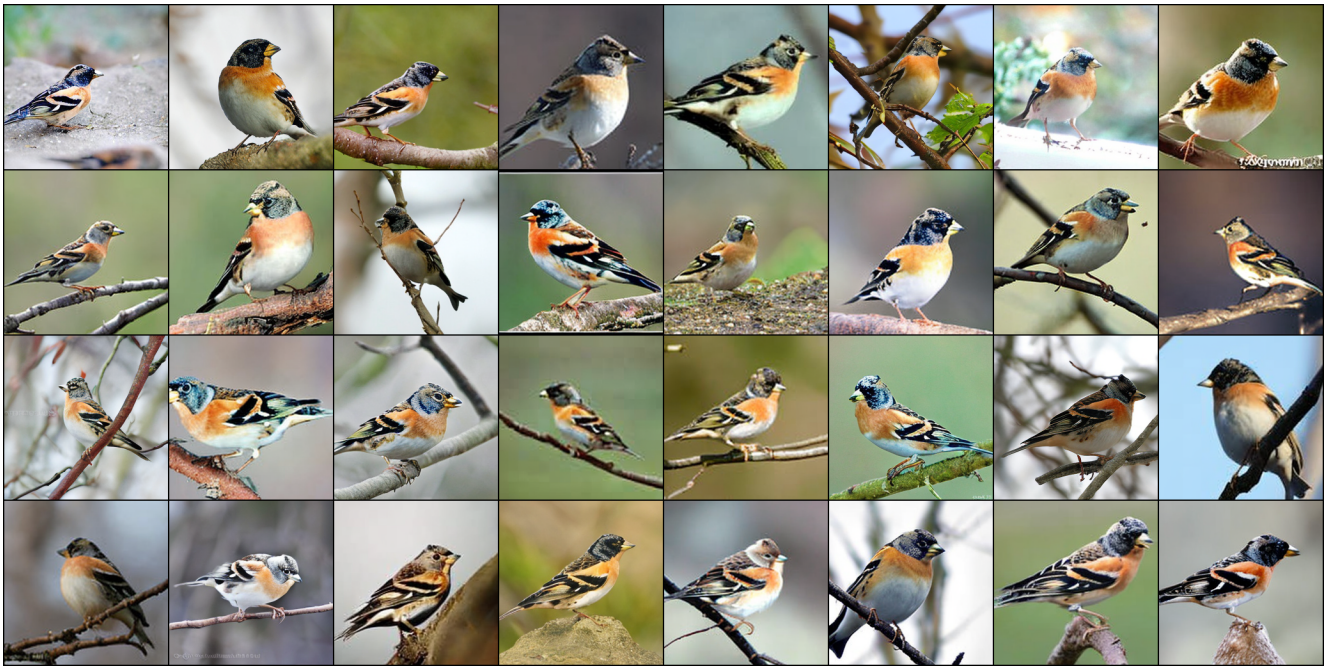


Figure 17. Sample images generated with our flexible *DiT-XL* model when performing inference using only the powerful model, for the *ImageNet* category 'Brambling'.

More results on CFG. In the main text, we presented results on performing inference with different CFG scales and different invocations to our weak model for the unconditional and conditional part. The 4 generated curves in Fig. 6 (middle) correspond to performing our scheduler as $250/250$, $130/130$, $70/70$, and $30/0$ where x/y means using the powerful model for the last x denoising steps for the conditional and y denoising steps for the unconditional part. When performing CFG, we

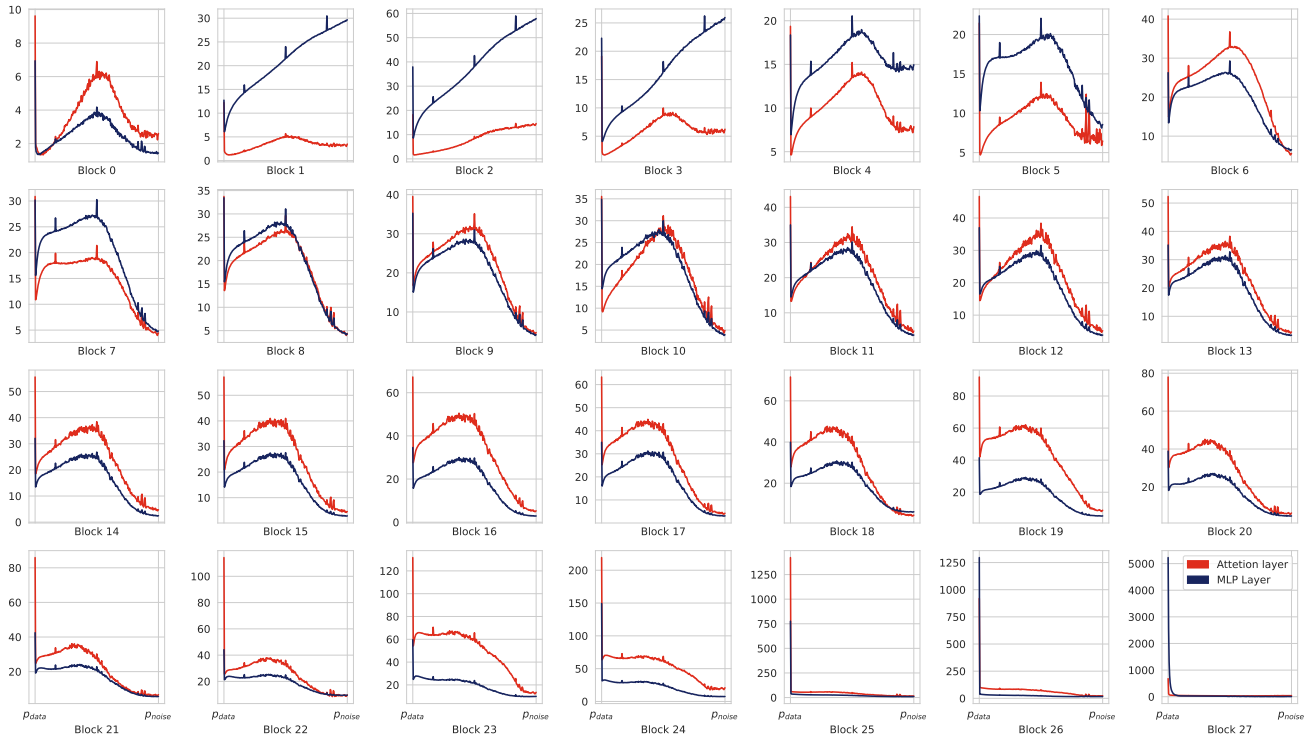


Figure 18. We plot average distance (L_2-norm) between activation of different layers during successive steps of the denoising process of the *DiT-XL/2* model. Different layers exhibit different characteristics. Similar observations have been made in [70].

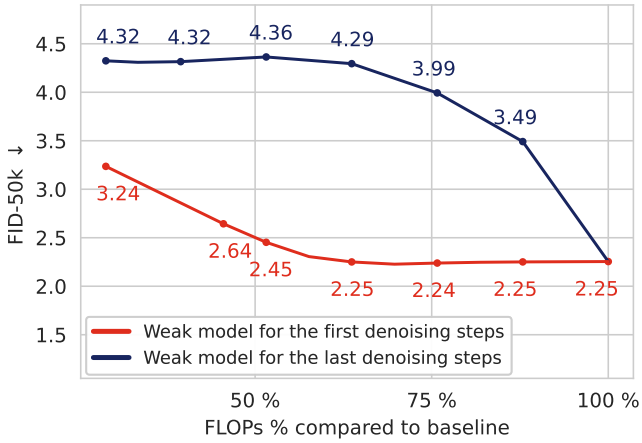


Figure 19. We compare our scheduler versus a different scheduler that uses the weak model for the last denoising steps when generating class-conditioned images. Points correspond to the minimum — concerning CFG scale — FID values.

use the update rule as presented in the main text

$$\begin{cases} \epsilon_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \emptyset; p_{\text{uncond}}) + s_{\text{cfg1}}(\epsilon_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, c; p_{\text{cond}}) - \epsilon_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, \emptyset; p_{\text{uncond}})), & \text{if } p_{\text{cond}} = p_{\text{uncond}} \\ \epsilon_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, c; p_{\text{uncond}}) + s_{\text{cfg2}}(\epsilon_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, c; p_{\text{cond}}) - \epsilon_\theta(\mathbf{x}_{t-1}|\mathbf{x}_t, c; p_{\text{uncond}})), & \text{if } p_{\text{cond}} < p_{\text{uncond}} \end{cases}.$$

This guidance scheme seeks to reduce errors made by the powerful model, enhancing potential differences in predictions of the corresponding weak model, when the two models disagree, indicating the general direction towards higher-quality samples. In practice, different values of s_{cfg1} and s_{cfg2} lead to the best results. We find that the rule $(1-s_{\text{cfg1}})/(1-s_{\text{cfg2}}) = 2.5$,

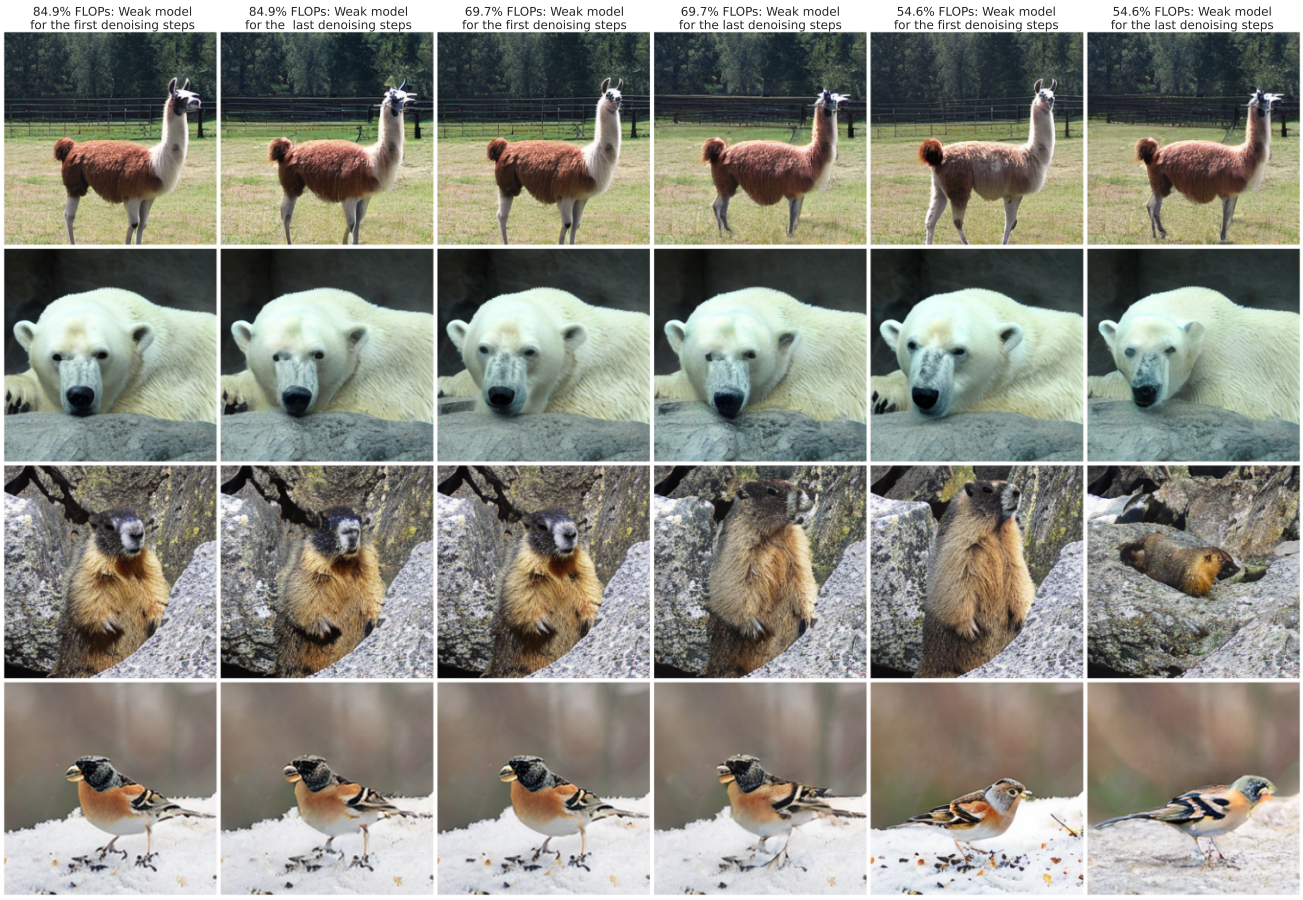
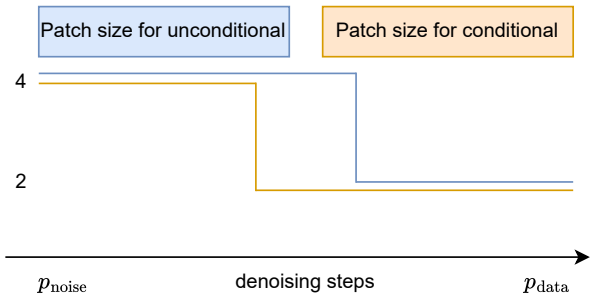


Figure 20. We compare our scheduler versus a different scheduler that uses the weak model for the last denoising steps when generating class-conditioned images. Using the weak model for the last denoising steps leads to images with lower image fidelity.



works consistently across experiments. Although we fix the value of the CFG scale during inference, different combinations are likely to lead to higher quality images as demonstrated by previous work [14], which we leave for future exploration.

We point out that our scheduler is very stable in terms of performance attained for similar compute. For instance, performing inference with a $70/70$ scheduler or a $90/50$ scheduler, which both require the same overall compute, produces FID results of 2.64 and 2.65 respectively. Finally, we present detailed experiments on the effect of the CFG scale for different levels of compute and more metrics in Fig. 21.

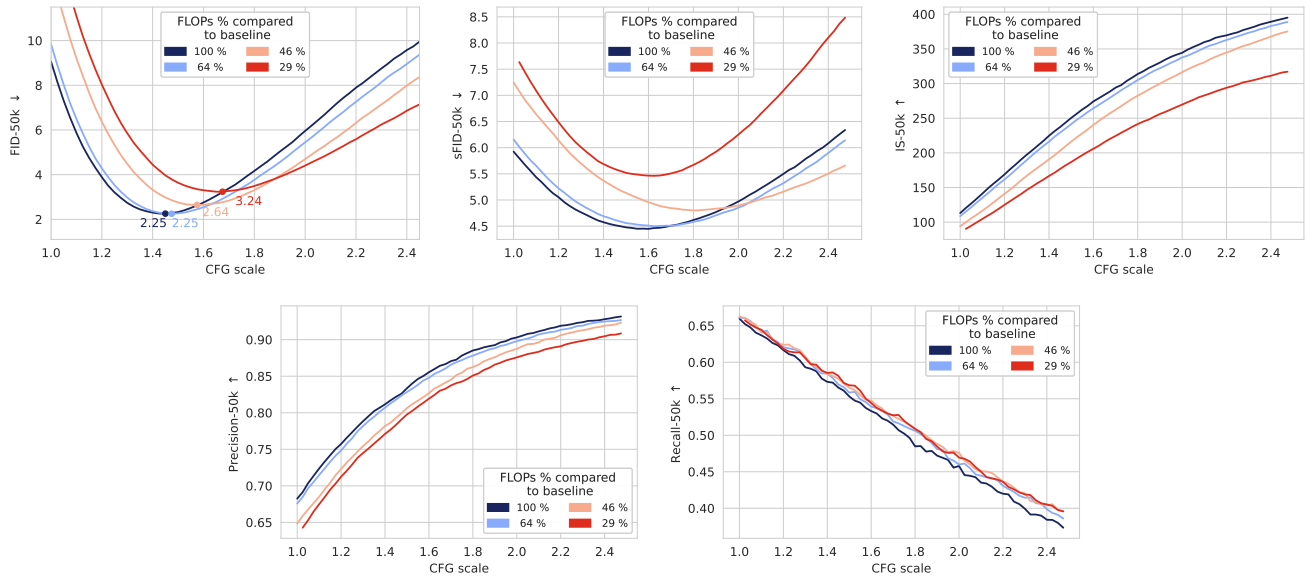


Figure 21. Effect of CFG scale on the generated images from our class-conditioned *FlexiDiT* model. We plot (a) FID, (b) sFID, (c) inception score, (d) precision, and (e) recall when generation $50,000$ samples with 250 steps of the DDPM scheduler.

B.5. Text-to-Image Experiments

Generally, T2I generation is performed for a fixed target CFG scale. For our experiments we choose $s_{\text{cfg}} = 4.5$ for the *T2I Transf.* model, as this is the value used in [15] and $s_{\text{cfg}} = 6.0$ for the *Emu* model, as for these values we observed the best quality images. In general, we can match with our dynamic inference other target values of the CFG scale. One simply needs to adjust the used CFG scale for the dynamic inference accordingly.

We follow the evaluation protocol of PIXART-α [15] and perform inference using the same solvers as they do, namely iDDPM [24] for 100 steps, DPM solver [68] for 20 steps, and SA solver [106] for 25 steps. In the main text — Fig. 7 (left) — we presented results for the iDDPM solver. We present results for all the schedulers with the settings used in PIXART-α in Fig. 22, 23 and 24. For all the schedulers, there are settings where we reach the Pareto front of FID vs CLIP score of the baseline model with a lot less required compute.

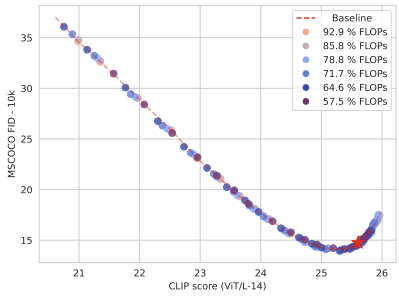


Figure 22. FID vs CLIP score using iDDPM for 100 steps for the *T2I Transf.* model.

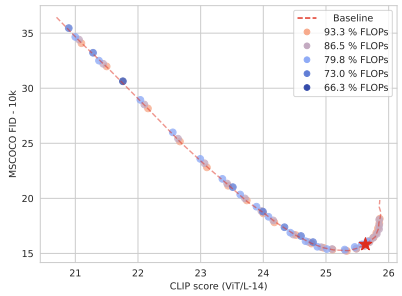


Figure 23. FID vs CLIP score using the DPM-solver for 20 steps for the *T2I Transf.* model.

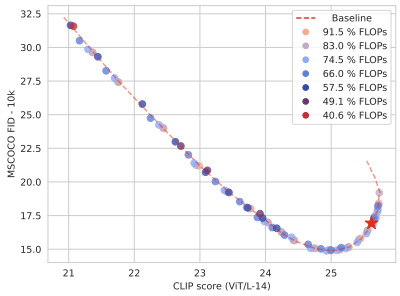


Figure 24. FID vs CLIP score using the SA-solver for 25 steps for the *T2I Transf.* model.

To better characterize the effect of reducing compute, i.e. heavier use of the weak model, we also present more detailed results for the DDPM scheduler in Fig. 25. Less compute-heavy inference schedulers, often produce images with smaller possible maximum CLIP scores (for large CFG guidance scales s_{cfg}). In practice, as large CFG scale values lead to larger values of FID, these are less preferred. In every case, our weak models can max the FID vs CLIP score tradeoff of the base model for the default configuration used, i.e. $s_{\text{cfg}} = 4.5$.

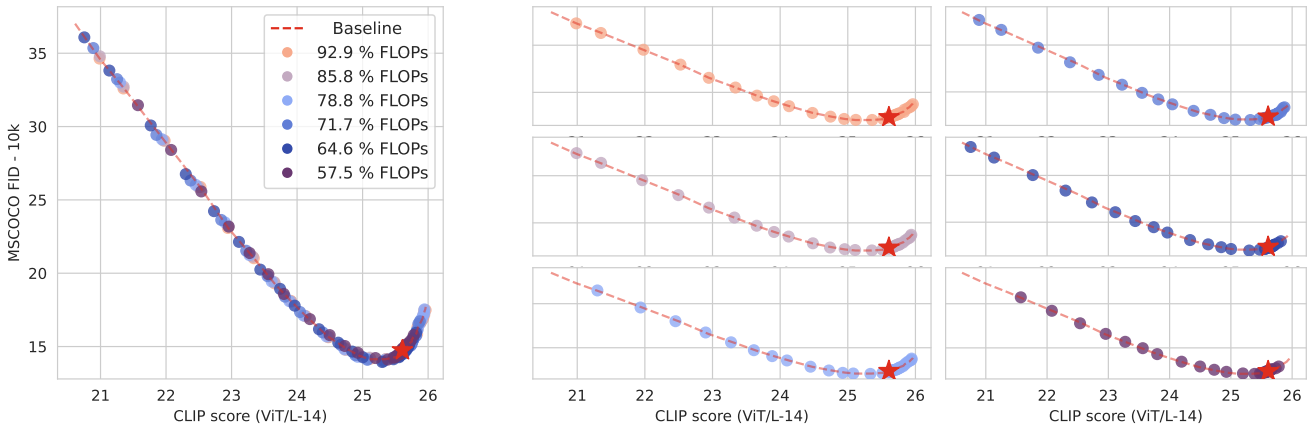


Figure 25. FID vs CLIP score using DDPM for 100 steps for the *T2I Transf.* model, for different levels of compute. On the right, we present results separately for each compute level.

We also provide results on using a smaller overall number of steps with the DDPM solver in Fig. 27. To generate the baseline curves, we sample $10,000$ samples using a CFG scale s_{cfg} from the set $\{1.0, 1.125, 1.25, 1.375, 1.5, 1.625, 1.75, 2.0, 2.25, 2.5, 2.75, 3.0, 3.5, 4.0, 4.5, 5.0, 5.5, 6.0, 6.5, 7.0, 7.5, 8.0, 8.5\}$. The same values are used when sampling with our flexible models. Finally, we also provide FID vs CLIP for the *Emu* model in Fig. 26. In this case, we take CFG scales s_{cfg} from the set $\{1.0, 1.5, 2.0, 2.5, 3.0, 4.5, 6.0, 7.5, 8.0, 9.0\}$. We use captions from the training set of MS COCO to generate images. Neither of the models was trained on images from this dataset.

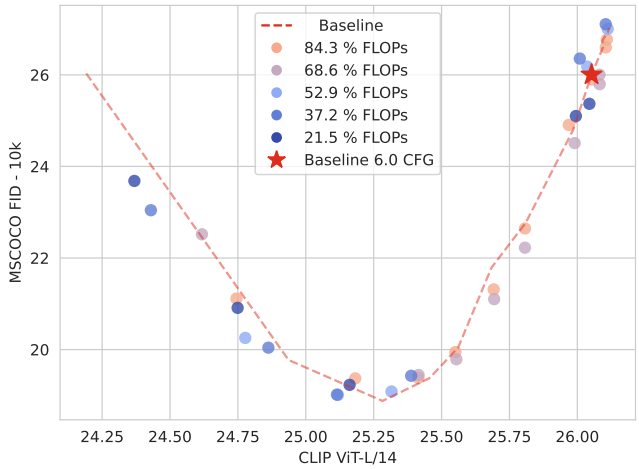


Figure 26. FID vs CLIP score using DDIM for 50 steps for the *Emu* model.

VQA results. VQA scores are calculated by querying a visual-question-answering model to produce an alignment score by computing the probability of a "Yes" answer to a simple "Does this figure show {text}?". question. To calculate this score, we use the *clip-flant5-xxl* model from huggingface[7] as suggested in [60]. We provide more detailed results on the VQA benchmark in Tables 1 and 2. More specifically, we provide per dataset VQA scores, along with the CFG scale s_{cfg} used to generate the images for each case . As we can see, using the weak model requires a bigger CFG scale to reach the same level of optimality (calculated from the FID vs CLIP score tradeoff). We also note that using the weak model often leads to images with better text alignment. We hypothesize that fewer tokens (as a result of larger patch sizes) help with spatial consistency at the beginning of the denoising process. To calculate VQA scores, we take the first 200 prompts from each dataset and

[7] https://huggingface.co/zhiqiulin/clip-flant5-xxl

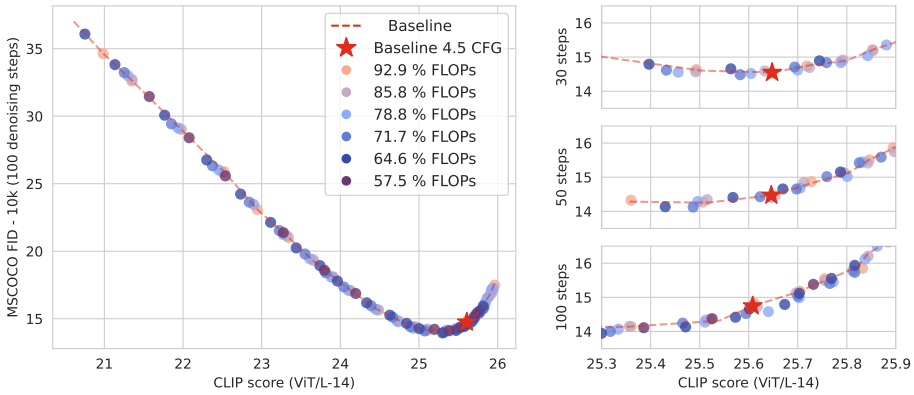


Figure 27. We plot FID vs CLIP score when generating images with different CFG scales. (left) Overall Pareto front when generating images with 100 denoising steps. (right) Pareto front generating images with a different number of steps zoomed in the typical tradeoff generation values.

use the *train* split from the *DrawBench* [84], *train* split from the *Pick-a-Pic* [55], *test* split from the *Winoground* [94] and *tifa_v1.0_text_inputs*[8] from the *TIFA160* [43] dataset.

	CFG scale s_{cfg}	*DrawBench*	*Pick-a-Pic*	*Winoground*	*TIFA160*	Average
T2I Transf. (100 %)	4.5	58.93	50.56	62.01	81.65	63.29
T2I Transf. (92.9 %)	4.5	58.37	51.53	62.09	<u>82.16</u>	63.54
T2I Transf. (85.8 %)	4.5	57.58	51.67	62.41	81.53	63.30
T2I Transf. (78.8 %)	4.7	58.62	51.97	62.04	80.60	63.31
T2I Transf. (71.7 %)	4.7	58.44	51.56	63.03	80.57	63.40
T2I Transf. (64.6 %)	4.9	<u>60.16</u>	<u>52.34</u>	62.98	80.06	<u>63.89</u>
T2I Transf. (57.7 %)	5.0	59.04	50.80	<u>63.27</u>	79.90	63.26
T2I Transf. (50.5 %)	5.0	56.77	51.72	61.87	79.38	62.44
T2I Transf. (43.4 %)	5.0	56.87	51.92	61.30	78.33	62.11

Table 1. Detailed VQA evaluations for the benchmarks tested with the *T2I Transf.* model. As in the class-conditioned experiments, using more of the weak model during denoising requires a higher CFG scale s_{cfg} to reach optimum performance.

	CFG-scale s_{cfg}	*DrawBench*	*Pick-a-Pic*	*Winoground*	*TIFA160*	Average
Emu (100 %)	6.0	69.44	58.70	65.75	<u>86.77</u>	70.17
Emu (84.3 %)	6.0	68.00	58.93	<u>67.33</u>	86.51	70.19
Emu (68.6 %)	6.25	69.53	<u>60.62</u>	66.00	85.33	<u>70.37</u>
Emu (52.9 %)	6.5	<u>69.79</u>	58.14	66.23	86.20	70.09

Table 2. Detailed VQA evaluations for the benchmarks tested with the *Emu* model. As in the class-conditioned experiments, using more of the weak model during denoising requires a higher CFG scale s_{cfg} to reach optimum performance.

Alignment between powerful and weak model. It is common practice nowadays to train images (and especially videos) in different stages, where a large (potentially lower quality) dataset is used for the first stage, followed by a shorter fine-tuning stage, characterized by higher quality and aesthetically more pleasing images. Although we are directly distilling the weak model from the predictions of the powerful model, the data used throughout training are still important. In practice,

[8] https://github.com/Yushi-Hu/tifa/blob/main/tifa_v1.0/tifa_v1.0_text_inputs.json

our fine-tuning is sample efficient, and we find that even a few thousand images (< 5000) are enough to succeed. We thus suggest fine-tuning on the last (potentially smaller) but higher-quality dataset. When generating images based on shorter prompts with *Emu*, we use a prompt re-writer, prompting a small LLM to expand on the information provided. We consider this prompt re-writer as part of the model.

C. Implementation Details

We provide additional details on the experiments in the main text.

C.1. Figure Details

Prompts used for Fig. 1. We provide in Table 3 the exact prompts used to generate the images.

Prompts for Fig. 1
The image shows a frog wearing a golden crown with intricate designs, sitting on a wooden log in a serene environment reminiscent of a Japanese anime setting. The frog's crown is adorned with small gems and its eyes are large and expressive. The log is covered in moss and surrounded by lush greenery, with a few cherry blossoms visible in the background. The frog's skin is a vibrant shade of green with blue stripes, and it has a regal demeanor, as if it is a monarch of the forest. The overall atmosphere is peaceful and whimsical.
The image shows a serene waterfall cascading down a rocky slope in a lush tropical forest, reminiscent of Claude Monet's impressionist style. Sunlight filters through the dense foliage above, casting dappled shadows on the misty veil surrounding the falls. The water plunges into a crystal-clear pool, surrounded by large rocks and vibrant greenery. The atmosphere is tranquil, with a warm color palette and soft brushstrokes evoking a sense of serenity. The forest floor is covered in a thick layer of leaves, and the sound of the waterfall echoes through the air.

Table 3. Details on the prompts used to generate the images in the paper.

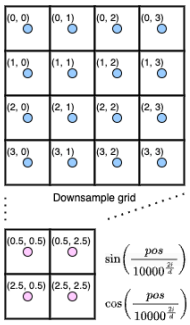
Details on Fig. 2. In Fig. 2 (b), we fix randomness of the denoising process in terms of the initial sampled image $p(\mathbf{x}_T)$, and from the denoising process in Eq. (2). Then we generate images with 250 steps using the *DiT-XL/2* official public checkpoint. During denoising, we modify only 1 of the 250 denoising steps, bypassing the model predictions from that step through a high/low pass filter. We then compare the resulting generated images in terms of LPIPS, L_2 distance, SSIM, and DreamSim. In general, modifying one of the first denoising steps, leads to larger final image differences, due to the accumulation of differences. We still note a distinctive pattern: a high-pass filter, i.e. removing low-pass components, leads to larger image differences during the first denoising steps. The opposite holds for the last denoising steps. We can thus argue, that low-pass components, i.e. 'coarser' image details are more important compared to high-frequency details, for the first denoising steps. To calculate spatial frequencies, we keep the corresponding values of the FFT of an image.

Details on Fig. 9. In Fig. 9, we plot GPU utilization when propagating different sequence lengths. All experiments are conducted in *bfloat16* using *PyTorch 2.5* and *CUDA 12.4* and with our *Emu* DiT that has 24 layers and a hidden dimension of 2048. We also plot peak FLOPs and memory bandwidth for the GPU tested, an *NVIDIA H100 SXM5* in this case. When we report FLOPs, we count additions and multiplications separately, as it is commonly done [41]. We count FLOPs as the theoretical required operations for a forward pass, and not the actual GPU operations, which might be a higher number due to the potential recalculation of partial results. As bytes for the x-axis, we only consider the model parameters (2 bytes per model parameter). Note that for our choice of weak models, the GPU is fully utilized (it reaches the maximum compute intensity that can be achieved for this application). In reality, compute intensity drops when larger sequence lengths are used, mainly due to the larger memory footprint of intermediate activations. Thus, latency benefits are indeed even larger than FLOPs benefits reported in the paper. Compiling and fusing operations is crucial to ensure that the GPU is not bottlenecked by waiting instructions from the CPU. When we are performing inference with the weak model without first merging the LoRAs, inference time is proportional to the additional FLOPs required. For the attention operation, we use the *memory_efficient_attention* operation from the *xformers* library. Other efficient implementations of attention do not lead to significant differences. Our T2I model operates in a 128×128 latent space, which means that using a patch size of $(1, 2, 2)$ results in 4096 tokens, compared to 1024 tokens for the $(1, 4, 4)$ patch size. Our T2V model operates in a $32 \times 88 \times 48$ latent space, which means that using a patch size of $(1, 2, 2)$, $(2, 2, 2)$, $(1, 4, 4)$ leads to 33792, 16896 and 8448 tokens respectively. We obtain similar results using different-sized models, like our *Video DiT* model.

When we report FLOPs in the paper, we report numbers for the denoising process, as it is commonly done. We thus ignore latency induced by decoders that map samples from a latent space, or potential prompt-rewrite modules. The added latency by the decoder, is in our settings negligible.

C.2. Flexifying Diffusion Transformers

Although for the class-conditioned experiments, we use a single embedding and de-embedding layer that we always project to the required patch size, we note that this projection can be done once to pre-calculate embedding and de-embeddding parameters during inference. These projected embeddings can then be used out of the box, for the tradeoff of some minuscule additional memory. The choice of p' as the underlying patch size is not too important for our experiments. In practice, we use a value of $p' = 4$. As mentioned in the main text, we add positional embeddings according to the coordinates of each patch in the original image. A schematic of this can be found on the right.



Apart from the architecture modifications listed in the main text, we experimented with adding patch size specific temperatures in the self-attention computation:

$$\text{softmax}\left(\frac{\mathbf{Q}\mathbf{K}^\top}{\tau_p\sqrt{d}}\right)\mathbf{V},$$

where $\mathbf{Q}, \mathbf{K}, \mathbf{V}$ are the queries, keys and values respectively and τ_p is a patch size specific temperature initialized as 1. We do not include this in the end, as it occasionally leads to instabilities during fine-tuning, even under different parametrizations.

Class-conditioned implementation details. We largely use the same hyperparameters as [78] to fine-tune. When fine-tuning to match distributions, we train to minimize the MMD loss, as introduced in Section B.1. During bootstrapping, we denoise images with a DDPM scheduler, operating on $T = 250$ steps, the same as the target inference scheduler. As we found that MMD distance is higher for diffusion steps closer to $x_0 \sim q(\mathbf{x}_0)$ — see also Fig. 11 —, we bias the sampling to reflect that during training as well.

Parameter	Value
training data	*ImageNet*
learning rate	10^{-4}
weight decay	0.0
EMA update frequency	every step
EMA update rate	0.9999

Table 4. Class-conditioned implementation details.

In our experiments, we focused on fine-tuning pre-trained models, as we were interested in efficiency. We note that training with different patch sizes has been used in the past to also accelerate pre-training [3, 9]. We believe that flexible patch sizes can also be used in this application to accelerate pre-training.

***T2I Transf.* implementation details.** For the *T2I Transf.* model, we follow exactly the recipe of [15], and fine-tune a 256×256 pre-trained variant[9]. For fine-tuning, we use the same image dataset, namely the SAM dataset[10] with captions generated from a vision-language model. The model has overall the same parameters as the *DiT-XL/2* model, with the addition of cross-attention blocks. When adding new embedding and de-embedding layers, we initialize them as we did for the class-conditioned experiments. Embedding layers are initialized to $Q_{\text{embed}}{}^\dagger w_{\text{embed}}$ and de-embedding layers are initialized to $w_{\text{de-embed}} Q_{\text{de-embed}}{}^\dagger$. Here w_{embed}, $w_{\text{de-embed}}$ are the pre-trained model parameters and Q_{embed}, $Q_{\text{de-embed}}$ are the same — patch size dependent — fixed projection matrices that better preserve the norm of the output activations at initialization. As aforementioned, we add a patch size embedding that is added to all tokens in the sequences after the tokenization step. This embedding is equal to 0 for the pre-trained patch size, to ensure functional preservation. We fine-tune the *T2I Transf.* model on a small subset of the SAM dataset used to originally train the target model. We add LoRAs on the self-attention and feed-forward layers, with a LoRA dimension of 32. We use a higher learning rate, due to the different learning objectives — distilling a powerful model's predictions into the ones of a weal model.

[9] Our starting pre-trained model exactly matches the public checkpoint https://huggingface.co/PixArt-alpha/PixArt-XL-2-SAM-256x256.

[10] https://segment-anything.com/.

Parameter	Value
training data	SAM with captions from a VLM model
optimizer	*AdamW*
learning rate	8×10^{-4}
weight decay	10^{-2}
gradient clipping	0.02
batch size	512
EMA update frequency	every step
EMA update rate	0.9999
LoRA rank	32

Table 5. Image text-conditioned implementation details.

***Emu* implementation details.** Our *Emu* model is fundamentally identical to the *T2I Transf.* model. Small variations are due to different ways to calculate text embeddings, which lead to a different number and size of the cross-attention tokens, and slight architectural modifications — primarily the use of QK-normalization [22] and the use of learnable positional embeddings. We train using a high-quality aesthetic dataset. To calculate metrics based on the 1024×1024 images generated with this model, we follow the evaluation protocol of [105] and resize images to 512×512. For both our *Emu* and our *Video DiT* model, we use a LoRA rank of 64.

T2V implementation details. As aforementioned, our *Video DiT* model has a pre-trained patch size of $(p_{\text{f}}, p_{\text{h}}, p_{\text{w}}) = (1, 2, 2)$. Compared to our T2I experiments, we only change the 2D convolutional layers used for tokenization with a 3D convolution layer. When increasing the temporal patch size p_{f}, we duplicate parameters along that dimension. When interpolating positional embeddings, we also do that along the temporal dimension. No additional changes are made. For evaluation, we use the prompts from `https://github.com/facebookresearch/MovieGenBench/blob/main/benchmark/MovieGenVideoBench.txt` to generate videos of length equal to 256 frames. We evaluate according to VBench [45] and report the average over *Subject Consistency*, *Background Consistency*, *Temporal Flickering*, *Motion Smoothness*, *Dynamic Degree*, *Aesthetic Quality*, *Imaging Quality*, *Temporal Style* and *Overall Consistency*.

C.3. Human Evaluation Details

We prompt humans, asking them to: "Compare the two side-by-side images. Focus on visual appeal and flawlessness, considering factors like aesthetic appeal, clarity, composition, color harmony, lighting, and overall quality, then select 'left' if the left image is better, 'right' if the right image is better, or 'tie' if both are equally appealing or you have no preference.". In total, we collected votes for the 4 different settings presented in the paper and aggregated them across 200 prompts. For each setting and each prompt, we ask 3 people for votes. In cases where there are 3 votes for each of 'left', 'right', and 'tie', we ask a fourth labeler to break the tie.

D. Generated Samples

We provide more examples of generated samples.

D.1. Text-Conditioned Image Generation

We showcase more examples with varying amounts of compute in Fig. 28, 29 and 30. We further show more examples of how performance and diversity in image generation are preserved in Fig. 31 and 32. Finally, we show examples of the effect of CFG scale s_{cfg} and reducing the overall number of FLOPs used to generate an image using our method, in Fig. 33. For all the images seen in the paper with our *Emu* model, we use 50 steps of the DDIM scheduler.

D.2. Text-Conditioned Video Generation

We showcase more examples of video generation with our flexible model in Fig. 34 and 35.

D.3. Class-Conditioned *ImageNet* Generation

We provide a more comprehensive comparison for generated images from the same original sample from p_{noise}, using varying amounts of compute from our flexible model in Fig. 36. Note that for class-conditioned generation we do not use

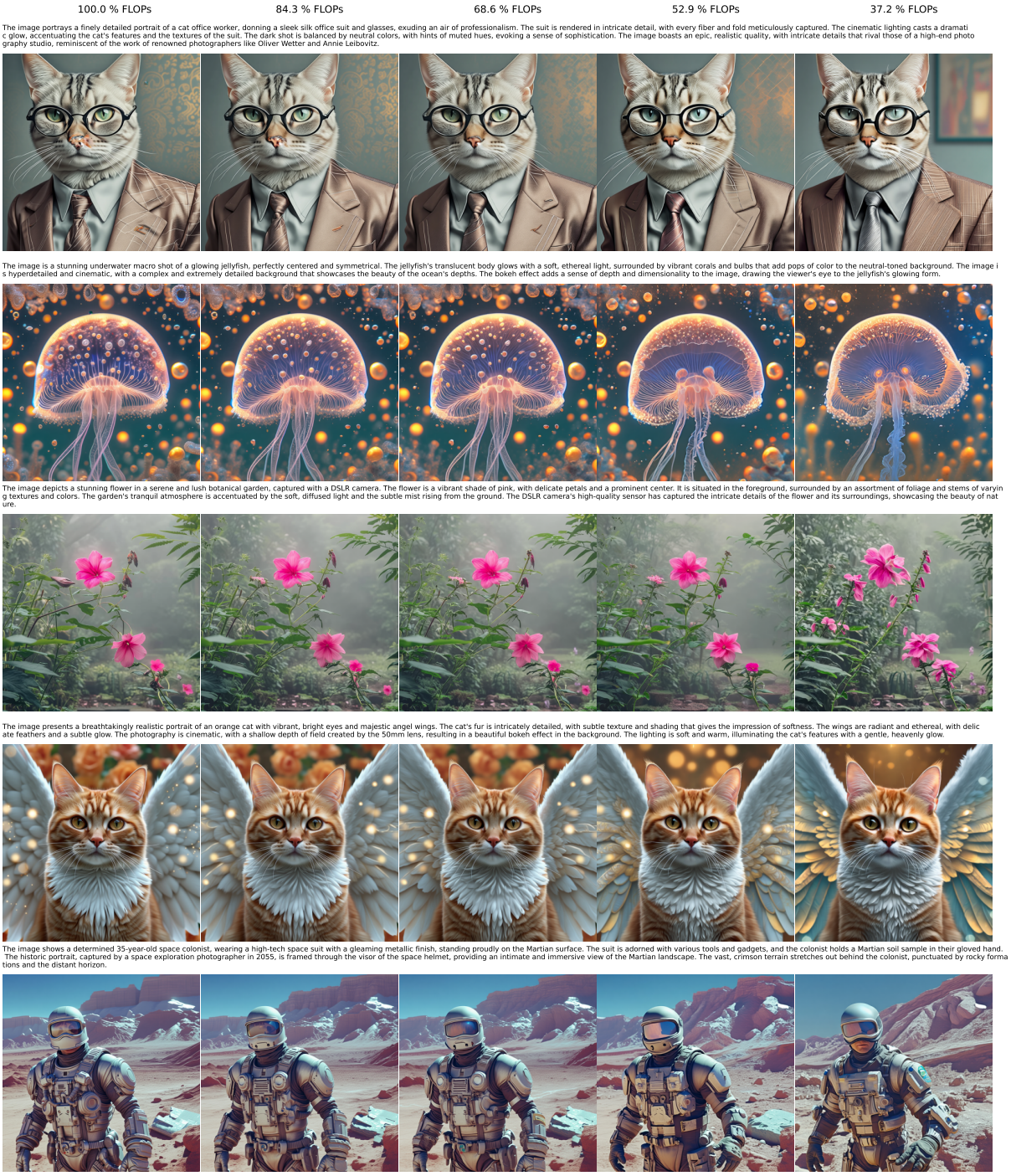


Figure 28. More samples generated by our *Emu* model for varying amounts of compute.

LoRAs, and images generated from the original baseline model may not be exactly the same. They do however have the same characteristics (FID score) as seen in Sec 4.1. We also show samples of our flexible *DiT-XL/2* model when using only 64% of the compute of the original model in Fig. 37, 38, 39, 40, 41, 42 and 43. All images are generated using 250 DDPM steps and a CFG-scale equal to 4.0.

Figure 29. More samples generated by our *Emu* model for varying amounts of compute.

Figure 30. More samples generated by our *Emu* model for varying amounts of compute.

Figure 31. Samples for the prompt: ”A playful kitten just waking up.”. We showcase the image generated by the baseline and our flexible scheduler using only 53% of FLOPs.

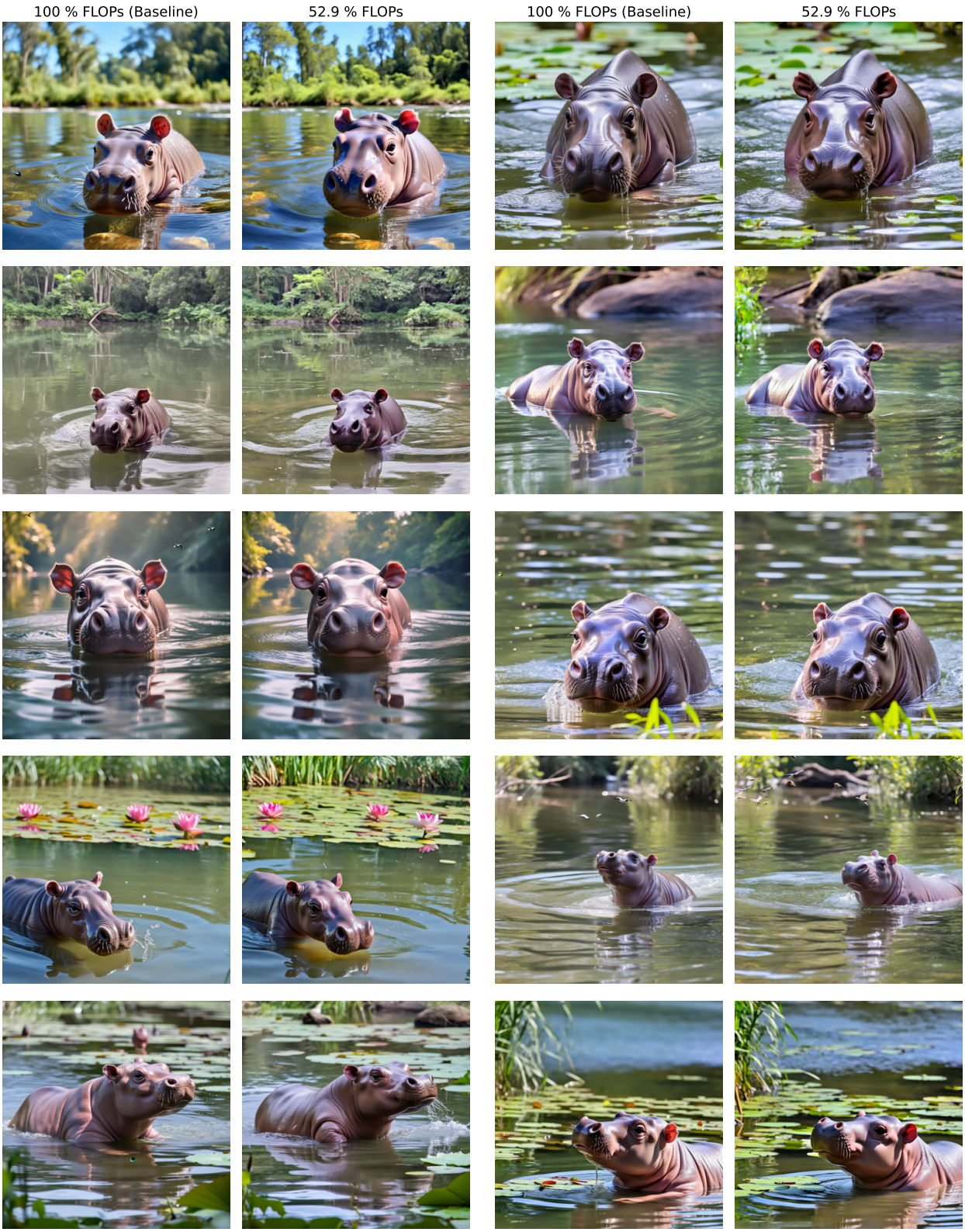
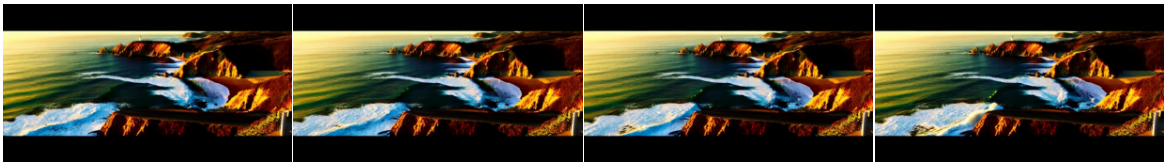


Figure 32. Samples for the prompt: ”A baby hippo swimming in the river.”. We showcase the image generated by the baseline and our flexible scheduler using only 53% of FLOPs.

Figure 33. Effect of CFG and total compute for the prompt: ‘The image shows a gigantic juicy burger placed on a white plate on a wooden table. The burger is composed of a large beef patty, crispy bacon, melted cheese, lettuce, tomato, onion, pickles, and a slice of red tomato, all sandwiched between a soft bun. The burger is so large that it occupies most of the plate, with some toppings falling out of the sides. The bun is slightly toasted, and the cheese is melted to perfection, giving off a savory aroma. The burger is garnished with a side of crispy fries and a refreshing glass of cola.‘.

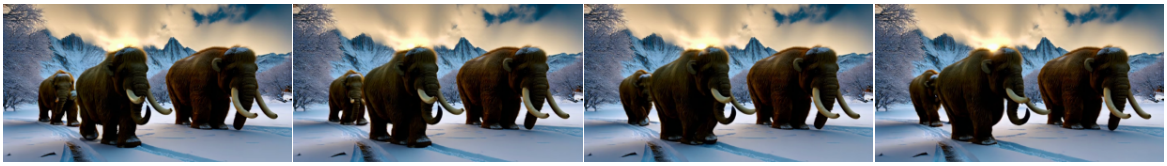
Drone view of waves crashing against the rugged cliffs along Big Sur's garay point beach. The crashing blue waters create white-tipped waves, while the golden light of the setting sun illuminates the rocky shore. A small island with a lighthouse sits in the distance, and green shrubbery covers the cliff's edge. The steep drop from the road down to the beach is a dramatic feat, with the cliff's edges jutting out over the sea. This is a view that captures the raw beauty of the coast and the rugged landscape of the Pacific Coast Highway.



A gorgeously rendered papercraft world of a coral reef, rife with colorful fish and sea creatures.

A litter of golden retriever puppies playing in the snow. Their heads pop out of the snow, covered in.

Several giant wooly mammoths approach treading through a snowy meadow, their long wooly fur lightly blows in the wind as they walk, snow covered trees and dramatic snow capped mountains in the distance, mid afternoon light with wispy clouds and a sun high in the distance creates a warm glow, the low camera view is stunning capturing the large furry mammal with beautiful photography, depth of field.



This close-up shot of a Victoria crowned pigeon showcases its striking blue plumage and red chest. Its crest is made of delicate, lacy feathers, while its eye is a striking red color. The bird's head is tilted slightly to the side, giving the impression of it looking regal and majestic. The background is blurred, drawing attention to the bird's striking appearance.

A tropical fish swimming in ocean reefs

Aerial shot of the ocean. a maelstrom forms in the water swirling around until it reveals the fiery depths below.

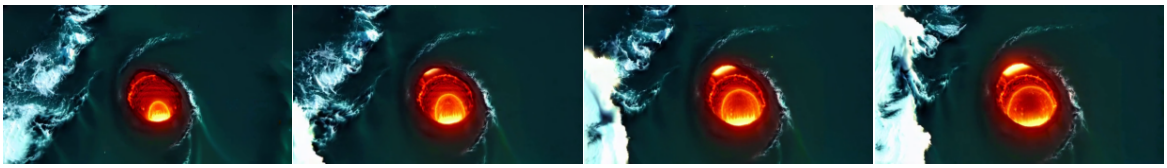


Figure 34. More samples generated by our *Video DiT* model, using 25.2 % compute compared to the pre-trained baseline.

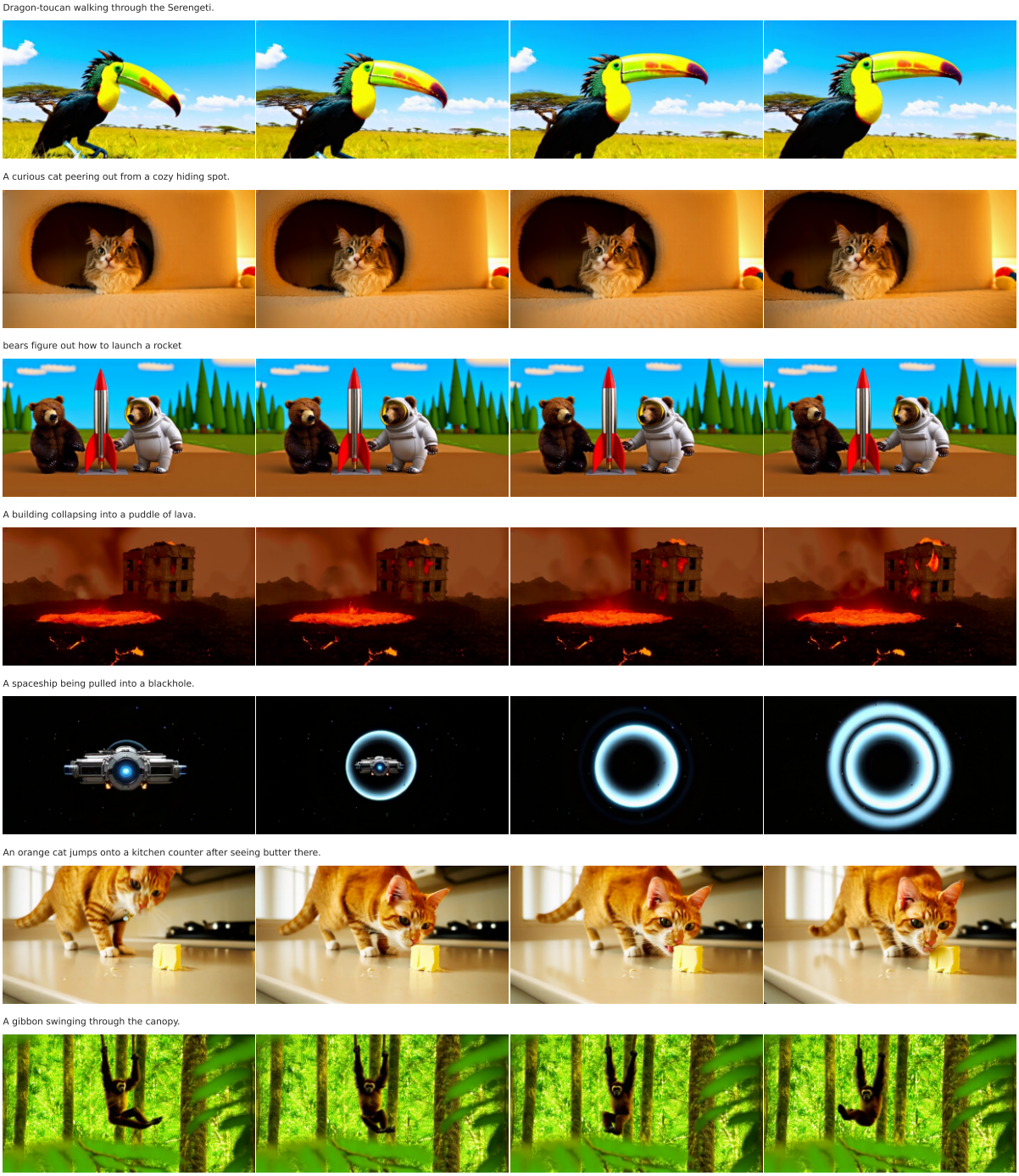


Figure 35. More samples generated by our *Video DiT* model, using 25.2 % compute compared to the pre-trained baseline.

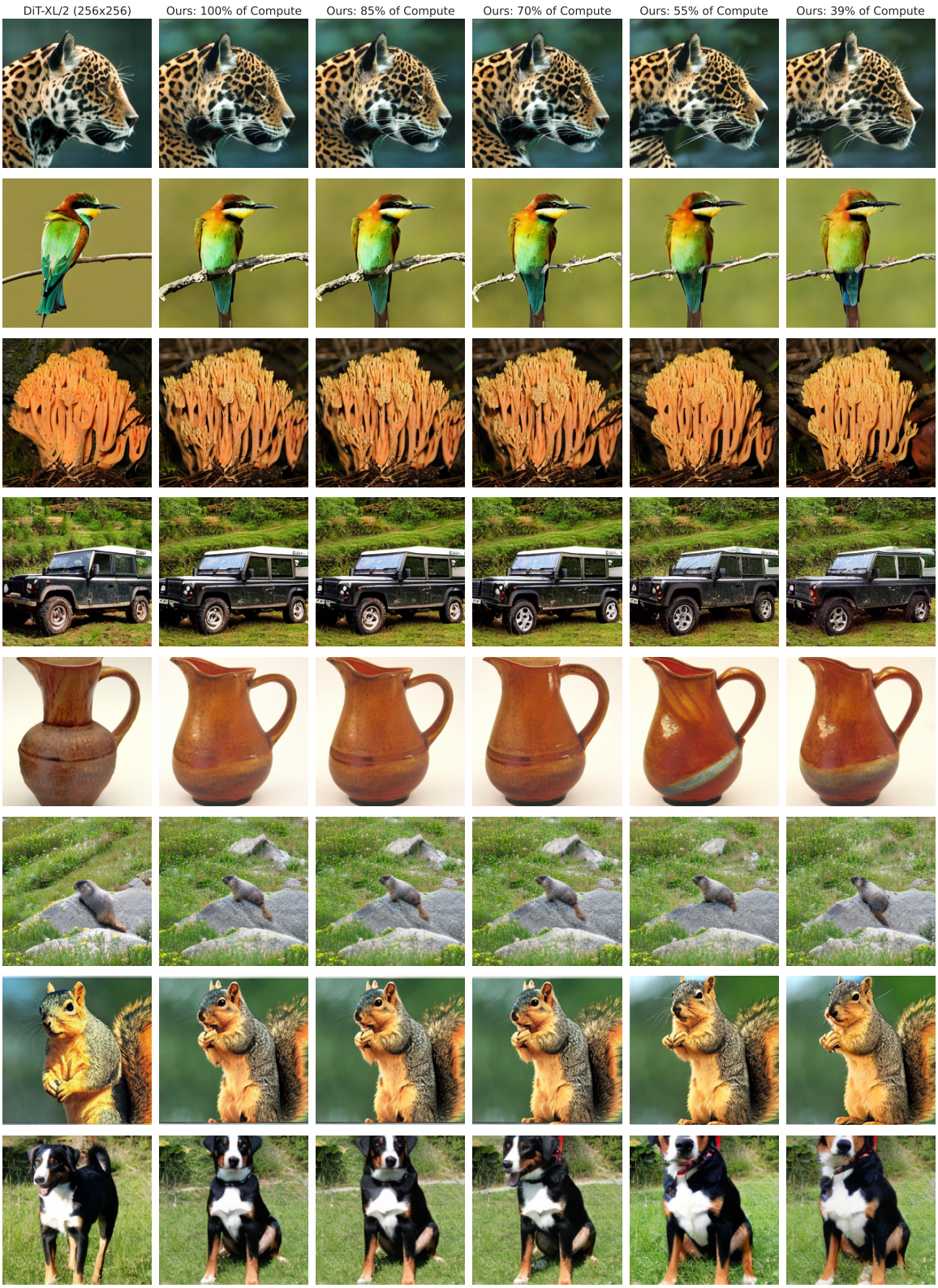


Figure 36. Sample for the *ImageNet* dataset comparing the baseline model and varying inference schedulers of our model, using different levels of compute.

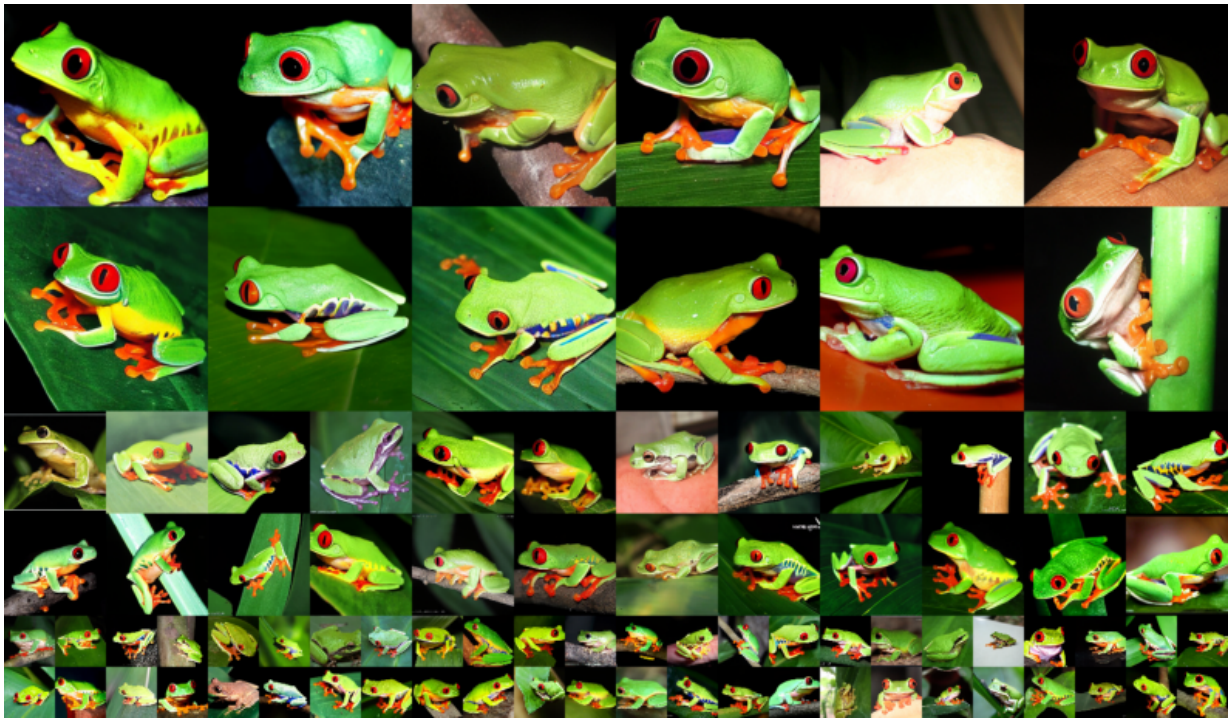


Figure 37. Samples for the *ImageNet* class 'tree frog, tree-frog' from our *FlexiDiT* model that uses only 46% of the compute compared to the baseline model.

Figure 38. Samples for the *ImageNet* class 'prairie chicken, prairie grouse, prairie fowl' from our *FlexiDiT* model that uses only 46% of the compute compared to the baseline model.

Figure 39. Samples for the *ImageNet* class 'hummingbird' from our *FlexiDiT* model that uses only 46% of the compute compared to the baseline model.

Figure 40. Samples for the *ImageNet* class 'cairn, cairn terrier' from our *FlexiDiT* model that uses only 46% of the compute compared to the baseline model.

Figure 41. Samples for the *ImageNet* class 'French bulldog' from our *FlexiDiT* model that uses only 46% of the compute compared to the baseline model.

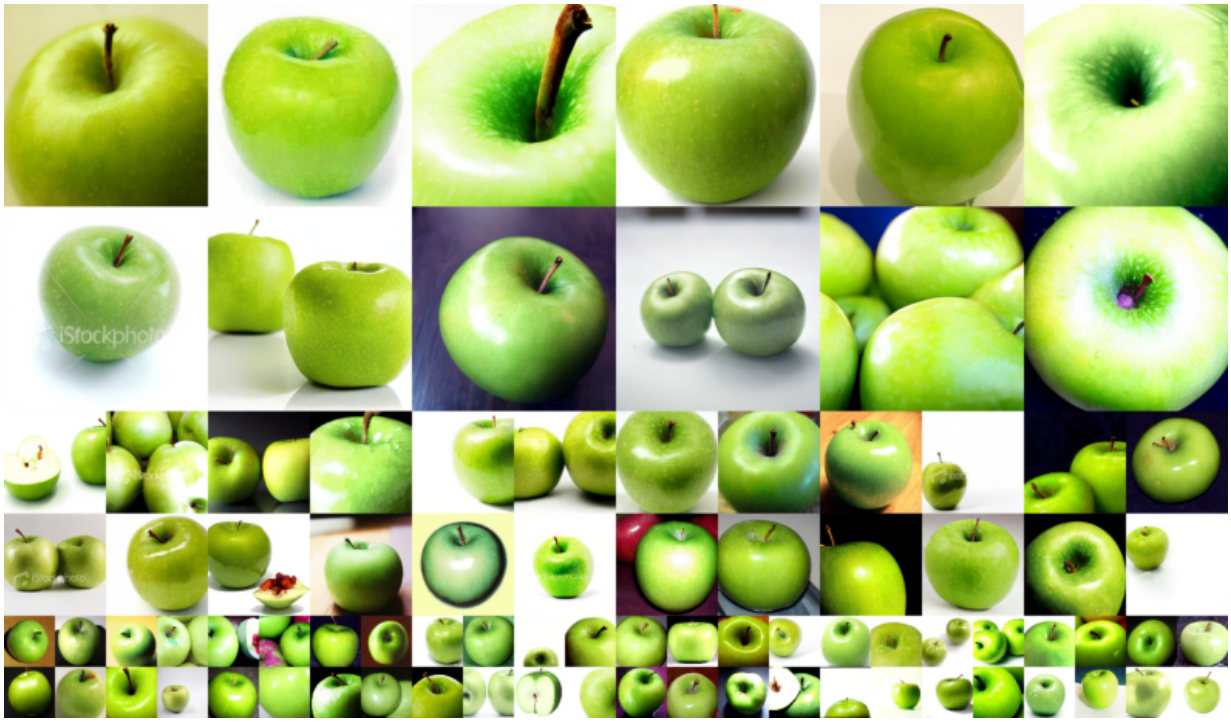


Figure 42. Samples for the *ImageNet* class 'Granny Smith' from our *FlexiDiT* model that uses only 46% of the compute compared to the baseline model.

Figure 43. Samples for the *ImageNet* class 'cock' from our *FlexiDiT* model that uses only 46% of the compute compared to the baseline model.