Cache Me if You Can: Accelerating Diffusion Models through Block Caching

Felix Wimbauer[1,2,3] Bichen Wu[1] Edgar Schoenfeld[1] Xiaoliang Dai[1] Ji Hou[1]
Zijian He[1] Artsiom Sanakoyeu[1] Peizhao Zhang[1] Sam Tsai[1] Jonas Kohler[1]
Christian Rupprecht[4] Daniel Cremers[2,3] Peter Vajda[1] Jialiang Wang[1]
[1]Meta GenAI [2]Technical University of Munich [3]MCML [4]University of Oxford
felix.wimbauer@tum.de jialiangw@meta.com

Figure 1. **Speeding up diffusion models through block caching.** We observe that there are many redundant layer computations at different timesteps in diffusion models when generating an image. Our block caching technique allows us to avoid these unnecessary computations, therefore speeding up inference by a factor of **1.5x-1.8x** while maintaining image quality. Compared to the standard practice of naively reducing the number of denoising steps to match our inference speed, our approach produces more detailed and vibrant results.

Abstract

Diffusion models have recently revolutionized the field of image synthesis due to their ability to generate photorealistic images. However, one of the major drawbacks of diffusion models is that the image generation process is costly. A large image-to-image network has to be applied many times to iteratively refine an image from random noise. While many recent works propose techniques to reduce the number of required steps, they generally treat the underlying denoising network as a black box. In this work, we investigate the behavior of the layers within the network and find that 1) the layers' output changes smoothly over time, 2) the layers show distinct patterns of change, and 3) the change from step to step is often very small. We hypothesize that many layer computations in the denoising network are redundant. Leveraging this, we introduce block caching, in which we reuse outputs from layer blocks of previous steps to speed up inference. Furthermore, we propose a technique to automatically determine caching schedules based on each block's changes over timesteps. In our experiments, we show through FID, human evaluation and qualitative analysis that Block Caching allows to generate images with higher visual quality at the same computational cost. We demonstrate this for different state-of-the-art models (LDM and EMU) and solvers (DDIM and DPM).

Project page: fwmb.github.io/blockcaching

1. Introduction

Recent advances in diffusion models have revolutionized the field of generative AI. Such models are typically pre-trained on billions of text-image pairs, and are commonly referred to as "foundation models". Text-to-image foundation models such as LDM [41], Dall-E 2/3 [2, 38], Imagen [43], and Emu [8] can generate very high quality, pho-

This work was done during Felix' internship at Meta GenAI.

torealistic images that follow user prompts. These foundation models enable many downstream tasks, ranging from image editing [4, 17] to synthetic data generation [20], to video and 3D generations [34, 46].

However, one of the drawbacks of such models is their high latency and computational cost. The denoising network, which typically is a U-Net with residual and transformer blocks, tends to be very large in size and is repeatedly applied to obtain a final image. Such high latency prohibits many applications that require fast and frequent inferences. Faster inference makes large-scale image generation economically and technically viable.

The research community has made significant efforts to speed up image generation foundation models. Many works aim to reduce the number of steps required in the denoising process by changing the solver [10, 27, 28, 45, 61]. Other works propose to distill existing neural networks into architectures that require fewer steps [44] or that can combine the conditional and unconditional inference steps [31]. While improved solvers and distillation techniques show promising results, they typically treat the U-Net model itself as a black box and mainly consider what to do with the network's output. This leaves a potential source of speed up—the U-Net itself—completely untapped.

In this paper, we investigate the denoising network in-depth, focusing on the behavior of attention blocks. Our observations reveal that: 1) The attention blocks change smoothly over denoising steps. 2) The attention blocks show distinct patterns of change depending on their position in the network. These patterns are different from each other, but they are consistent irrespective of the text inputs. 3) The change from step to step is typically very small in the majority of steps. Attention blocks incur the biggest computational cost of most common denoising networks, making them a prime target to reduce network latency.

Based on these observations, we propose a technique called **block caching**. Our intuition is that if a layer block does not change much, we can avoid recomputing it to reduce redundant computations. We extend this by a lightweight **scale-shift alignment mechanism**, which prevents artifacts caused by naive caching due to feature misalignment. Finally, we propose an effective mechanism to **automatically derive caching schedules**.

We analyse two different models: a replacement trained version of Latent Diffusion Models [41] on Shutterstock data, as well as the recently proposed EMU [8], as can be seen in Fig. 1. For both, we conduct experiments with two popular solvers: DDIM [48] and DPM [27]. For all combinations, given a fixed computational budget (inference latency), we can perform more steps with block caching and achieve better image quality. Our approach achieves both improved FID scores and is preferred in independent human evaluations.

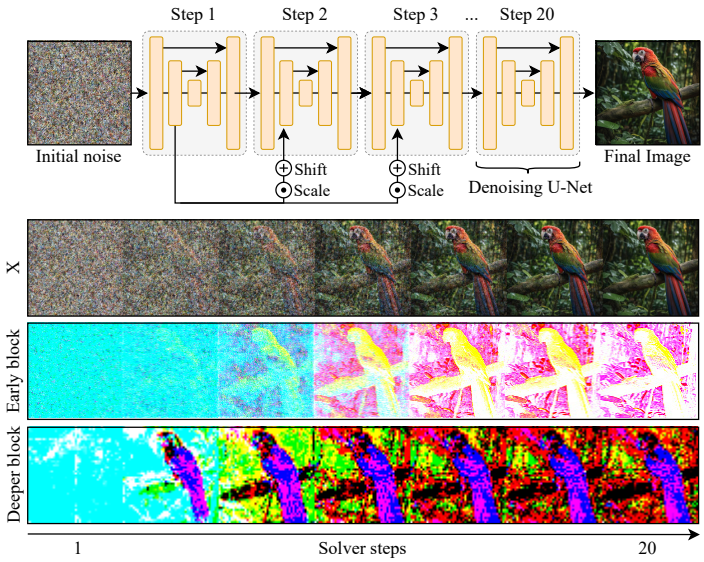
2. Related Work

In the following, we introduce important works that are related to our proposed method.

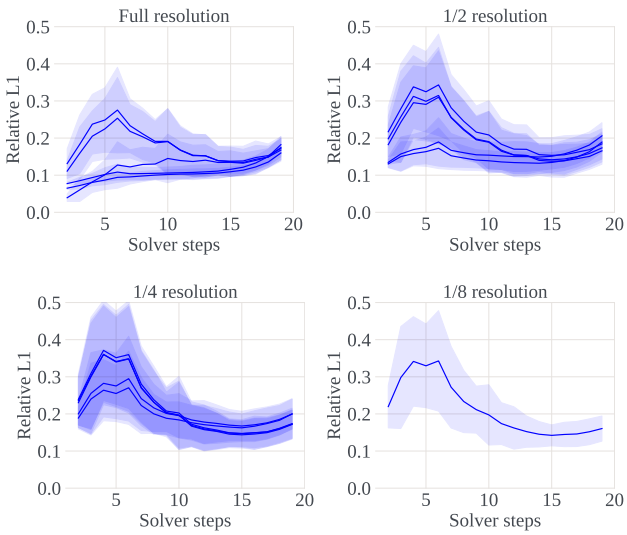
Text-to-Image Models. With recent advances in generative models, a vast number of text-conditioned models for image synthesis emerged. Starting out with GAN-based methods [14, 24, 35, 36, 40, 51, 53, 54, 58, 59, 64], researchers discovered important techniques such as adding self-attention layers [60] for better long-range dependency modeling and scaling up to very large architectures [3, 21]. Different autoencoder-based methods [16, 39], in particular generative transformers [5, 7, 12, 37], can also synthesize new images in a single forward pass and achieve high visual quality. Recently, the field has been dominated by diffusion models [47–49]. Advances such as classifier guidance [9], classifier-free guidance [18, 32], and diffusion in the latent space [41] have enabled modern diffusion models [1, 6, 8, 13, 32, 38, 41, 43, 55] to generate photorealistic images at high resolution from text. However, this superior performance often comes at a cost: Due to repeated applications of the underlying denoising neural network, image synthesis with diffusion models is very computationally expensive. This not only hinders their widespread usage in end-user products, but also slows down further research. To facilitate further democratization of diffusion models, we focus on accelerating diffusion models in this work.

Improved Solvers. In the diffusion model framework, we draw a new sample at every step from a distribution determined by the previous steps. The exact sampling strategy, defined by the so-called solver, plays an important role in determining the number of steps we have to make to obtain high-quality output. Starting out from the DDPM [19] formulation, DDIM [48] introduced implicit probabilistic models. DDIM allows the combination of DDPM steps without retraining and is popular with many current models. The DPM-Solver [27, 28] models the denoising process as an ordinary differential equation and proposes a dedicated high-order solver for diffusion ODEs. Similar approaches are adopted by [22, 25, 61–63]. Another line of works [10, 11, 23, 45, 52] proposed to train certain parts of the solver on a dataset. While better solvers can help to speed up image synthesis by reducing the number of required steps, they still treat the underlying neural network as a black box. In contrast, our work investigates the internal behavior of the neural network and gains speed up from caching. Therefore, the benefits of improved solvers and our caching strategy are not mutually exclusive.

Distillation. Distillation techniques present an alternative way to speed up inference. Here, a pretrained teacher net-



(a) Layer development during denoising.



(b) Per step change for all spatial transformer blocks ($L1_{rel}$).

Figure 2. **Overview.** We observe, that in diffusion models, not only the intermediate results x, but also the internal feature maps change smoothly over time. (a) We visualize output feature maps of two layer blocks within the denoising network via PCA. Structures change smoothly at different rates. (b) We also observe this smooth layer-wise change when plotting the change in output from one step to the next, averaging over many different prompts and randomly initialized noise. Besides the average, we also show the standard deviation as shaded area. The patterns always remain the same. *Configuration*: LDM-512, DPM, 20 Steps.

work creates new training targets for a student architecture, that needs fewer neural function evaluations than the teacher. Guidance distillation [31] replaces the two function evaluations of classifier-free guidance with a single one, while progressive distillation [44] reduces the number of sampling steps. [29] optimizes a student to directly predict the image generated by the teacher in one step.

Consistency models [30, 50] use a consistency formulation enabling a single-step student to do further steps. Finally, [56] distill a large teacher model into a much smaller student architecture. However, distillation does not come without cost. Apart from the computational cost of retraining the student model, some distillation techniques cannot handle negative or composite prompts [26, 31]. In this paper, we introduce a lightweight fine-tuning technique inspired by distillation, that leaves the original parameters unchanged while optimizing a small number of extra parameters without restricting the model.

3. Method

In this work, we investigate the behavior of the different layers in the diffusion U-Net to develop novel ways of speeding up the image generation process. The main insight of our method is that large latent diffusion models contain redundant computations that can be recycled between steps without compromising image quality. The key to our approach is to cache the outputs of U-Net blocks to be reused in the remaining diffusion steps.

3.1. Preliminaries

In the diffusion model framework, we start from an input image $x_0 \in [-1, 1]^{3\times H\times W}$. For a number of timesteps $t \in [1, T]$, we repeatedly add Gaussian noise $\epsilon_t \sim \mathcal{N}$ to the image, to gradually transform it into fully random noise.

$$x_t = x_{t-1} + \epsilon_t \tag{1}$$

$$x_T \sim \mathcal{N}(0, 1) \tag{2}$$

To synthesize novel images, we train a neural network $\Psi(x_t, t)$ to gradually *denoise* a random sample x_T. The neural network can be parameterized in different ways to predict x_0, ϵ_t or $\nabla \log(x_t)$ [49]. A solver Φ determines how to exactly compute x_{t-1} from the output of Ψ and t.

$$x_{t-1} = \Phi\left(x_t, t, \Psi\left(x_t, t\right)\right) \tag{3}$$

The higher the number of steps is, the higher the visual quality of the image generally becomes. Determining the number of steps presents users with a trade-off between image quality and speed.

3.2. Analysis

One of the key limitations of diffusion models is their slow inference speed. Existing works often propose new solvers or to distill existing models, so that fewer steps are required to produce high-quality images. However, both of these directions treat the given neural network as a black box.

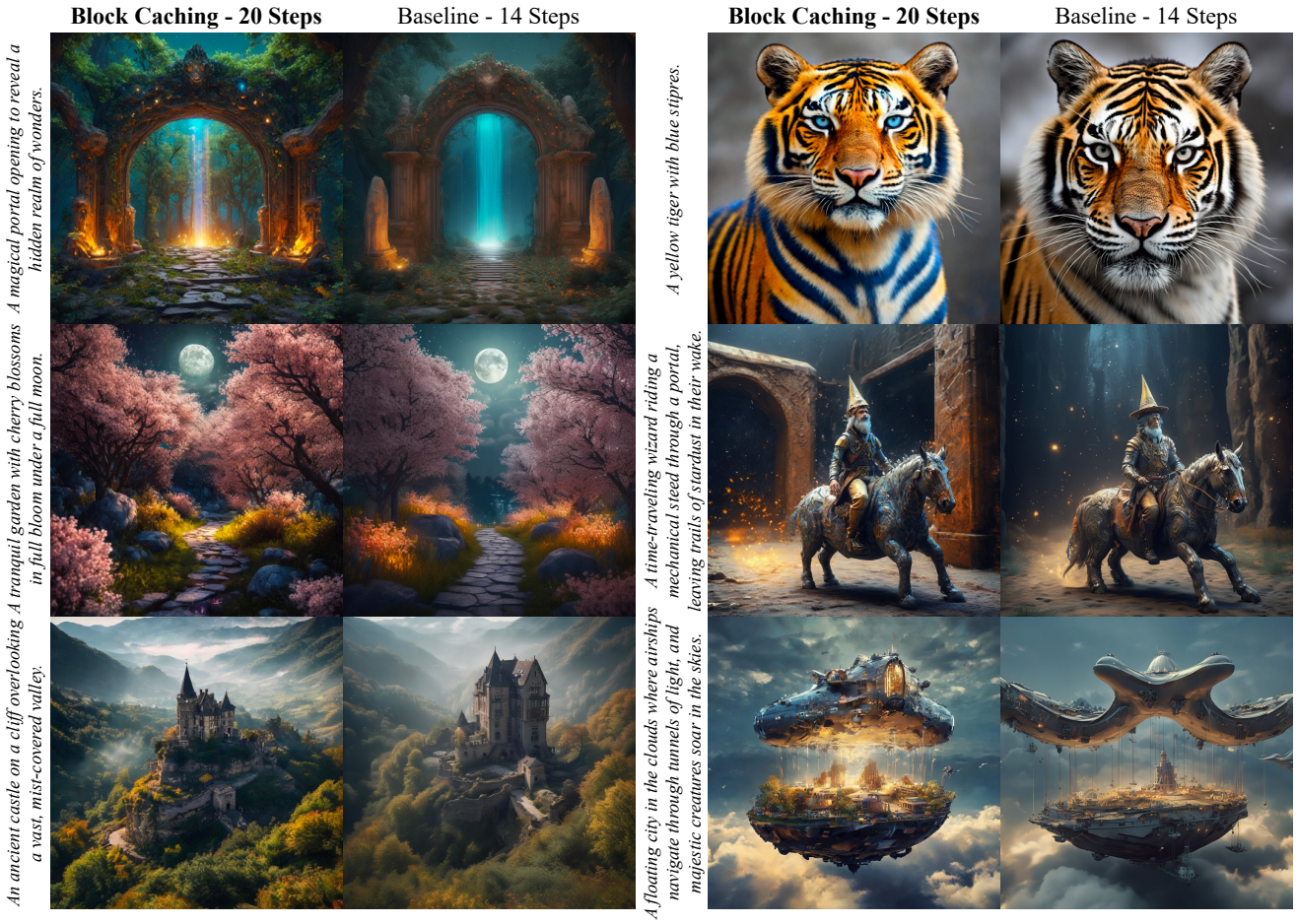


Figure 3. **Qualitative Results for EMU-768.** With identical inference speed, our caching technique produces finer details and more vibrant colors. For more results refer to the supplementary material. *Configuration*: DPM, Block caching with 20 steps vs Baseline with 14 steps.

In this paper, we move away from the "black box perspective" and investigate the *internal* behavior of the neural network Ψ to understand it at a per-layer basis. This is particularly interesting when considering the temporal component. To generate an image, we have to perform multiple forward passes, where the input to the network changes only gradually over time.

The neural network Ψ generally consists of multiple blocks of layers $B_i(x_i, s_i)$, $i \in [0, N-1]$, where N is the number of all blocks of the network, x is the output of an earlier block and s is the optional data from a skip connection. The common U-Net architecture [42], as used in many current works [8, 33, 41], is made up of `ResBlocks`, `SpatialTransformer` blocks, and up/downsampling blocks. `ResBlocks` mostly perform cheap convolutions, while `SpatialTransformer` blocks perform self- and cross-attention operations and are much more costly.

A common design theme of such blocks is that they rely on *residual connections*. Instead of simply passing the results of the layer computations to the next block, the result is combined with the original input of the current block via summation. This is beneficial, as it allows information (and gradients) to flow more freely through the network [15]. Rather than replacing the information, a block *changes* the information that it receives as input.

$$B_i(x, s) = C_i(x, s) + \text{concat}(x, s) \tag{4}$$
$$C_i(x, s) = \text{layers}_i(\text{concat}(x, s)) \tag{5}$$

To better understand the inner workings of the neural network, we visualize how much the changes the block applies to the input vary over time. Concretely, we consider two metrics: Relative absolute change L1_{rel}.

$$\text{L1}_{\text{rel}}(i, t) = \frac{||C_i(x_t, s_t) - C_i(x_{t-1}, s_{t-1})||_1}{||C_i(x_t, s_t)||_1} \tag{6}$$

To get representative results, we generate 32 images from different prompts with 2 random seeds each and report the averaged results in Fig. 2. Further, we visualize selected feature maps. We make three **key observations**:

1) Smooth change over time. Similarly to the intermediate images during denoising, the blocks change smoothly

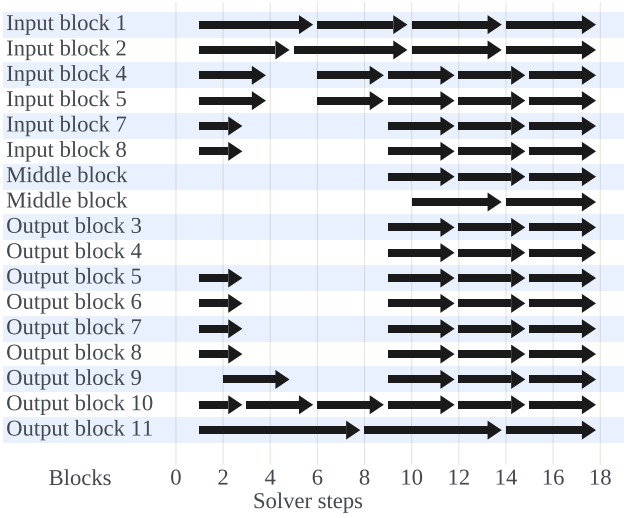


Figure 4. **Caching Schedule for LDM-512 at 20 steps with DPM.** Each arrow represents the cache lifetime of a spatial transformer block. For the duration of an arrow, the spatial transformer block reuses the cached result computed at the beginning of the arrow. E.g., *Input block 1* only computes the result at step 1, 6, 10, 14 and 18 and uses the cached value otherwise.

and gradually over time. This suggests that there is a clear temporal relation between the outputs of a block.

2) Distinct patterns of change. The different blocks do not behave uniformly over time. Rather, they apply a lot of change in certain periods of the denoising process, while they remain inactive in others. The standard deviation shows that this behavior is consistent over different images and random seeds. Note that some blocks, for example the blocks at higher resolutions (either very early or very late in the network) change most in the last 20%, while deeper blocks at lower resolutions change more in the beginning.

3) Small step-to-step difference. Almost every block has significant periods during the denoising process, in which its output only changes very little.

3.3. Block Caching

We hypothesize that a lot of blocks are performing redundant computations during steps where their outputs change very little. To reduce the amount of redundant computations and to speed up inference, we propose **Block Caching**.

Instead of computing new outputs at every step, we reuse the cached outputs from a previous step. Due to the nature of residual connections, we can perform caching at a per-block level without interfering with the flow of information through the network otherwise. We can apply our caching technique to almost all recent diffusion model architectures.

One of the major benefits of **Block Caching** compared to approaches that reduce the number of steps is that we have a more finegrained control over where we save computation. While we perform fewer redundant computations, we do not

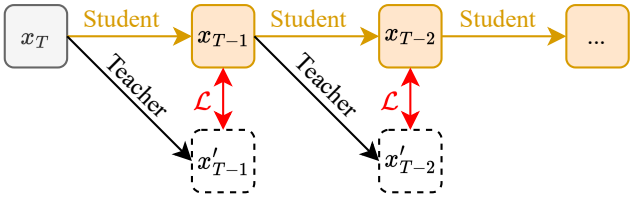


Figure 5. **Scale Shift Optimization.** The student network copies and freezes the weights of the teacher and has additional scale and shift parameters per block. These parameters are optimized to match the teacher output per block and step.

reduce the number of steps that require a lot of precision (*i.e.* where the change is high).

Automatic cache schedule. Not every block should be cached all the time. To make a more informed decision about when and where to cache, we rely on the metric described in Sec. 3.2. We first evaluate these metrics over a number of random prompts and seeds. Our intuition is that for any layer block i, we retain a cached value, which was computed at time step t_a, as long as the accumulated change does not exceed a certain threshold δ. Once the threshold is exceeded at time step t_b, we recompute the block's output.

$$\sum_{t=t_a}^{t_b-1} \mathrm{L1}_{\mathrm{rel}}(i,t) \leq \delta < \sum_{t=t_a}^{t_b} \mathrm{L1}_{\mathrm{rel}}(i,t) \tag{7}$$

With a lower threshold, the cached values will be refreshed more often, whereas a higher threshold will lead to faster image generation but will affect the appearance of the image more. The threshold δ can be picked such that it increases inference speed without negatively affecting image quality.

3.4. Scale-Shift Adjustment

While caching already works surprisingly well on its own, as shown in Sec. 4.2, we observe that aggressive caching can introduce artifacts into the final image. We hypothesize that this is due to a misalignment between the cached feature map and the "original" feature map at a given timestep. To enable the model to adjust to using cached values, we introduce a very lightweight **scale-shift adjustment mechanism** wherever we apply caching. To this end, we add a timestep-dependent scalar shift and scale parameter for each layer that receives a cached input. Concretely, we consider every channel separately, *i.e.* for a feature map of shape $(N \times C \times H \times W)$, we predict a vector of shape $(N \times C)$ for both scale and shift. This corresponds to a simple linear layer that receives the timestep embedding as input.

We optimize scale and shift on the training set while keeping all other parameters frozen. However, optimization of these additional parameters is not trivial. As we require

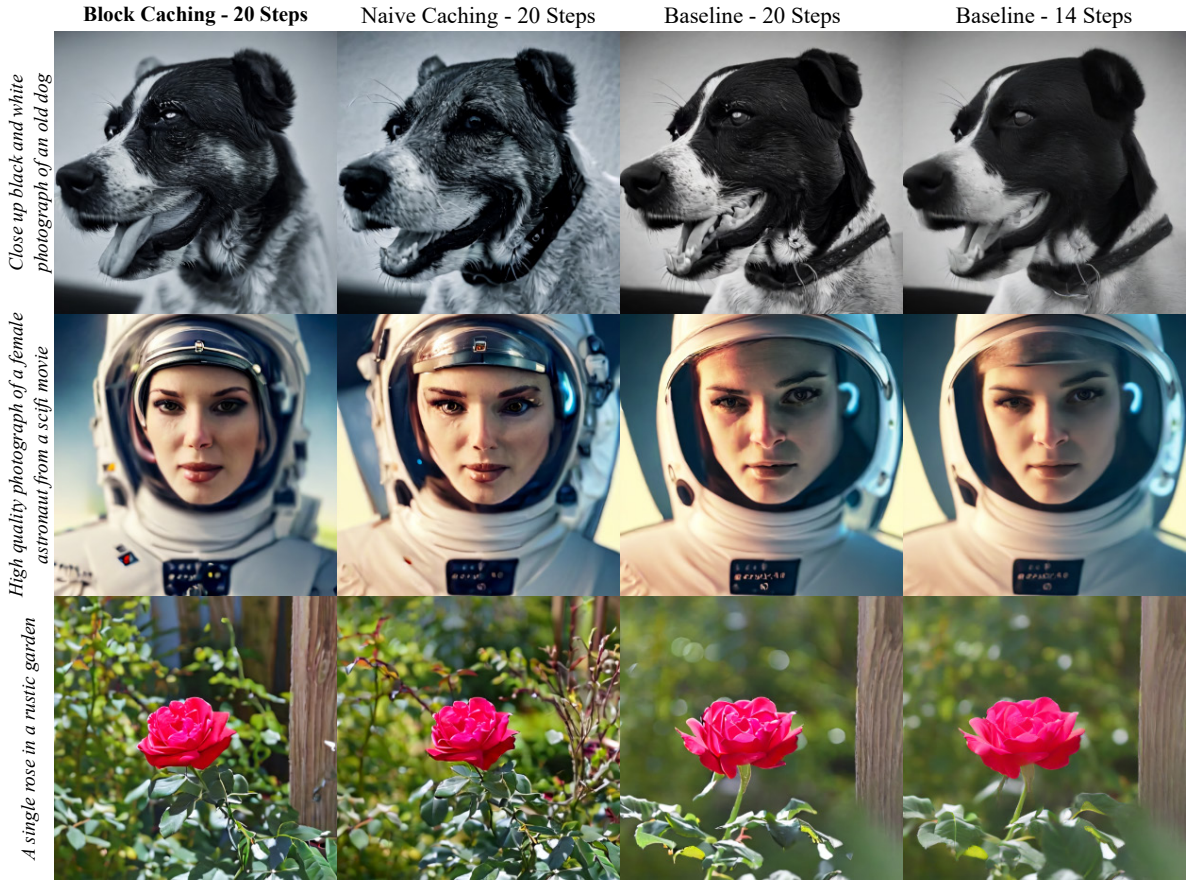


Figure 6. **Qualitative Results for LDM-512.** Our method often provides richer colors and finer details. Through our scale-shift adjustment, we avoid artifacts that are visible when naively applying block caching. More qualitative results for DPM and DDIM can be found in the supplementary material. *Configuration*: DPM, Block caching with 20 steps vs Baseline with 14 steps.

valid cached values, we cannot directly add noise to an image and train the network to denoise to the original image.

Therefore, we rely on an approach, shown in Fig. 5, that is inspired by distillation techniques. Our model with caching enabled acts as the student, while the same model with caching disabled acts as the teacher. We first unroll the consecutive steps of the denoising process for the student configuration and generate an image from complete noise. Then, we perform a second forward pass at every timestep with the teacher configuration, which acts as the training target. Note that for the teacher, we use the intermediate steps from the student's trajectory as input rather than unrolling the teacher. Otherwise, the teacher might take a different trajectory (leading to a different final output), which then is not useful as a training target.

This optimization is very resource-friendly, as the teacher and student can use the same weights, saving GPU memory, and we only optimize a small number of extra parameters, while keeping the parameters of the original model the same. During inference, the multiplication and addition with scale and shift parameters have no noticeable effect on the inference speed but improve image quality as shown in Sec. 4.2.

4. Experiments

In the following, we first demonstrate the general potential of our **Block Caching** technique and then analyze it in more detail through several ablation studies.

4.1. Experimental Setup

Our proposed method is general and can be applied to most recent diffusion models. In order to give a good overview, we conduct our experiments mainly on two models that represent light and heavy computational demands:

- **LDM-512** [41], a popular diffusion model with 900M parameters, that generates images at a 512×512 resolution, replacement trained on internal Shutterstock images.
- **EMU-768** [8], a state-of-the-art model with 2.7B parameters, which can produce photorealistic images at a resolution of 768×768.

For both models, we use classifier-free guidance [18] with a guidance strength of 5.0 and do not use any other performance-enhancing techniques. We run inference in bfloat16 type and measure the latency on a single Nvidia A100 GPU. For the optimization of the scale-shift adjustment parameters, we perform 15k training iterations on eight A100 GPUs. Depending on the model and the number of denoising steps, this takes between 12 and 48 hours.

4.2. Accelerating Inference through Caching

Our proposed caching technique can be viewed from two perspectives: 1) Given a fixed number of steps, caching allows us to accelerate the image generation process without decreasing quality. 2) Given a fixed computational budget, we can perform more steps when using caching, and therefore obtain better image quality than performing fewer steps without caching.

To demonstrate the flexibility of our approach, we consider two common inference settings: (i) Many approaches perform 50 denoising steps by default. Therefore, we apply caching with 50 solver steps and achieve the same latency as the 30 steps of the baseline model. (ii) By using modern solvers like DPM [27] or DDIM [48], it is possible to generate realistic-looking images with as few as 20 steps. If we apply caching with 20 solver steps, we can reduce the inference latency to an equivalent of performing 14 steps with the non-cached baseline model.

Analysis of LDM-512. We begin by performing a thorough qualitative and quantitative analysis of the LDM-512 model. After computing the layer block statistics for the automatic cache configuration, we find that a change threshold of $\delta = 0.5$ gives us the desired speedup. The resulting caching schedule is visualized in Fig. 4. As can be observed in the plots with relative feature changes (Fig. 2), we can aggressively cache the early and late blocks. On the other hand, the activations of the deeper blocks change faster, especially in the first half of the denoising process, and should therefore only be cached conservatively.

The results in Tab. 1 demonstrate that for both DPM and DDIM, the proposed caching with 20 steps significantly improves the FID value compared to the 14-step baseline, while being slightly faster. Similarly, 50 steps with caching outperforms the 30-step baseline, while maintaining a comparable latency. Moreover, our scale-shift adjustment mechanism further enhances the results. Notably, this full configuration even outperforms the 20-step and 50-step baselines. We hypothesize that caching introduces a slight momentum in the denoising trajectory due to the delayed updates in cached values, resulting in more pronounced features in the final output image.

Qualitative results can be seen in Fig. 6. Our full model (caching + scale-shift adjustment) produces more crisp and vibrant images with significantly more details when compared to the 14-step baseline. This can be explained by the fact that when performing only 14 steps, the model makes steps that are too big to add meaningful details to the image. Caching without scale-shift adjustment also yields images with more detail compared to the baseline. However, we often observe local artifacts, which are particularly noticeable in the image backgrounds. These artifacts appear like overly-emphasized style features. The application of our scale-shift adjustment effectively mitigates these effects.

Solver	Steps	Caching	SS	FID ↓	Img/s ↑	Speedup ↑
DPM [27]	20			17.15	2.17	1.00×
	14			18.67	3.10	1.43×
	20	✓		17.58	**3.64**	**1.68**×
	20	✓	✓	**15.95**	3.59	1.65×
DDIM [48]	20			17.43	2.17	1.00×
	14			17.11	3.10	1.43×
	20	✓		16.52	**3.48**	**1.60**×
	20	✓	✓	**16.02**	3.45	1.58×
DPM [27]	50			17.44	0.87	1.00×
	30			17.21	1.46	1.67×
	50	✓		17.23	**1.61**	**1.85**×
	50	✓	✓	**15.18**	1.59	1.82×
DDIM [48]	50			17.76	0.87	1.00×
	30			17.42	1.46	1.67×
	50	✓		16.65	**1.59**	**1.82**×
	50	✓	✓	**15.15**	1.56	1.79×

Table 1. **LDM-512 FID and Throughput Measurements.** For different solvers, we test our caching technique against baselines with 1) the same number of steps or 2) the same latency. In all cases, our proposed approach achieves significant speedup while improving visual quality as measured by FID on a COCO subset removing all faces. **Legend**: SS = Scale-shift adjustment, Img/s. = Images per second.

Solver	Steps *(Img/s)*		Votes (in %)		
	Caching	Baseline	Win	Tie	Lose
DPM	20 (0.28)	14 (0.27)	34.7	36.9	28.4
DDIM	20 (0.28)	14 (0.27)	28.0	48.8	23.2
DPM	50 (0.14)	30 (0.13)	27.8	54.3	17.9
DDIM	50 (0.13)	30 (0.13)	29.7	46.8	23.5

Table 2. **EMU-768 Visual Appeal Human Evaluation.** We present the percentages of votes indicating a win, tie, or loss for our method in comparison to the baseline. This is evaluated across various solvers and number of steps. In every comparison, both the caching and baseline configuration have roughly the same inference speed (reported as images per second).

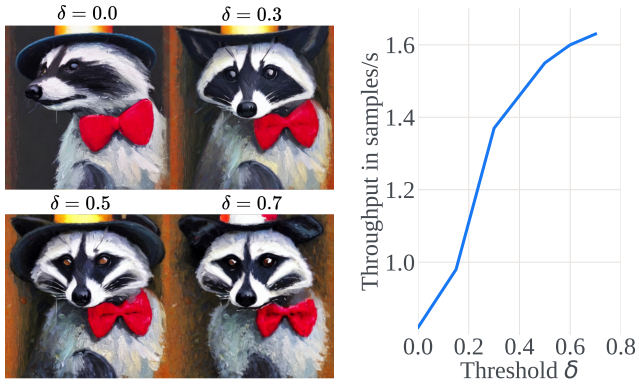


Figure 7. **Effect of Cache Threshold δ.** Left: Generated image for different δ. Right: Inference speed vs. δ. The higher δ, the more blocks are cached, resulting in faster inference. $\delta = 0.5$ gives a 1.5x speedup and the best visual quality. *Configuration*: DPM, LDM-512, Block caching with 50 steps.

Analysis of EMU-768. To demonstrate the generality of our proposed approach, we also apply caching and scale-shift adjustment to the EMU-768 model under the same settings as for LDM-512. As can be seen in Fig. 3, we achieve a very similar effect: The generated images are much more detailed and more vibrant, compared to the baseline. This is also confirmed by a human eval study, in which we asked 12 independent annotators to compare the visual appeal of images generated for the prompts from Open User Input (OUI) Prompts [8] and PartiPrompts[57] for different configurations. Specifically, we compared different configurations with the same latency for different samplers and collected 1320 votes in total. As reported in Tab. 2, our proposed caching technique is clearly preferred over the baseline in every run. Note that for many prompts, both images have very high quality, leading to a high rate in ties. This study shows that caching can be applied to a wide range of different models, samplers and step counts.

Effects of more aggressive caching. The extent to which the model caches results is controlled by the parameter δ. The higher δ, the longer the cache lifetime and the less frequent block outputs are recomputed. Fig. 7 shows synthesized images for varying δ values along with the corresponding inference speed. Although a higher δ leads to faster inference, the quality of the final image deteriorates when block outputs are recomputed too infrequently. We find that $\delta = 0.5$ not only provides a significant speedup by $1.5\times$ but also improves the image quality, thereby achieving the optimal trade-off (see Tab. 1).

Difficulty of Caching ResBlocks. As described above, we only cache `SpatialTransformer` blocks and not `ResBlocks`. This design choice is grounded in the observation, that `ResBlocks` change much less smoothly compared to `SpatialTransformer` blocks. In particular, `ResBlocks` are very important for generating local details in the image. To test this, we generate images where we only cache `ResBlocks` and leave `SpatialTransformer` blocks untouched. As can be seen in Fig. 8, even to gain a speedup of as low as 5%, the image quality deteriorates significantly.

Baseline - 20 Steps ResBlock Caching - 20 Steps

Figure 8. **Effect of Caching ResBlocks.** Caching ResBlocks instead of spatial transformer blocks results in fewer details and inferior image quality, while achieving only a small speedup of 5%. *Configuration*: DPM, EMU-768, Block caching with 20 steps.

5. Conclusion

In this paper, we first analyzed the inner workings of the denoising network, moving away from the common perspective of considering diffusion models as black boxes. Leveraging the insights from our analysis, we proposed the **Block Caching** technique. It reduces the redundant computations during inference of the diffusion models and significantly speeds up the image generation process by a factor of **1.5**$\times$-**1.8**$\times$ at a minimal loss of image quality. To showcase the adaptability of our approach, we performed experiments on LDM and EMU models with a parameter range from 900M to 2.7B. We tested our approach in different inference settings by varying solvers and number of steps. Our technique generates more vibrant images with more fine-grained details when compared to naively reducing the number of solver steps for the baseline model to match the compute budget. We confirmed our findings quantitatively by computing the FID and by human evaluation.

References

[1] Yogesh Balaji, Seungjun Nah, Xun Huang, Arash Vahdat, Jiaming Song, Karsten Kreis, Miika Aittala, Timo Aila, Samuli Laine, Bryan Catanzaro, et al. ediffi: Text-to-image diffusion models with an ensemble of expert denoisers. *arXiv:2211.01324*, 2022. 2

[2] James Betker, Gabriel Goh, Li Jing, Tim Brooks, Jianfeng Wang, Linjie Li, Long Ouyang, Juntang Zhuang, Joyce Lee, Yufei Guo, Wesam Manassra, Prafulla Dhariwal, Casey Chu, Yunxin Jiao, and Aditya Ramesh. Improving image generation with better captions. *OpenAI*, 2023. 1

[3] Andrew Brock, Jeff Donahue, and Karen Simonyan. Large scale gan training for high fidelity natural image synthesis. In *ICLR*, 2018. 2

[4] Tim Brooks, Aleksander Holynski, and Alexei A. Efros. Instructpix2pix: Learning to follow image editing instructions. In *CVPR*, 2023. 2

[5] Huiwen Chang, Han Zhang, Lu Jiang, Ce Liu, and William T Freeman. Maskgit: Masked generative image transformer. In *CVPR*, pages 11315–11325, 2022. 2

[6] Junsong Chen, Jincheng Yu, Chongjian Ge, Lewei Yao, Enze Xie, Yue Wu, Zhongdao Wang, James Kwok, Ping Luo, Huchuan Lu, et al. Pixart-α: Fast training of diffusion transformer for photorealistic text-to-image synthesis. *arXiv:2310.00426*, 2023. 2

[7] Mark Chen, Alec Radford, Rewon Child, Jeffrey Wu, Heewoo Jun, David Luan, and Ilya Sutskever. Generative pretraining from pixels. In *ICML*, pages 1691–1703. PMLR, 2020. 2

[8] Xiaoliang Dai, Ji Hou, Chih-Yao Ma, Sam Tsai, Jialiang Wang, Rui Wang, Peizhao Zhang, et al. Emu: Enhancing image generation models using photogenic needles in a haystack. *arXiv:2309.15807*, 2023. 1, 2, 4, 6, 8

[9] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. *NeurIPS*. 2

[10] Tim Dockhorn, Arash Vahdat, and Karsten Kreis. Genie: Higher-order denoising diffusion solvers. *NeurIPS*, 35: 30150–30166, 2022. 2

[11] Zhongjie Duan, Chengyu Wang, Cen Chen, Jun Huang, and Weining Qian. Optimal linear subspace search: Learning to construct fast and high-quality schedulers for diffusion models. *arXiv:2305.14677*, 2023. 2

[12] Patrick Esser, Robin Rombach, and Bjorn Ommer. Taming transformers for high-resolution image synthesis. In *CVPR*, pages 12873–12883, 2021. 2

[13] Zhida Feng, Zhenyu Zhang, Xintong Yu, Yewei Fang, Lanxin Li, Xuyi Chen, Yuxiang Lu, Jiaxiang Liu, Weichong Yin, et al. Ernie-vilg 2.0: Improving text-to-image diffusion model with knowledge-enhanced mixture-of-denoising-experts. In *CVPR*, pages 10135–10145, 2023. 2

[14] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *NeurIPS*, 2014. 2

[15] Moritz Hardt and Tengyu Ma. Identity matters in deep learning. *arXiv:1611.04231*, 2016. 4

[16] Kaiming He, Xinlei Chen, Saining Xie, Yanghao Li, Piotr Dollár, and Ross Girshick. Masked autoencoders are scalable vision learners. In *CVPR*, pages 16000–16009, 2022. 2

[17] Amir Hertz, Ron Mokady, Jay Tenenbaum, Kfir Aberman, Yael Pritch, and Daniel Cohen-Or. Prompt-to-prompt image editing with cross attention control. 2022. 2

[18] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. In *NeurIPS 2021 Workshop on Deep Generative Models and Downstream Applications*, 2021. 2, 7

[19] Jonathan Ho, Ajay Jain, and Pieter Abbeel. Denoising diffusion probabilistic models. *NeurIPS*, 2020. 2

[20] Anthony Hu, Lloyd Russell, Hudson Yeo, Zak Murez, George Fedoseev, Alex Kendall, Jamie Shotton, and Gianluca Corrado. Gaia-1: A generative world model for autonomous driving. *arXiv:2309.17080*, 2023. 2

[21] Minguk Kang, Jun-Yan Zhu, Richard Zhang, Jaesik Park, Eli Shechtman, Sylvain Paris, and Taesung Park. Scaling up gans for text-to-image synthesis. In *CVPR*, pages 10124–10134, 2023. 2

[22] Tero Karras, Miika Aittala, Timo Aila, and Samuli Laine. Elucidating the design space of diffusion-based generative models. *NeurIPS*, 35:26565–26577, 2022. 2

[23] Max WY Lam, Jun Wang, Rongjie Huang, Dan Su, and Dong Yu. Bilateral denoising diffusion models. *arXiv:2108.11514*, 2021. 2

[24] Bowen Li, Xiaojuan Qi, Thomas Lukasiewicz, and Philip Torr. Controllable text-to-image generation. *NeurIPS*, 2019. 2

[25] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. In *ICLR*, 2022. 2

[26] Nan Liu, Shuang Li, Yilun Du, Antonio Torralba, and Joshua B Tenenbaum. Compositional visual generation with composable diffusion models. In *ECCV*, pages 423–439. Springer, 2022. 3

[27] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver: A fast ode solver for diffusion probabilistic model sampling in around 10 steps. In *NeurIPS*, pages 5775–5787, 2022. 2, 7

[28] Cheng Lu, Yuhao Zhou, Fan Bao, Jianfei Chen, Chongxuan Li, and Jun Zhu. Dpm-solver++: Fast solver for guided sampling of diffusion probabilistic models. *arXiv:2211.01095*, 2022. 2

[29] Eric Luhman and Troy Luhman. Knowledge distillation in iterative generative models for improved sampling speed. *arXiv:2101.02388*, 2021. 3

[30] Simian Luo, Yiqin Tan, Longbo Huang, Jian Li, and Hang Zhao. Latent consistency models: Synthesizing high-resolution images with few-step inference, 2023. 3

[31] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *CVPR*, pages 14297–14306, 2023. 2, 3

[32] Alexander Quinn Nichol, Prafulla Dhariwal, Aditya Ramesh, Pranav Shyam, Pamela Mishkin, Bob Mcgrew, Ilya Sutskever, and Mark Chen. Glide: Towards photorealistic image generation and editing with text-guided diffusion models. In *ICML*, pages 16784–16804. PMLR, 2022. 2

[33] Dustin Podell, Zion English, Kyle Lacey, Andreas Blattmann, Tim Dockhorn, Jonas Müller, Joe Penna, and Robin Rombach. Sdxl: improving latent diffusion models for high-resolution image synthesis. *arXiv:2307.01952*, 2023. 4

[34] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. Dreamfusion: Text-to-3d using 2d diffusion. In *ICLR*, 2023. 2

[35] Tingting Qiao, Jing Zhang, Duanqing Xu, and Dacheng Tao. Mirrorgan: Learning text-to-image generation by redescription. In *CVPR*, pages 1505–1514, 2019. 2

[36] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *arXiv:1511.06434*, 2015. 2

[37] Aditya Ramesh, Mikhail Pavlov, Gabriel Goh, Scott Gray, Chelsea Voss, Alec Radford, Mark Chen, and Ilya Sutskever. Zero-shot text-to-image generation. In *ICML*, pages 8821–8831. PMLR, 2021. 2

[38] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. Hierarchical text-conditional image generation with clip latents. *arXiv:2204.06125*, 1(2):3, 2022. 1, 2

[39] Ali Razavi, Aaron Van den Oord, and Oriol Vinyals. Generating diverse high-fidelity images with vq-vae-2. *NeurIPS*, 32, 2019. 2

[40] Scott Reed, Zeynep Akata, Xinchen Yan, Lajanugen Logeswaran, Bernt Schiele, and Honglak Lee. Generative adversarial text to image synthesis. In *ICML*, pages 1060–1069. PMLR, 2016. 2

[41] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, pages 10684–10695, 2022. 1, 2, 4, 6

[42] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, pages 234–241. Springer, 2015. 4

[43] 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. *NeurIPS*, 35:36479–36494, 2022. 1, 2

[44] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. In *ICLR*, 2021. 2, 3

[45] Neta Shaul, Juan Perez, Ricky TQ Chen, Ali Thabet, Albert Pumarola, and Yaron Lipman. Bespoke solvers for generative flow models. *arXiv:2310.19075*, 2023. 2

[46] Uriel Singer, Adam Polyak, Thomas Hayes, Xi Yin, Jie An, Songyang Zhang, Qiyuan Hu, Harry Yang, Oron Ashual, Oran Gafni, et al. Make-a-video: Text-to-video generation without text-video data. In *ICLR*, 2023. 2

[47] Jascha Sohl-Dickstein, Eric Weiss, Niru Maheswaranathan, and Surya Ganguli. Deep unsupervised learning using nonequilibrium thermodynamics. In *ICML*. PMLR, 2015. 2

[48] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. In *ICLR*, 2020. 2, 7

[49] Yang Song, Jascha Sohl-Dickstein, Diederik P Kingma, Abhishek Kumar, Stefano Ermon, and Ben Poole. Score-based generative modeling through stochastic differential equations. In *ICLR*, 2020. 2, 3

[50] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. *arXiv:2303.01469*, 2023. 3

[51] Ming Tao, Hao Tang, Fei Wu, Xiao-Yuan Jing, Bing-Kun Bao, and Changsheng Xu. Df-gan: A simple and effective baseline for text-to-image synthesis. In *CVPR*, pages 16515–16525, 2022. 2

[52] Daniel Watson, William Chan, Jonathan Ho, and Mohammad Norouzi. Learning fast samplers for diffusion models by differentiating through sample quality. In *ICLR*, 2021. 2

[53] Weihao Xia, Yujiu Yang, Jing-Hao Xue, and Baoyuan Wu. Tedigan: Text-guided diverse face image generation and manipulation. In *CVPR*, pages 2256–2265, 2021. 2

[54] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. Attngan: Fine-grained text to image generation with attentional generative adversarial networks. In *CVPR*, pages 1316–1324, 2018. 2

[55] Zeyue Xue, Guanglu Song, Qiushan Guo, Boxiao Liu, Zhuofan Zong, Yu Liu, and Ping Luo. Raphael: Text-to-image generation via large mixture of diffusion paths. *arXiv:2305.18295*, 2023. 2

[56] Xingyi Yang, Daquan Zhou, Jiashi Feng, and Xinchao Wang. Diffusion probabilistic model made slim. In *CVPR*, pages 22552–22562, 2023. 3

[57] Jiahui Yu, Yuanzhong Xu, Jing Yu Koh, Thang Luong, Gunjan Baid, Zirui Wang, Vijay Vasudevan, Alexander Ku, Yinfei Yang, Burcu Karagol Ayan, et al. Scaling autoregressive models for content-rich text-to-image generation. *arXiv:2206.10789*, 2(3):5, 2022. 8

[58] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *ICCV*, pages 5907–5915, 2017. 2

[59] Han Zhang, Tao Xu, Hongsheng Li, Shaoting Zhang, Xiaogang Wang, Xiaolei Huang, and Dimitris N Metaxas. Stackgan++: Realistic image synthesis with stacked generative adversarial networks. *IEEE TPAMI*, 41(8):1947–1962, 2018. 2

[60] Han Zhang, Ian Goodfellow, Dimitris Metaxas, and Augustus Odena. Self-attention generative adversarial networks. In *ICML*, pages 7354–7363. PMLR, 2019. 2

[61] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. In *NeurIPS 2022 Workshop on Score-Based Methods*, 2022. 2

[62] Qinsheng Zhang, Jiaming Song, and Yongxin Chen. Improved order analysis and design of exponential integrator for diffusion models sampling. *arXiv:2308.02157*, 2023.

[63] Wenliang Zhao, Lujia Bai, Yongming Rao, Jie Zhou, and Jiwen Lu. Unipc: A unified predictor-corrector framework for fast sampling of diffusion models. *NeurIPS*, 2023. 2

[64] Minfeng Zhu, Pingbo Pan, Wei Chen, and Yi Yang. Dm-gan: Dynamic memory generative adversarial networks for text-to-image synthesis. In *CVPR*, pages 5802–5810, 2019. 2