

# CacheQuant: Comprehensively Accelerated Diffusion Models

## Supplementary Material

### 7. Detailed experimental implementations

We use the pre-training models of DiT-XL/2<sup>1</sup>, DDIMs<sup>2</sup>, and LDMs<sup>3</sup> from the official website. For Stable Diffusion, we use the CompVis codebase<sup>4</sup> and its v1.4 checkpoint. The conditional generation models consist of a diffusion model and a decoder model. Like the previous works [27, 38, 74], we focus only on the diffusion model and does not quantize the decoder model. In the reconstruction training, we set the calibration samples to 1024 and the training batch to 32 for all experiments. However, for the Stable Diffusion, we adjust the reconstruction calibration samples to 512 and the training batch to 4 due to time and memory source constraints. We use open-source tool *pytorch-OpCounter*<sup>5</sup> to calculate the Size and Bops of models before and after quantization. And following the quantization setting, we only calculate the diffusion model part, not the decoder and encoder parts. We use the ADM’s TensorFlow evaluation suite *guided-diffusion*<sup>6</sup> to evaluate FID and IS, and use the open-source code *clip-score*<sup>7</sup> to evaluate CLIP scores. The accelerated diffusion models are deployed by utilizing CUTLASS<sup>8</sup> and PyTorch<sup>9</sup>. The speed up ratio is calculated by measuring the time taken to generate a single image on the RTX 3090. As per the standard practice [37, 38, 50], we employ the zero-shot approach to evaluate Stable Diffusion on COCO-val, resizing the generated 512 × 512 images and validation images in 300 × 300 with the center cropping to evaluate FID and IS score.

### 8. Express $X_{cq}W_q$ with two correction methods

Based on Eq. 8 and Eq. 9, the direct correction simply expresses  $X_{cq}W_q$  as:

$$X_{cq}W_q = \frac{X_g W_g}{a} - \frac{b}{a} \quad (15)$$

Our method corrects for  $X_c$  and correct  $O_{cq}$ , respec-

Table 6. Results of LDM-4 on ImageNet with 250 DDIM steps.

	Method	Retrain	Speed ↑	FID ↓
Cache	Deepcache- $N=12$	✗	7.58×	6.35
	Deepcache- $N=25$	✗	11.71×	9.51
	Deepcache- $N=35$	✗	13.75×	12.32
	Deepcache- $N=50$	✗	15.28×	26.63
Quantization	EDA-DM (W8A8)	✓	1.91×	4.13
	EDA-DM (W4A8)	✓	1.91×	4.13
	EDA-DM (W4A4)	✓	3.35×	44.12
Ours	CacheQuant- $N=5$ (W8A8)	✗	7.87×	4.03
	CacheQuant- $N=10$ (W8A8)	✗	12.20×	4.68
	CacheQuant- $N=15$ (W8A8)	✗	16.65×	5.51
	CacheQuant- $N=20$ (W8A8)	✗	18.06×	7.21

Table 7. Results of Stable Diffusion on PartiPrompt with 50 PLMS steps.

	Method	Retrain	Speed ↑	CLIP score ↑
PLMS	PLMS-35 steps	✗	1.47×	27.14
	PLMS-25 steps	✗	2.04×	27.10
	PLMS-20 steps	✗	2.46×	27.04
	PLMS-15 steps	✗	3.07×	26.82
	PLMS-10 steps	✗	4.32×	25.92
	CacheQuant- $N=2$ (W8A8)	✗	3.13×	27.19
Ours	CacheQuant- $N=3$ (W8A8)	✗	3.91×	27.18
	CacheQuant- $N=5$ (W8A8)	✗	5.20×	27.05
	CacheQuant- $N=6$ (W8A8)	✗	5.55×	27.05
	CacheQuant- $N=8$ (W8A8)	✗	5.85×	26.88

tively. Based on Eq. 8 and Eq. 9, derive the equation:

$$X_c = \frac{X_g}{a_1} - \frac{b_1}{a_1} \quad (16)$$

$$X_{cq}W_q = \frac{X_c W_g}{a_2} - \frac{b_2}{a_2} \quad (17)$$

Furthermore, the expression for  $X_{cq}W_q$  is as:

$$X_{cq}W_q = \frac{X_g}{a_1} \cdot \frac{W_g}{a_2} - \frac{b_1}{a_1} \cdot \frac{W_g}{a_2} - \frac{b_2}{a_2} \quad (18)$$

Since the correction parameters  $(a, b) \in \mathbb{R}^{C^o}$  and  $(a_1, b_1) \in \mathbb{R}^{C^i}$ ,  $(a_2, b_2) \in \mathbb{R}^{C^o}$ , the two representations of  $X_{cq}W_q$  are equivalent if and only if  $a_1 = 1$  and  $b_1 = 0$ .

### 9. Experimental settings for evaluation of acceleration-vs-performance tradeoff

We evaluate the tradeoff between acceleration and performance for various approaches in Sec 5.4. The detail experimental settings and results in Figure 6 are shown in Table 6 and 7.

<sup>1</sup><https://github.com/facebookresearch/DiT>

<sup>2</sup><https://github.com/ermongroup/ddim>

<sup>3</sup><https://github.com/CompVis/latent-diffusion>

<sup>4</sup><https://github.com/CompVis/stable-diffusion>

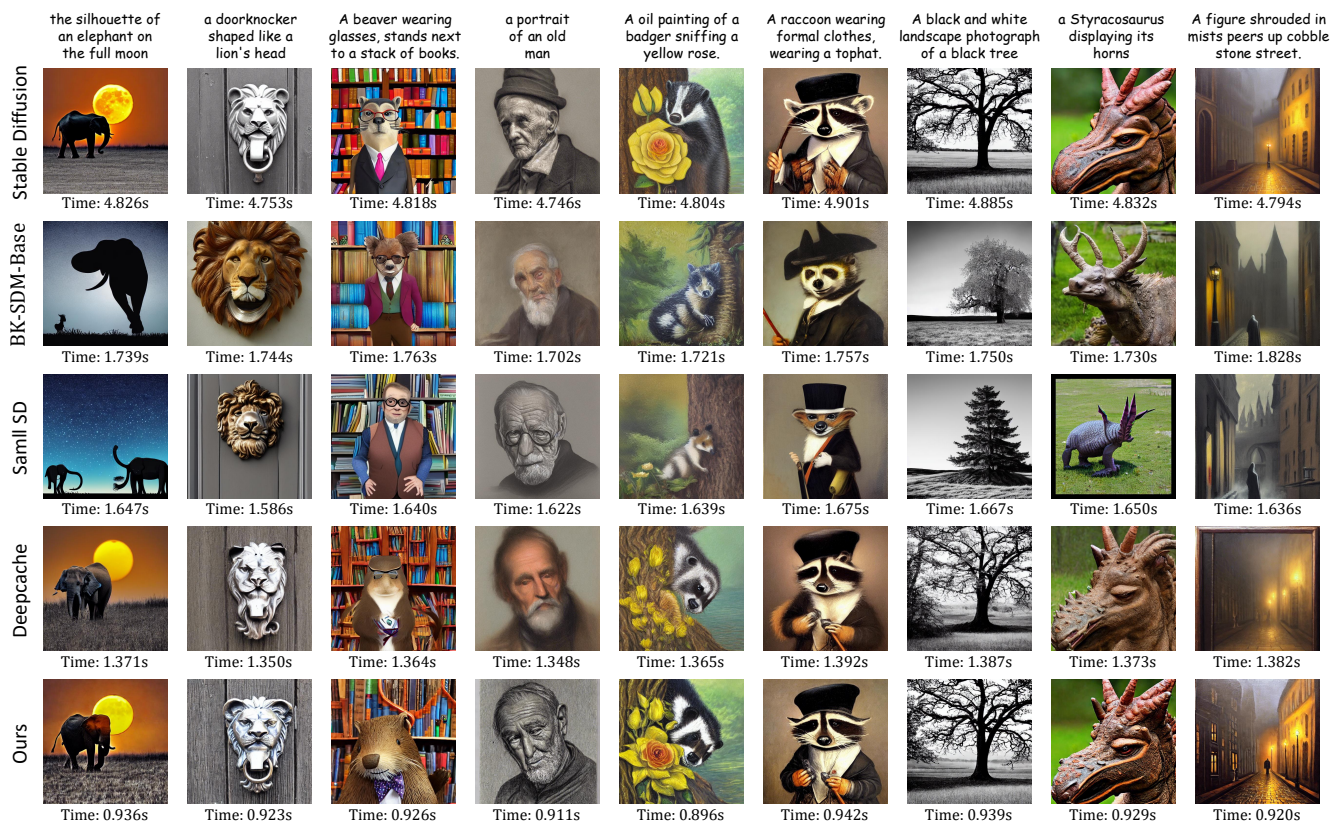
<sup>5</sup><https://github.com/Lyken17/pytorch-OpCounter>

<sup>6</sup><https://github.com/openai/guided-diffusion>

<sup>7</sup><https://github.com/Taited/clip-score>

<sup>8</sup><https://github.com/NVIDIA/cutlass>

<sup>9</sup><https://pytorch.org/blog/quantization-in-practice/>

Figure 9. Visualization of the generated images by  $\Delta$ -DiT and CacheQuant, with  $N=2$  cache frequency.Figure 10. Visualization of the generated images by BK-SDM-Base, Small SD, Deepcache with  $N=10$  cache frequency, and CacheQuant. All the methods adopt the 50-step PLMS. The time here is the duration to generate a single image.

893

## 10. Comparison of generated results

894

895

896

897

898

Within this section, we present random samples derived from original models and other accelerated methods with a fixed random seed. Our method maintains 8-bit precision. We visualize the generated image quality and latency of different methods in Figures 9 and 10.

## 11. Limitations and future work

899

900

901

902

903

904

While CacheQuant achieves remarkable results in a training-free manner at 8-bit precision, it relies on reconstruction to recovery performance at W4A8 precision. In the future, we will further refine CacheQuant to improve its compatibility with W4A8 precision.