PermuQuant: Lowering Per-Group Quantization Error by Reordering Channels for Diffusion Models

Yongsen Cheng[1]*, **Kai Liu**[1]*, **Kaiwen Tao**[1], **Junxian Li**[1], **Zhixin Wang**[2], **Zhikai Chen**[2], **Renjing Pei**[2], **Yulun Zhang**[1]†
[1]Shanghai Jiao Tong University, [2]Huawei Noah's Ark Lab



Figure 1: **PermuQuant** is a post-training quantization framework for low-bit diffusion models. In the W4A4 setting, it achieves 3.5× DiT memory reduction and 6.3× speedup on a single RTX 5090 32GB GPU by eliminating CPU offloading. In the challenging W3A3 setting, PermuQuant still produces visually clean results with faithful details, significantly outperforming other baselines.

Abstract

Large-scale visual generative models have achieved remarkable performance. However, their high computational and memory costs make deployment challenging in resource-constrained scenarios, such as interactive applications and personal single-GPU usage. Post-training quantization (PTQ) offers a practical solution by compressing pretrained models without expensive retraining. However, existing PTQ methods still suffer from severe quality degradation under extremely low-bit settings. In this paper, we identify channel ordering as an important but underexplored factor in per-group quantization. In this setting, each contiguous group shares one quantization scale. When channels with very different statistics are

*Equal contribution.
†Corresponding author: Yulun Zhang, yulun100@gmail.com

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

placed in the same group, the scale can be dominated by outliers and cause large quantization errors. Based on this observation, we propose **PermuQuant**, a simple and effective PTQ framework for low-bit diffusion models. PermuQuant sorts channels by a joint second-moment criterion before per-group quantization, placing channels with similar activation and weight statistics into the same group. It further uses a calibration-based acceptance rule to apply reordering only when the selected permutation reduces quantization error on calibration data. The selected permutations are absorbed into adjacent modules or applied to weights offline, avoiding explicit runtime permutation operations. Extensive experiments on multiple large diffusion models show that PermuQuant consistently reduces quantization error and outperforms existing PTQ baselines. On FLUX.1-dev with an NVIDIA RTX 5090, PermuQuant achieves up to a **1.7**× single step speedup and reduces the DiT memory footprint by **3.5**× under W4A4 NVFP4 quantization. Code is available at `https://github.com/yscheng04/PermuQuant`.

1 Introduction

Diffusion models have become a dominant paradigm for visual generation. By learning to reverse a gradual denoising process, they can generate high-fidelity images with strong diversity [14, 51]. Recent advances further improve their sampling process, noise schedules, and continuous-time formulations [39, 52, 53]. Together with classifier-free guidance and latent diffusion, these methods have enabled powerful text-to-image generation systems [13, 42, 45]. Meanwhile, the backbone of diffusion models has shifted from convolutional UNets to scalable transformer architectures [4, 14, 16, 59]. This trend greatly improves generation quality. However, it also increases the memory footprint and computational cost of inference. Since diffusion sampling requires repeated network evaluations, the cost becomes especially high for large models and high-resolution generation.

Model quantization [38] has emerged as a widely adopted technique to compress floating-point parameters and activations into lower-precision formats, thereby reducing memory overhead and accelerating inference. The resulting speedup stems from both diminished cache loading costs and faster low-bit arithmetic. Existing quantization approaches are broadly categorized into two paradigms: post-training quantization (PTQ) and quantization-aware training (QAT). While QAT explicitly models quantization effects during training and achieves strong performance at extremely low bit-widths [5, 43], it typically requires full or partial model retraining, incurring substantial computational and data costs. In contrast, PTQ operates directly on pretrained checkpoints without additional training, making it considerably more resource-efficient. Recent advances in PTQ [1, 19, 24, 56–58] have demonstrated that competitive accuracy can be preserved even at low bit-widths. These methods offer a highly practical pathway for deploying large-scale visual models in resource-constrained environments. For example, interactive applications require low-latency inference, while personal deployment on a single GPU is often limited by memory capacity.

Despite this progress, extremely low-bit weight-activation quantization remains challenging (see Fig. 1). In particular, 3-bit quantization can cause severe degradation in image quality and text-image alignment. A common remedy is per-group quantization, which partitions each token or channel into small contiguous groups of equal size and assigns each group its own scale. This strategy gives finer-grained scaling than per-token or per-channel quantization.

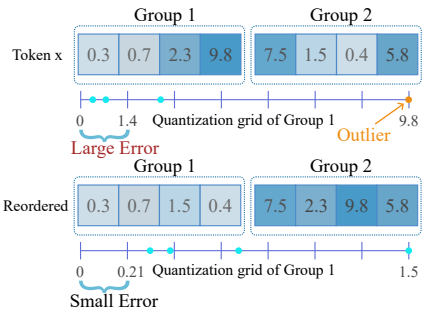


Figure 2: Example of per-group quantization. Quantization error is greatly affected by channel orders.

The **key insight** of this work is simple: *per-group quantization makes channel order important*. As illustrated in Fig. 2, since groups are formed over contiguous channels, the order decides which channels share a scale. This matters because each group scale is controlled by the largest-magnitude values in the group. If statistically mismatched channels are grouped together, outlier channels dominate the shared scale. The remaining channels are then quantized on an overly coarse grid, which increases quantization error. By contrast, placing channels with similar statistics into the same group produces more balanced and better matched scales, thereby reducing quantization error.

In this paper, we propose **PermuQuant**, a simple and effective PTQ framework for low-bit diffusion models. PermuQuant exploits the group structure of per-group quantization by reordering channels before quantization. It uses channel second moments estimated from calibration data to construct more favorable contiguous groups. To formalize this strategy, we show that second-moment based sorting minimizes an upper bound of the expected quantization error. We further derive an approximation guarantee for the expected error objective under the uniform noise approximation [34]. These results provide a theoretical justification for using channel second moments to guide reordering.

To make reordering practical for linear layers, **firstly**, PermuQuant jointly considers activation and weight statistics. This is necessary because preserving the function of a linear layer requires applying the same permutation to both sides. We therefore use a joint second-moment criterion to select the channel order. **Secondly**, not all layers benefit equally from reordering. To address this issue, we introduce a calibration-based acceptance rule. It applies reordering only when the selected permutation sufficiently reduces the quantization error over calibration data. **Thirdly**, another practical challenge is that explicit channel permutation may introduce extra runtime operations. To avoid this cost, we apply the permutation to weight matrix offline and absorb the activation-side permutation into adjacent modules. This design enables PermuQuant to improve low-bit quantization without explicit runtime permutation operations. Extensive experiments on multiple large diffusion models show that PermuQuant consistently reduces quantization error and delivers strong performance under extremely low-bit settings. We further validate its practical efficiency on Flux.1-dev with an NVIDIA RTX 5090, where W4A4 NVFP4 quantization achieves a **1.7**$\times$ single step speedup and reduces the DiT memory footprint by **3.5**$\times$. Our main contributions are summarized as follows:

- We are first to identify channel order as an important factor in per-group quantization. We show that channel reordering can be used to reduce low-bit quantization error.
- We propose PermuQuant, a simple PTQ framework that makes channel reordering practical for quantized linear layers. It jointly considers activation and weight statistics, and uses a calibration-based rule to apply reordering only when it provides reliable gains.
- Extensive experiments on multiple large diffusion models demonstrate the effectiveness of PermuQuant. Our method consistently outperforms existing PTQ baselines under low-bit settings. On Flux-1.dev with an NVIDIA RTX 5090, PermuQuant achieves a **1.7**$\times$ single step speedup and reduces the DiT memory by **3.5**$\times$ under W4A4 NVFP4 quantization.

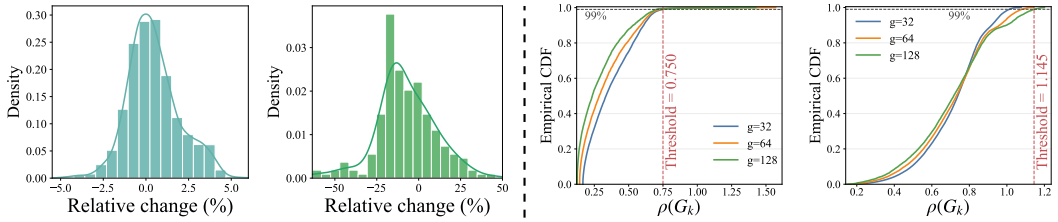
2 Related Work

2.1 Diffusion Model

Diffusion models have become a dominant family of generative models for high-quality visual generation. They generate samples by reversing a gradual noising process, where a neural network iteratively denoises corrupted inputs [14, 51]. Recent advances in sampling, noise schedules, and conditional generation have enabled powerful text-to-image systems [2, 8, 13, 39, 42, 45, 52, 53]. Meanwhile, the backbone architecture has gradually shifted from convolutional UNets [14, 46] to scalable transformer-based designs [4, 16, 59]. Despite their strong generation quality, diffusion models remain expensive at inference time due to sequential denoising steps and increasingly large backbones. Existing acceleration methods mainly follow two directions. One direction reduces the number of denoising steps through fast samplers or distillation [26, 30, 31, 36, 47, 48, 54, 61, 62, 64]. The other reduces the cost of each step through efficient architectures, sparse inference, distributed inference, and quantization [7, 9, 18, 20, 21, 23, 32, 33, 49, 55]. This work focuses on low-bit quantization, which reduces memory and computation without changing the sampling schedule.

2.2 Model Quantization

Model quantization [38] is a widely used technique for compressing model size and accelerating model inference. By representing weights and activations with low-bit numerical formats, quantization can substantially reduce memory footprint and computational cost. Existing methods mainly include quantization-aware training (QAT) and post-training quantization (PTQ). QAT [5, 6, 27, 35, 43, 44] simulates low-precision operations during training and performs well in extremely low-bit settings, but requires costly retraining. PTQ [25, 29], in contrast, directly quantizes pretrained models without end-to-end retraining and is more resource-efficient. Recent PTQ methods have enabled 4-bit quantization for LLMs and diffusion models with limited quality degradation [19, 28]. GPTQ [10] uses Hessian-based compensation, while SmoothQuant [58], OmniQuant [50], QuaRot [1], and ConvRot [15]



(a) Random permutation (b) Sorting (c) Activation ratio (d) Weight ratio

Figure 3: (a) and (b): Relative change in activation quantization error caused by random channel permutations and sorting channels by their second moments on Z-Image-Turbo. (c) and (d): Empirical validation of Assumption 3.1. We plot the CDF of the ratio $\rho(G_k)$, which estimates the constant C required by each group. For more than 99% of groups in activation and weight, $\rho(G_k) < 1.2$.

suppress activation outliers through smoothing or orthogonal transformations. SVDQuant [19] absorbs outliers into a low-rank branch. RPTQ [63] and IGQ-ViT [37] group channels with similar statistics to share quantization parameters. In this work, we optimize channel ordering for fixed-size per-group quantization. We demonstrate that channel reordering can further reduce quantization error even under 3-bit setting while preserving compatibility with standard per-group quantization kernels.

3 Method

In this section, we first review the basic concept of quantization in Sec. 3.1, with a focus on per-group quantization. We then present the channel reordering technique in Sec. 3.2, which aims to reduce per-group quantization error. We first motivate why channel order matters, then describe how channels are reordered, with theoretical analysis to explain its effectiveness. Finally, we propose PermuQuant in Sec. 3.3. We extend channel reordering to practical quantized linear layers with joint second-moment channel reordering. We then adopt calibration-based acceptance rule to select effective permutations. We also discuss how the selected permutations can be deployed without explicit runtime operations.

3.1 Background

Quantization reduces the memory cost and inference latency of large models by representing weights or activations with low-bit integers. In low-bit settings, a scale factor maps floating-point values to a discrete integer grid, so the quantization error strongly depends on the scales. A common practice is to use fine-grained scaling, such as per-token activation quantization or per-channel weight quantization, since different tokens or channels can have very different value ranges.

In extremely low-bit settings, an even more fine-grained scheme, namely *per-group quantization*, is often adopted. It further partitions the values within each token or channel into contiguous groups of equal size. All scalars within the same group share one quantization scale. Let $\mathbf{x} \in \mathbb{R}^d$ denote a token-wise or channel-wise vector. We divide its entries into $K = d/g$ contiguous groups $G_1, \ldots, G_K$, where each group has size g. For a group G_k, the symmetric quantization scale is computed as

$$s_k = \frac{\max_{i \in G_k} |x_i|}{Q}, \tag{1}$$

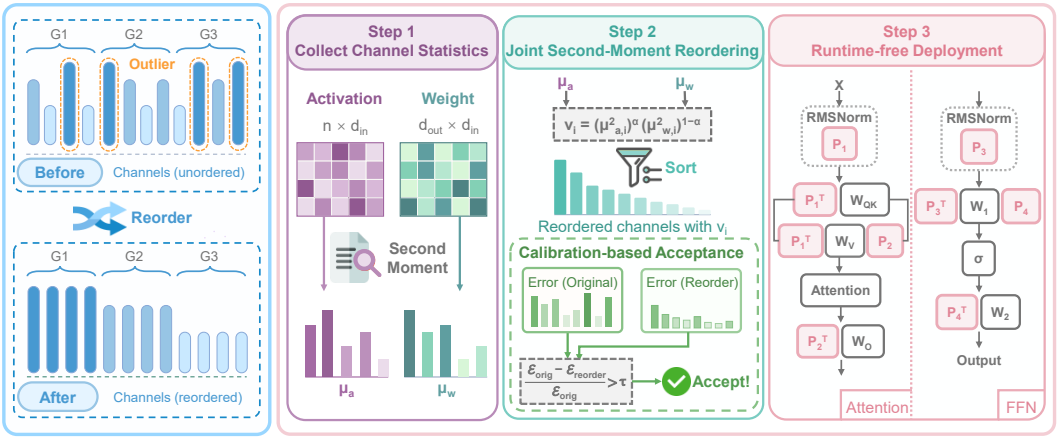
where Q denotes the largest representable integer. Each element x_i is quantized and dequantized as

$$z_i = \text{clip}\left(\text{round}\left(\frac{x_i}{s_k}\right), -Q, Q\right), \qquad \hat{x}_i = s_k z_i. \tag{2}$$

3.2 Lower Quantization Error with Reordering

For clarity, we present the following analysis using activation quantization as an example. The same idea also applies to weight quantization by treating weight channels as the grouped vectors.

Motivation. In per-group quantization, channels are partitioned into contiguous groups, and all channels within the same group share one quantization scale. As a result, the quantization error depends not only on the underlying value distribution, but also on the channel order. Therefore, even when the set of channels remains unchanged, reordering them can lead to substantially different quantization errors. We also empirically observe that different random permutations can produce noticeably different quantization errors in Fig. 3a. It highlights that the permutation is an important factor in per-group quantization. Therefore, the core issue is how to find a good permutation. More



(a) Reordering reduces outlier channels **(b) PermuQuant**

Figure 4: Overview of PermuQuant. (a) Channel reordering places channels with similar statistics into the same group. (b) PermuQuant first collects activation and weight statistics from calibration data. It then performs joint second-moment reordering with calibration-based acceptance. The accepted permutations are folded into adjacent modules offline for runtime-free deployment.

formally, let π denote a permutation over the d channels, and let $\{G_k(\pi)\}$ be the contiguous groups induced by π. Our goal is to find a permutation that minimizes the expected quantization error:

$$\pi^\star = \arg\min_{\pi} \mathbb{E}_{\mathbf{x}} \left[\|\mathbf{x} - \hat{\mathbf{x}}_\pi\|_2^2 \right], \tag{3}$$

where $\hat{\mathbf{x}}_\pi$ denotes the dequantized tensor obtained under the grouping $\{G_k(\pi)\}$.

Intuition. The objective in Eq. (3) is difficult to optimize directly, since the permutation space is combinatorial. We therefore seek a simple criterion that can identify low-error group partitions. It's well known that in low-bit quantization, outliers are a major source of error. Under per-group quantization, however, the effect of an outlier is determined by its group. A high-magnitude channel may dominate the scale when grouped with low-magnitude channels, but becomes less harmful when grouped with channels of similar magnitudes. Therefore, a desirable permutation should avoid mixing channels with very different value ranges in the same group. Based on this observation, the intuition of our method is simple: **channels with similar magnitude should be placed into the same group.**

Second-moment based reordering. To formalize this intuition, we derive an upper bound on the expected quantization error in Eq. (3). We then show how this bound can be reduced through an appropriate channel reordering. Consider a random token vector $\mathbf{x} \in \mathbb{R}^d$ drawn from the activation distribution, and let $\hat{\mathbf{x}}$ be its dequantized version. Denote by $\mathbf{e} = \mathbf{x} - \hat{\mathbf{x}}$ the quantization error. For a fixed grouping $\{G_k\}$ with group size g, the scale of each group is determined by its largest-magnitude entry. Therefore, the expected quantization error is bounded by

$$\mathbb{E}_{\mathbf{x}} \|\mathbf{e}\|_2^2 \le \frac{g}{4Q^2} \sum_k \mathbb{E}_{\mathbf{x}} \left[\max_{i \in G_k} |x_i|^2 \right], \tag{4}$$

where Q is the maximum quantization level. See Appendix B.1 for the proof.

Equation (4) shows that the expected quantization error depends on the expected squared maximum within each group. However, directly optimizing this term is difficult, since it depends on the joint distribution of channels assigned to the same group and changes with every candidate permutation. To relate this quantity to channel-wise statistics, we propose the following assumption.

Assumption 3.1 (Extremal control) *For each channel i, let $\mathbb{E}[x_i^2] = \mu_i^2$. We assume that there exists a constant $C > 0$ such that, for any group G_k of size g,*

$$\mathbb{E}\left[\max_{i \in G_k} |x_i|^2 \right] \le C \log_2(2g) \max_{i \in G_k} \mu_i^2. \tag{5}$$

Intuitively, Assumption 3.1 provides a bound on the expected squared maximum within a group. To empirically validate Assumption 3.1, we compute the group-wise ratio $\rho(G_k) =$

$\mathbb{E}[\max_{i\in G_k}|x_i|^2]/(\log_2(2g)\max_{i\in G_k}\mu_i^2)$, which estimates the smallest constant C required for each group. We then plot the empirical CDF of $\rho(G_k)$ over all groups and group sizes. As shown in Fig. 3, more than 99% of groups have $\rho(G_k) < 1.2$, indicating that the extremal-control condition holds with a small empirical constant for most groups. Substituting Eq. (5) into Eq. (4), we obtain

$$\mathbb{E}\|\mathbf{e}\|_2^2 \le \frac{Cg\log(2g)}{4Q^2}\sum_k \max_{i\in G_k}\mu_i^2. \tag{6}$$

Equation (6) shows that the expected quantization error is controlled by the group-wise maxima of channel second moments. Therefore, the upper bound can be reduced by placing channels with similar second moments into the same group. Actually, we have the following proposition:

Proposition 3.2 (Optimality of second-moment sorting) *For a fixed group size g, sorting channels in descending order of μ_i^2 and then partitioning them into contiguous groups minimizes $\sum_k \max_{i\in G_k}\mu_i^2$ among all contiguous group partitions induced by channel permutations.*

We defer the proof to Appendix B.2. This motivates our second-moment based channel reordering strategy, where **channels are sorted by their second moments**.

Beyond minimizing this upper bound, the second-moment based ordering also provides an approximation guarantee for the expected quantization error objective under the uniform-noise approximation [34]. The following theorem states the resulting approximation ratio.

Theorem 3.3 (Approximation guarantee) *Let $\mathcal{E}(P) := \mathbb{E}_x\mathbb{E}_{\epsilon|x}\|\epsilon\|_2^2$ denote the expected quantization error under the uniform-noise approximation [34]. ϵ is the additive quantization noise and $\epsilon_i \mid x \sim \mathrm{Unif}(-s_k/2, s_k/2)$ for $i \in G_k$. Let P_{sort} be the partition obtained by sorting channels in descending order of μ_i^2, where $\mathbb{E}[x_i^2] = \mu_i^2$. Assume that Assumption 3.1 holds. Then*

$$\mathcal{E}(P_{\mathrm{sort}}) \le C\log_2(2g)\cdot\min_P \mathcal{E}(P). \tag{7}$$

Therefore, second-moment based sorting is a $C\log_2(2g)$-approximation to the optimal partition with respect to expected quantization error. We defer the full proof to Appendix B.3.

3.3 PermuQuant

Joint second-moment channel reordering. The above analysis considers only one side of the linear layer. In practice, reordering must preserve the function of the linear layer. This requires applying the same permutation to both the activations and the corresponding input channels of the weight matrix. Given an activation matrix $\mathbf{X}$, a weight matrix $\mathbf{W}$, and a permutation matrix $\mathbf{P}$, we have

$$\mathbf{Y} = \mathbf{XW} = \mathbf{XPP}^\top\mathbf{W} \approx Q(\mathbf{XP})Q(\mathbf{P}^\top\mathbf{W}), \tag{8}$$

where $Q(\cdot)$ denotes the quantize-then-dequantize operator. Therefore, the final channel order should take into account the statistics of both activation and weight sides.

For each input channel i, let $\mu_{a,i}^2$ denote the activation second moment and $\mu_{w,i}^2$ denote the second moment of the corresponding weight channel. We define a joint second moment criterion as

$$v_i = (\mu_{a,i}^2)^\alpha(\mu_{w,i}^2)^{1-\alpha}, \tag{9}$$

where $\alpha \in [0,1]$ balances the relative importance of activation and weight. We then sort channels according to v_i. Different layers may be dominated by activation or weight to different extents. To account for this, we uniformly enumerate a set of candidate values for α in the interval $[0,1]$. We then select the value that yields the smallest quantization error on the calibration data.

Calibration-based acceptance rule. However, reordering is not uniformly beneficial for all layers. First, activations and weights may prefer different channel orders. Thus, a permutation that reduces error on one side can increase the error on the other. Second, some layers may not have a clear sortable structure in their channel statistics, or the original channel order already provides an effective grouping. In such cases, the overall benefit of reordering can be marginal or even negative. To prevent such cases, we propose a calibration-based acceptance rule. Let $\mathcal{E}_{\mathrm{orig}}$ and $\mathcal{E}_{\mathrm{reorder}}$ denote the quantization errors on the calibration set before and after reordering. We accept the permutation if

$$\frac{\mathcal{E}_{\mathrm{orig}} - \mathcal{E}_{\mathrm{reorder}}}{\mathcal{E}_{\mathrm{orig}}} > \tau, \tag{10}$$

Table 1: Quantitative comparison on MJHQ30K and GenEval. For MJHQ30K, we report FID, KID$\times 10^3$, and ImageReward (IR); for GenEval, we report the standard scores. The best and second-best results among **W3A3** methods are highlighted in **bold** and underline, respectively.

Model	Method	Venue	MJHQ30K			GenEval						
			FID ↓	KID ↓	IR ↑	Single ↑	Two ↑	Count ↑	Color ↑	Pos. ↑	Attr. ↑	Overall ↑
Z-Image -Turbo (8 Steps)	BF16	–	67.6	5.7	1.001	1.00	0.97	0.71	1.00	0.63	0.83	0.86
	RTN	–	250.8	183.0	-1.609	0.69	0.21	0.21	0.47	0.21	0.10	0.32
	SmoothQuant [58]	ICML'23	221.7	136.1	-1.388	0.85	0.55	0.43	0.77	0.18	0.20	0.50
	OmniQuant [50]	ICLR'24	87.7	18.0	0.575	0.92	0.87	**0.71**	0.90	0.58	0.65	0.77
	SVDQuant [19]	ICLR'25	186.7	92.9	-0.940	0.85	0.61	0.32	0.83	0.50	0.43	0.59
	ConvRot [15]	arXiv'25	86.5	14.1	0.587	**1.00**	0.87	**0.71**	0.97	0.61	0.70	0.81
	PermuQuant	–	134.9	49.2	-0.163	0.96	0.87	0.61	0.93	0.55	0.50	0.74
	PermuQuant-H	–	**80.4**	**12.3**	**0.715**	0.96	**0.97**	0.64	**1.00**	**0.63**	**0.75**	**0.83**
FLUX.1 -dev (50 Steps)	BF16	–	60.6	2.8	0.917	1.0	0.89	0.61	0.93	0.68	0.53	0.77
	RTN	–	131.0	52.4	0.386	0.81	0.45	0.29	0.67	0.32	0.10	0.44
	SmoothQuant [58]	ICML'23	142.6	59.7	0.149	0.81	0.32	0.29	0.63	0.34	0.13	0.42
	OmniQuant [50]	ICLR'24	85.5	16.0	**0.798**	0.92	0.71	0.46	0.83	0.55	0.38	0.64
	SVDQuant [19]	ICLR'25	103.3	24.7	0.501	0.86	0.29	0.45	0.61	0.47	0.11	0.47
	ConvRot [15]	arXiv'25	68.6	6.2	0.709	0.92	0.89	**0.71**	0.90	0.68	**0.50**	**0.77**
	PermuQuant	–	74.9	9.1	0.726	0.96	0.84	0.50	**0.97**	0.58	0.40	0.71
	PermuQuant-H	–	**64.7**	**4.6**	0.759	**1.00**	**0.92**	0.54	0.93	**0.74**	0.48	**0.77**
SANA-1.5 -1.6B (20 Steps)	BF16	–	58.1	1.2	1.075	1.00	0.97	0.79	0.97	0.66	0.55	0.82
	RTN	–	103.6	30.1	0.394	**0.96**	0.63	0.54	0.83	0.47	0.43	0.64
	SmoothQuant [58]	ICML'23	102.6	29.4	0.466	0.88	0.68	0.46	0.87	0.47	0.45	0.64
	OmniQuant [50]	ICLR'24	96.8	26.0	0.541	**0.96**	0.76	0.57	0.83	0.47	0.53	0.69
	SVDQuant [19]	ICLR'25	89.7	19.5	0.513	**0.96**	0.76	0.46	0.93	0.45	0.55	0.69
	ConvRot [15]	arXiv'25	98.1	24.8	0.440	**0.96**	0.74	0.46	0.83	0.55	0.55	0.68
	PermuQuant	–	82.8	14.7	0.700	0.92	0.76	**0.61**	0.93	0.45	0.60	0.71
	PermuQuant-H	–	**68.0**	**5.5**	**0.919**	**0.96**	**0.89**	0.57	**0.97**	**0.66**	**0.73**	**0.80**

where τ is a predefined threshold. Otherwise, we keep the original channel order. In this way, PermuQuant is applied only to layers where reordering yields a sufficiently large empirical improvement. This avoids possible increase in quantization error when reordering is not effective.

Runtime-free deployment. Importantly, the permutation introduces negligible additional runtime cost. The permutation on weight matrix can be performed offline. The permutation on activation can be absorbed either into the preceding linear modules or into its preceding LayerNorm/RMSNorm.

4 Experiments

4.1 Experimental Settings

Data and evaluation. Following prior work [20], we sample calibration prompts from COCO Captions [22]. For generation quality, we sample 1K prompts from MJHQ-30K [17] and report FID [12, 40], KID [3], and ImageReward [60]. For instruction alignment, we use GenEval [11], covering single object, two objects, counting, color, position, and attribute binding. Please refer to Appendix E for more implementation details.

4.2 Comparison with State-of-the-art Methods

We compare our method with representative PTQ methods, including RTN [38], SmoothQuant [58], OmniQuant [50], SVDQuant [19] and ConvRot [15]. All methods are evaluated under 3-bit symmetric per-group quantization on three diffusion models: Z-Image-Turbo [4], FLUX.1-dev [16], and SANA-1.5-1.6B [59]. We report two variants of our method, **PermuQuant** and **PermuQuant-H**, where the latter applies reordering after the Hadamard transformation [1].

Quantitative results. Table 1 reports the quantitative comparison on MJHQ30K and GenEval. Across the three evaluated diffusion models, direct 3-bit RTN causes severe degradation, especially on FID and KID. Existing PTQ methods alleviate this issue, but their gains vary. Our method consistently improves low-bit performance. On Z-Image-Turbo, **PermuQuant-H** reduces FID from 86.5 to 80.4 and KID from 14.1 to 12.3 compared with ConvRot, while improving the GenEval overall score from 0.81 to 0.83. For FLUX.1-dev, **PermuQuant-H** achieves the best FID and KID among

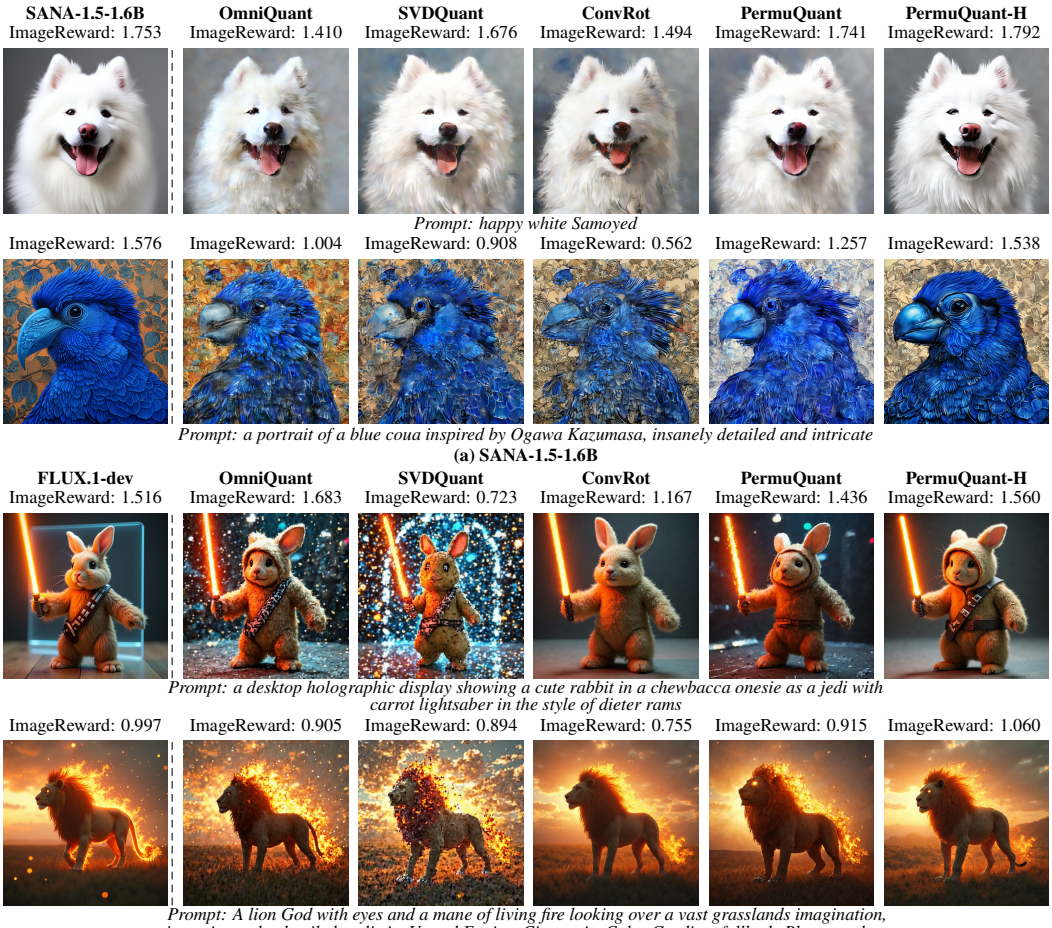


Figure 5: Qualitative comparison on SANA-1.5-1.6B and FLUX.1-dev. All methods are evaluated under the W3A3 setting. The Image Reward score of each generated image is shown in the figure.

quantized methods and matches the best GenEval overall score. On SANA-1.5-1.6B, **PermuQuant** already improves the GenEval overall score from 0.68 to 0.71 over ConvRot. These results show our strategy's effectiveness across different diffusion backbones.

Qualitative results. We show qualitative comparisons in Fig. 5. Under W3A3 quantization, existing PTQ methods can produce noticeable generation artifacts, such as noisy backgrounds, unstable textures, and degraded color consistency. In contrast, our method generates images that are visually closer to the BF16 outputs. It better preserves object structures, colors, and styles across different prompts and diffusion backbones. More visual results are provided in Appendix H.4.

4.3 Ablation Study

We conduct ablation studies on Z-Image-Turbo to analyze the main design choices of our method. The first three studies are performed with Hadamard transformation, while the group-size study is performed without Hadamard transformation. The results are summarized in Tab. 2.

Reordering criterion. We first compare different criteria for channel reordering. Sorting by the second moment achieves the best FID and ImageReward. Compared with variance-based sorting, it improves ImageReward from 0.629 to 0.715. Random, absmax, and round-robin ordering are less effective. This supports our choice of second moment as the reordering criterion.

Source of statistics. We next study whether the reordering statistics should be computed from weights, activations, or both according to our joint sorting strategy. Activation statistics outperform weight statistics, and joint second-moment reordering further improves FID and ImageReward. This shows the benefit of accounting for both activation and weight quantization difficulty.

Table 2: Ablation studies. B and R denote the baseline and reordered variants, respectively.

Metric	Random	Absmax	Round-Robin	Variance	Second Moment
FID ↓	89.0	85.3	83.5	83.3	80.4
IR ↑	0.608	0.638	0.630	0.629	0.715
GenEval ↑	0.83	0.84	0.77	0.82	0.83

(a) Reordering criterion.

Metric	Weight	Activation	Joint
FID ↓	87.7	83.2	80.4
IR ↑	0.621	0.647	0.715
GenEval ↑	0.82	0.84	0.83

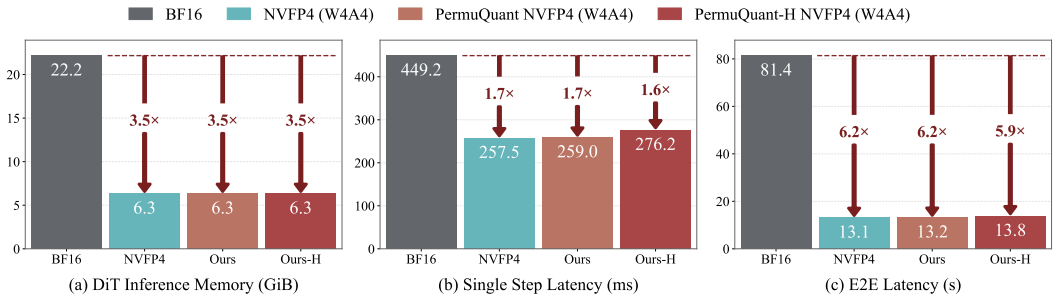
(b) Source of statistics.

Metric	No τ	τ=0	τ=2.5	τ=5
FID ↓	85.4	80.4	80.3	82.8
IR ↑	0.624	0.715	0.679	0.635
GenEval ↑	0.83	0.83	0.83	0.81

(c) Threshold choices.

Metric	g=32 B	g=32 R	g=128 B	g=128 R	g=256 B	g=256 R
FID ↓	251	135	77.8	74.3	93.9	85.8
IR ↑	-1.61	-0.163	0.814	0.846	0.533	0.563
GenEval ↑	0.32	0.74	0.81	0.83	0.82	0.84

(d) Group size.



(a) DiT Inference Memory (GiB) (b) Single Step Latency (ms) (c) E2E Latency (s)

Figure 6: Efficiency comparison on FLUX.1-dev using an RTX 5090 Desktop 32GB GPU. Reordering introduces almost no extra overhead compared with NVFP4. By eliminating CPU offloading, our NVFP4 variants achieve up to a 6.3× E2E latency speedup over the BF16 baseline.

Threshold choices. We also evaluate different thresholds for accepting a reordered permutation. Without the acceptance threshold, reordering can hurt performance, leading to worse FID and ImageReward. Using $\tau = 0$ gives the best ImageReward and achieves strong FID and GenEval scores. A moderate threshold, $\tau = 2.5$, gives a similar FID and the same GenEval score. However, a larger threshold degrades the results. We therefore use $\tau = 0$ as the default setting.

Group size. Finally, we evaluate the effect of reordering under different group sizes. For $g = 32$, we use the 3-bit setting. For $g = 128$ and $g = 256$, we use the 4-bit setting. Reordering improves the baseline under all group sizes. The gain is especially large for $g = 32$, where GenEval improves from 0.32 to 0.74. These results show that our method is effective across different group size.

4.4 Efficiency Comparison

In Fig. 6, we report the measured DiT inference memory, single-step latency, and end-to-end (E2E) latency for FLUX.1-dev on an RTX 5090 Desktop GPU with 32GB memory. Compared with the BF16 baseline, Our method retains the same memory footprint and introduces almost no extra overhead over NVFP4. It also achieves nearly identical single-step latency. For E2E latency, our NVFP4 variants reduce the latency from 81.4 s to less than 14 s. Here, E2E latency includes text encoding, DiT denoising, and VAE decoding. Notably, this large E2E gain is partly because quantization eliminates the CPU offloading required by BF16 inference.

5 Conclusion

In this paper, we propose PermuQuant, a simple and effective low-bit quantization framework for diffusion models based on channel reordering. Our key observation is that, under per-group quantization, the channel order determines the induced group partitions and therefore has a substantial impact on quantization error. Based on this observation, PermuQuant groups channels with similar activation and weight statistics together to reduce within-group range mismatch. We then use a calibration-based rule to apply reordering only when it reliably lowers quantization error. Extensive experiments across multiple large diffusion models show that PermuQuant consistently preserves generation quality even under extremely low-bit quantization. On FLUX.1-dev with an NVIDIA RTX 5090, PermuQuant achieves up to a **1.7**× single step speedup and reduces the DiT memory footprint by **3.5**× under W4A4 NVFP4 quantization. These results highlight channel reordering as a simple and practical direction for low-bit diffusion model compression.

Acknowledgments and Disclosure of Funding

This work was supported by Huawei Explore X funds. This work was also sponsored by AI Hundred Schools Program and was carried out using the Ascend AI technology stack.

References

[1] Saleh Ashkboos, Amirkeivan Mohtashami, Maximilian L Croci, Bo Li, Pashmina Cameron, Martin Jaggi, Dan Alistarh, Torsten Hoefler, and James Hensman. Quarot: Outlier-free 4-bit inference in rotated llms. In *NeurIPS*, 2024.

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

[3] Mikołaj Bińkowski, Danica J Sutherland, Michael Arbel, and Arthur Gretton. Demystifying mmd gans. *arXiv*, 2018.

[4] Huanqia Cai, Sihan Cao, Ruoyi Du, Peng Gao, Steven Hoi, Zhaohui Hou, Shijie Huang, Dengyang Jiang, Xin Jin, Liangchen Li, et al. Z-image: An efficient image generation foundation model with single-stream diffusion transformer. *arXiv*, 2025.

[5] Mengzhao Chen, Wenqi Shao, Peng Xu, Jiahao Wang, Peng Gao, Kaipeng Zhang, and Ping Luo. Efficientqat: Efficient quantization-aware training for large language models. In *ACL*, 2025.

[6] Zheng Chen, Zhi Yang, Xiaoyang Liu, Weihang Zhang, Mengfan Wang, Yifan Fu, Linghe Kong, and Yulun Zhang. BinaryDemoire: Moiré-aware binarization for image demoiréing. *arXiv*, 2026.

[7] Zigeng Chen, Xinyin Ma, Gongfan Fang, Zhenxiong Tan, and Xinchao Wang. Asyncdiff: Parallelizing diffusion models by asynchronous denoising. In *NeurIPS*, 2024.

[8] Prafulla Dhariwal and Alexander Nichol. Diffusion models beat gans on image synthesis. In *NeurIPS*, 2021.

[9] Jiarui Fang, Jinzhe Pan, Aoyu Li, Xibo Sun, and Jiannan Wang. Pipefusion: Patch-level pipeline parallelism for diffusion transformers inference. *arXiv*, 2024.

[10] Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. Gptq: Accurate post-training quantization for generative pre-trained transformers. *arXiv*, 2022.

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

[12] Martin Heusel, Hubert Ramsauer, Thomas Unterthiner, Bernhard Nessler, and Sepp Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium. In *NeurIPS*, 2017.

[13] Jonathan Ho and Tim Salimans. Classifier-free diffusion guidance. *arXiv*, 2022.

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

[15] Feice Huang, Zuliang Han, Xing Zhou, Yihuang Chen, Lifei Zhu, and Haoqian Wang. Convrot: Rotation-based plug-and-play 4-bit quantization for diffusion transformers. *arXiv*, 2025.

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

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

[18] Muyang Li, Tianle Cai, Jiaxin Cao, Qinsheng Zhang, Han Cai, Junjie Bai, Yangqing Jia, Kai Li, and Song Han. Distrifusion: Distributed parallel inference for high-resolution diffusion models. In *CVPR*, 2024.

[19] Muyang Li, Yujun Lin, Zhekai Zhang, Tianle Cai, Xiuyu Li, Junxian Guo, Enze Xie, Chenlin Meng, Jun-Yan Zhu, and Song Han. Svdquant: Absorbing outliers by low-rank components for 4-bit diffusion models. In *ICLR*, 2025.

[20] Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-diffusion: Quantizing diffusion models. In *ICCV*, 2023.

[21] Yanyu Li, Huan Wang, Qing Jin, Ju Hu, Pavlo Chemerys, Yun Fu, Yanzhi Wang, Sergey Tulyakov, and Jian Ren. Snapfusion: Text-to-image diffusion model on mobile devices within two seconds. In *NeurIPS*, 2023.

[22] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *ECCV*, 2014.

[23] Jiacheng Liu, Chang Zou, Yuanhuiyi Lyu, Junjie Chen, and Linfeng Zhang. From reusing to forecasting: Accelerating diffusion models with taylorseers. In *ICCV*, 2025.

[24] Kai Liu, Shaoqiu Zhang, Linghe Kong, and Yulun Zhang. Clq: Cross-layer guided orthogonal-based quantization for diffusion transformers. *arXiv*, 2025.

[25] Kai Liu, Qian Zheng, Kaiwen Tao, Zhiteng Li, Haotong Qin, Wenbo Li, Yong Guo, Xianglong Liu, Linghe Kong, Guihai Chen, et al. Low-bit model quantization for deep neural networks: A survey. *arXiv*, 2025.

[26] Luping Liu, Yi Ren, Zhijie Lin, and Zhou Zhao. Pseudo numerical methods for diffusion models on manifolds. *arXiv*, 2022.

[27] Zechun Liu, Zhiqiang Shen, Marios Savvides, and Kwang-Ting Cheng. Reactnet: Towards precise binary neural network with generalized activation functions. In *ECCV*, 2020.

[28] Zechun Liu, Changsheng Zhao, Igor Fedorov, Bilge Soran, Dhruv Choudhary, Raghuraman Krishnamoorthi, Vikas Chandra, Yuandong Tian, and Tijmen Blankevoort. Spinquant: Llm quantization with learned rotations. *arXiv*, 2024.

[29] Zhenhua Liu, Yunhe Wang, Kai Han, Wei Zhang, Siwei Ma, and Wen Gao. Post-training quantization for vision transformer. In *NeurIPS*, 2021.

[30] 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*, 2022.

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

[32] Zhaoyang Lyu, Xudong Xu, Ceyuan Yang, Dahua Lin, and Bo Dai. Accelerating diffusion models via early stop of the diffusion process. *arXiv*, 2022.

[33] Xinyin Ma, Gongfan Fang, and Xinchao Wang. Deepcache: Accelerating diffusion models for free. In *CVPR*, 2024.

[34] Daniel Marco and David L. Neuhoff. The validity of the additive noise model for uniform scalar quantizers. *TIT*, 2005.

[35] Brais Martinez, Jing Yang, Adrian Bulat, and Georgios Tzimiropoulos. Training binary neural networks with real-to-binary convolutions. In *ICLR*, 2020.

[36] Chenlin Meng, Robin Rombach, Ruiqi Gao, Diederik Kingma, Stefano Ermon, Jonathan Ho, and Tim Salimans. On distillation of guided diffusion models. In *CVPR*, 2023.

[37] Jaehyeon Moon, Dohyung Kim, Junyong Cheon, and Bumsub Ham. Instance-aware group quantization for vision transformers. In *CVPR*, 2024.

[38] Markus Nagel, Marios Fournarakis, Rana Ali Amjad, Yelysei Bondarenko, Mart Van Baalen, and Tijmen Blankevoort. A white paper on neural network quantization. *arXiv*, 2021.

[39] Alexander Quinn Nichol and Prafulla Dhariwal. Improved denoising diffusion probabilistic models. In *ICML*, 2021.

[40] Gaurav Parmar, Richard Zhang, and Jun-Yan Zhu. On aliased resizing and surprising subtleties in gan evaluation. In *CVPR*, 2022.

[41] William Peebles and Saining Xie. Scalable diffusion models with transformers. In *ICCV*, 2023.

[42] 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*, 2023.

[43] Haotong Qin, Ruihao Gong, Xianglong Liu, Mingzhu Shen, Ziran Wei, Fengwei Yu, and Jingkuan Song. Forward and backward information retention for accurate binary neural networks. In *CVPR*, 2020.

[44] Haotong Qin, Yulun Zhang, Yifu Ding, Yifan liu, Xianglong Liu, Martin Danelljan, and Fisher Yu. Quantsr: Accurate low-bit quantization for efficient image super-resolution. In *NeurIPS*, 2023.

[45] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *CVPR*, 2022.

[46] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015.

[47] Tim Salimans and Jonathan Ho. Progressive distillation for fast sampling of diffusion models. *arXiv*, 2022.

[48] Axel Sauer, Dominik Lorenz, Andreas Blattmann, and Robin Rombach. Adversarial diffusion distillation. In *ECCV*, 2024.

[49] Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *CVPR*, 2023.

[50] Wenqi Shao, Mengzhao Chen, Zhaoyang Zhang, Peng Xu, Lirui Zhao, Zhiqian Li, Kaipeng Zhang, Peng Gao, Yu Qiao, and Ping Luo. Omniquant: Omnidirectionally calibrated quantization for large language models. *arXiv*, 2023.

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

[52] Jiaming Song, Chenlin Meng, and Stefano Ermon. Denoising diffusion implicit models. *arXiv*, 2020.

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

[54] Yang Song, Prafulla Dhariwal, Mark Chen, and Ilya Sutskever. Consistency models. In *ICML*, 2023.

[55] Hasan Unlu. Efficient neural network deployment for microcontroller. *arXiv*, 2020.

[56] Junyi Wu, Zhiteng Li, Zheng Hui, Yulun Zhang, Linghe Kong, and Xiaokang Yang. QuantCache: Adaptive importance-guided quantization with hierarchical latent and layer caching for video generation. In *ICCV*, 2025.

[57] Tianxing Wu, Zheng Chen, Cirou Xu, Bowen Chai, Yong Guo, Yutong Liu, Linghe Kong, and Yulun Zhang. LSGQuant: Layer-sensitivity guided quantization for one-step diffusion real-world video super-resolution. In *ICML*, 2026.

[58] Guangxuan Xiao, Ji Lin, Mickael Seznec, Hao Wu, Julien Demouth, and Song Han. Smoothquant: Accurate and efficient post-training quantization for large language models. In *ICML*, 2023.

[59] Enze Xie, Junsong Chen, Yuyang Zhao, Jincheng Yu, Ligeng Zhu, Chengyue Wu, Yujun Lin, Zhekai Zhang, Muyang Li, Junyu Chen, et al. Sana 1.5: Efficient scaling of training-time and inference-time compute in linear diffusion transformer. *arXiv*, 2025.

[60] Jiazheng Xu, Xiao Liu, Yuchen Wu, Yuxuan Tong, Qinkai Li, Ming Ding, Jie Tang, and Yuxiao Dong. Imagereward: Learning and evaluating human preferences for text-to-image generation. In *NeurIPS*, 2023.

[61] Tianwei Yin, Michaël Gharbi, Taesung Park, Richard Zhang, Eli Shechtman, Fredo Durand, and William T Freeman. Improved distribution matching distillation for fast image synthesis. In *NeurIPS*, 2024.

[62] Tianwei Yin, Michaël Gharbi, Richard Zhang, Eli Shechtman, Fredo Durand, William T Freeman, and Taesung Park. One-step diffusion with distribution matching distillation. In *CVPR*, 2024.

[63] Zhihang Yuan, Lin Niu, Jiawei Liu, Wenyu Liu, Xinggang Wang, Yuzhang Shang, Guangyu Sun, Qiang Wu, Jiaxiang Wu, and Bingzhe Wu. RPTQ: Reorder-based post-training quantization for large language models. *arXiv*, 2023.

[64] Qinsheng Zhang and Yongxin Chen. Fast sampling of diffusion models with exponential integrator. *arXiv*, 2022.

A Table of Contents

In the supplementary material, we provide complete proofs, implementation details, analysis, and results, including:

- Sec. B: Proofs of the expected quantization error bound, the optimality of second-moment sorting, and the approximation guarantee.
- Sec. C: Additional details for runtime-free deployment.
- Sec. D: Discussion of how to combine reordering with the Hadamard transform.
- Sec. E: Implementation details.
- Sec. F: Limitations of PermuQuant.
- Sec. G: Broader impact.
- Sec. H: Additional results, including the additional experimental results and the additional visual quality comparisons.

B Proofs

B.1 Proof of Eq. (4)

We prove that

$$\mathbb{E}_{\mathbf{x}}\|\mathbf{e}\|_2^2 \leq \frac{g}{4Q^2}\sum_k \mathbb{E}_{\mathbf{x}}\left[\max_{i\in G_k}|x_i|^2\right]. \tag{11}$$

Proof. Let $e_i = x_i - \hat{x}_i$ denote the quantization error of the i-th element. For each group G_k, the quantization scale is

$$s_k = \frac{\max_{i\in G_k}|x_i|}{Q}. \tag{12}$$

Under round-to-nearest quantization, $|e_i| \leq s_k/2$ for $i \in G_k$. Thus,

$$\begin{aligned}\|\mathbf{e}\|_2^2 &= \sum_k \sum_{i\in G_k} e_i^2 \\ &\leq \sum_k \sum_{i\in G_k} \frac{s_k^2}{4} \\ &= \sum_k \frac{g}{4}\left(\frac{\max_{i\in G_k}|x_i|}{Q}\right)^2 \\ &= \frac{g}{4Q^2}\sum_k \max_{i\in G_k}|x_i|^2.\end{aligned} \tag{13}$$

Taking expectation over $\mathbf{x}$ gives

$$\mathbb{E}_{\mathbf{x}}\|\mathbf{e}\|_2^2 \leq \frac{g}{4Q^2}\sum_k \mathbb{E}_{\mathbf{x}}\left[\max_{i\in G_k}|x_i|^2\right], \tag{14}$$

which proves Eq. (4).

B.2 Proof of Prop. 3.2

We prove that sorting channels by μ_i^2 and then partitioning them into contiguous groups minimizes

$$\sum_k \max_{i\in G_k}\mu_i^2. \tag{15}$$

Proof. Let $a_i = \mu_i^2$ and assume, without loss of generality, that

$$a_1 \geq a_2 \geq \cdots \geq a_d. \tag{16}$$

Let the group size be g and $K = d/g$. Under the sorted partition, the group maxima are

$$a_1,\ a_{g+1},\ a_{2g+1},\ \ldots,\ a_{(K-1)g+1}. \tag{17}$$

Thus, its objective value is

$$S_{\text{sort}} = \sum_{k=1}^{K} a_{(k-1)g+1}. \tag{18}$$

For any partition into K groups of size g, consider the first $(k-1)g+1$ largest elements

$$a_1, \ldots, a_{(k-1)g+1}. \tag{19}$$

Since each group contains at most g elements, these elements must occupy at least k different groups. Therefore, the k-th largest group maximum of any partition is at least

$$a_{(k-1)g+1}. \tag{20}$$

Let $m_1 \geq m_2 \geq \cdots \geq m_K$ be the group maxima of any partition sorted in descending order. Then

$$m_k \geq a_{(k-1)g+1}, \quad k = 1, \ldots, K. \tag{21}$$

It follows that

$$\sum_{k=1}^{K} m_k \geq \sum_{k=1}^{K} a_{(k-1)g+1} = S_{\text{sort}}. \tag{22}$$

Thus, the sorted partition minimizes $\sum_k \max_{i \in G_k} \mu_i^2$.

B.3 Proof of Theorem 3.3

Theorem 3.3 (Approximation guarantee). *Let $\mathcal{E}(P) := \mathbb{E}_x \mathbb{E}_{\epsilon|x} \|\epsilon\|_2^2$ denote the expected quantization error under the uniform-noise approximation [34]. ϵ is the additive quantization noise and $\epsilon_i \mid x \sim \text{Unif}(-s_k/2, s_k/2)$ for $i \in G_k$, with $s_k = \max_{j \in G_k} |x_j|/Q$. Let P_{sort} be the partition obtained by sorting channels in descending order of μ_i^2, where $\mathbb{E}[x_i^2] = \mu_i^2$. Assume that Assumption 3.1 holds. Then*

$$\mathcal{E}(P_{\text{sort}}) \leq C \log(2g) \cdot \min_P \mathcal{E}(P). \tag{23}$$

Proof. Let $P = \{G_1, \ldots, G_K\}$ be a partition of the d channels into $K = d/g$ groups, where each group has size g. For each group G_k, define

$$M_k := \max_{i \in G_k} |x_i|. \tag{24}$$

Under online dynamic per-group symmetric quantization, the quantization step size of group G_k is

$$s_k = \frac{M_k}{Q}, \tag{25}$$

where Q is the largest representable integer magnitude determined by the bit-width.

Under the uniform-noise approximation, the quantization error is modeled as additive noise ϵ. Conditioned on the input activation x, for each $i \in G_k$, we have

$$\epsilon_i \mid x \sim \text{Unif}\left(-\frac{s_k}{2}, \frac{s_k}{2}\right). \tag{26}$$

Therefore,

$$\mathbb{E}_{\epsilon|x}\left[\epsilon_i^2\right] = \frac{s_k^2}{12}, \qquad i \in G_k. \tag{27}$$

It follows that the conditional expected noise energy of group G_k is

$$\mathbb{E}_{\epsilon|x}\left[\sum_{i \in G_k} \epsilon_i^2\right] = \sum_{i \in G_k} \mathbb{E}_{\epsilon|x}\left[\epsilon_i^2\right] = g \cdot \frac{s_k^2}{12} = \frac{g}{12Q^2} M_k^2. \tag{28}$$

Taking expectation over the activation distribution and summing over all groups gives

$$\mathcal{E}(P) = \mathbb{E}_x \mathbb{E}_{\epsilon|x} \|\epsilon\|_2^2 = \frac{g}{12Q^2} \sum_{k=1}^{K} \mathbb{E}_x \left[M_k^2 \right]. \tag{29}$$

Next, by Assumption 3.1, for every group G_k,

$$\mathbb{E}_x \left[M_k^2 \right] = \mathbb{E}_x \left[\max_{i \in G_k} |x_i|^2 \right] \leq C \log(2g) \cdot \max_{i \in G_k} \mu_i^2, \tag{30}$$

where $\mu_i^2 = \mathbb{E}_x[x_i^2]$ denotes the second moment of channel i. Substituting this bound into Eq. (29), we get

$$\mathcal{E}(P) \leq \frac{Cg \log(2g)}{12Q^2} \sum_{k=1}^{K} \max_{i \in G_k} \mu_i^2. \tag{31}$$

We also need a matching lower bound. For any group G_k, let

$$i_k^* \in \arg\max_{i \in G_k} \mu_i^2. \tag{32}$$

Since

$$M_k^2 = \max_{i \in G_k} |x_i|^2 \geq |x_{i_k^*}|^2, \tag{33}$$

taking expectation over x gives

$$\mathbb{E}_x \left[M_k^2 \right] \geq \mathbb{E}_x \left[|x_{i_k^*}|^2 \right] = \mu_{i_k^*}^2 = \max_{i \in G_k} \mu_i^2. \tag{34}$$

Substituting this into Eq. (29), we obtain

$$\mathcal{E}(P) \geq \frac{g}{12Q^2} \sum_{k=1}^{K} \max_{i \in G_k} \mu_i^2. \tag{35}$$

Define the proxy objective

$$\Phi(P) := \sum_{k=1}^{K} \max_{i \in G_k} \mu_i^2. \tag{36}$$

Equations (31) and (35) imply the sandwich bound

$$\frac{g}{12Q^2} \Phi(P) \leq \mathcal{E}(P) \leq \frac{Cg \log(2g)}{12Q^2} \Phi(P). \tag{37}$$

By the optimality of second-moment sorting shown in Appendix B.2, P_{sort} minimizes the proxy objective $\Phi(P)$. Therefore, for any valid partition P,

$$\Phi(P_{\text{sort}}) \leq \Phi(P). \tag{38}$$

Let

$$P^* \in \arg\min_P \mathcal{E}(P) \tag{39}$$

be the optimal partition with respect to $\mathcal{E}(P)$. Since P_{sort} minimizes $\Phi(P)$, we have

$$\Phi(P_{\text{sort}}) \leq \Phi(P^*). \tag{40}$$

Using the upper bound in Eq. (37), we get

$$\mathcal{E}(P_{\text{sort}}) \leq \frac{Cg \log(2g)}{12Q^2} \Phi(P_{\text{sort}}) \leq \frac{Cg \log(2g)}{12Q^2} \Phi(P^*). \tag{41}$$

Using the lower bound in Eq. (37) on P^* gives

$$\Phi(P^*) \leq \frac{12Q^2}{g} \mathcal{E}(P^*). \tag{42}$$

Combining the two inequalities yields

$$\mathcal{E}(P_{\text{sort}}) \leq C \log(2g) \cdot \mathcal{E}(P^*) = C \log(2g) \cdot \min_P \mathcal{E}(P). \tag{43}$$

This proves that second-moment based sorting is a $C \log(2g)$-approximation to the optimal partition with respect to the expected quantization error under the uniform-noise approximation.

C Additional Detail for Run-time Free Deployment.

As illustrated in Fig. 4(b), the activation-side permutation before each linear layer can be absorbed either into its preceding linear layer or into its preceding normalization layer. The weight-side permutation can also be pre-applied offline. Therefore, PermuQuant does not require an explicit permutation operator during inference. Here, we provide the technical details for the normalization case. In particular, we show that channel reordering can be fused into RMSNorm or LayerNorm with negligible additional runtime cost. This is possible because both RMSNorm and LayerNorm are permutation-equivariant along the normalized channel dimension. Let $P \in \{0,1\}^{d\times d}$ denote a channel permutation matrix acting on a token vector $x \in \mathbb{R}^d$, where d is the channel dimension.

RMSNorm. RMSNorm computes

$$\mathrm{RMSNorm}(x;\gamma) = \frac{x}{\sqrt{\frac{1}{d}\sum_{i=1}^{d} x_i^2 + \varepsilon}} \odot \gamma, \tag{44}$$

where $\gamma \in \mathbb{R}^d$ is the channel-wise scale. Since a permutation does not change the sum of squares,

$$\sum_{i=1}^{d} (Px)_i^2 = \sum_{i=1}^{d} x_i^2, \tag{45}$$

the normalization factor is invariant to channel reordering. Therefore,

$$P\,\mathrm{RMSNorm}(x;\gamma) = \mathrm{RMSNorm}(Px;P\gamma). \tag{46}$$

LayerNorm. LayerNorm computes

$$\mathrm{LN}(x) = \frac{x-\mu(x)}{\sqrt{\sigma^2(x)+\varepsilon}}, \qquad \mu(x) = \frac{1}{d}\sum_{i=1}^{d} x_i, \qquad \sigma^2(x) = \frac{1}{d}\sum_{i=1}^{d}(x_i-\mu(x))^2. \tag{47}$$

Both the mean and variance are invariant to channel permutation:

$$\mu(Px) = \mu(x), \qquad \sigma^2(Px) = \sigma^2(x). \tag{48}$$

Thus,

$$P\,\mathrm{LN}(x) = \mathrm{LN}(Px). \tag{49}$$

If LayerNorm is followed by channel-wise modulation,

$$y = (1+s)\odot \mathrm{LN}(x) + b, \tag{50}$$

then the same permutation can be applied to the modulation vectors:

$$Py = (1+Ps)\odot \mathrm{LN}(Px) + Pb. \tag{51}$$

Therefore, for LayerNorm with adaptive scale and shift, channel reordering can be moved before normalization as long as the modulation vectors are reordered accordingly.

Why the fusion is nearly zero-cost. A naive implementation first computes normalization and then launches a separate gather kernel:

$$x \xrightarrow{\text{Norm}} y \xrightarrow{\text{Reorder}} y_P. \tag{52}$$

This introduces an additional activation read, an additional activation write, and an extra kernel launch. In contrast, our fused implementation performs

$$x \xrightarrow{\text{indexed load}} x_P \xrightarrow{\text{Norm in registers}} y_P \tag{53}$$

within a single normalization kernel.

The key observation is that normalization already needs to read the full input row. We therefore replace the original sequential load with an indexed load according to the permutation, while keeping the remaining computation unchanged. Specifically, the fused kernel:

1. loads one activation row as $x[\pi(1)],\ldots,x[\pi(d)]$ from global memory;

2. computes the RMS statistic or the mean and variance on the loaded values;
3. applies the corresponding channel-wise scale or modulation;
4. writes the reordered normalized output contiguously.

In this way, the reordering is absorbed into the mandatory input-read stage of normalization. The fused kernel avoids a standalone reorder pass over the activation tensor. The only additional work is reading the permutation indices and generating indexed memory addresses. This is much cheaper than materializing an intermediate tensor and performing an extra global-memory gather and writeback. Thus, the activation-side permutation is nearly zero-cost at runtime. For adaptive LayerNorm, which is commonly used in DiT Models [41], the same strategy applies to the dynamic modulation parameters. During the kernel load stage, the per-channel scale and shift are gathered on the fly using the same permutation. This avoids materializing reordered modulation tensors through a separate pass.

Implementation detail. In our Triton implementation, each program processes one row of the normalized tensor. When reordering is enabled, the kernel first loads the permutation indices $\pi(j)$. It then directly issues indexed global-memory loads, rather than materializing a reordered tensor:

$$x_j \leftarrow x[\pi(j)]. \tag{54}$$

The subsequent normalization and channel-wise affine or modulation operations are computed on these loaded values. The final output is stored contiguously. As a result, channel reordering does not appear as a separate operator. It is absorbed into the row load of RMSNorm or LayerNorm.

D Reordering in the Hadamard-Transformed Basis

In PermuQuant-H, reordering is performed after the Hadamard transformation. This is because per-group quantization is applied to the Hadamard-transformed representation. Let $\mathbf{H}$ denote the Hadamard transform and $\mathbf{P}$ denote the permutation matrix. If reordering is applied before the Hadamard transformation, the representation becomes $\mathbf{xPH}$. The Hadamard transform mixes the permuted channels again. Thus, the pre-Hadamard order does not directly determine the contiguous groups that are quantized. In contrast, applying reordering after the Hadamard transformation gives $\mathbf{xHP}$. The permutation then directly determines which transformed channels share a per-group quantization scale. Therefore, the reordering is aligned with the actual representation being quantized.

This design does not require explicitly storing the matrix $\mathbf{HP}$. In practice, the Hadamard transform is implemented by a fast Hadamard transform rather than by materializing $\mathbf{H}$. The post-Hadamard permutation can be fused into the output layout of the Hadamard kernel. Equivalently, the Hadamard computation is kept unchanged, and only the order in which transformed channels are written or consumed is modified. The additional cost is therefore limited to lightweight index mapping, rather than an extra matrix multiplication. When implemented as part of the Hadamard, reordering avoids an extra global-memory pass and preserves the runtime-free property up to negligible indexing overhead.

E Implementation Details

We conduct experiments under the 3-bit setting. We adopt per-group symmetric quantization for both activations and weights. For FLUX.1-dev, we use a group size of 64. For Z-Image-Turbo and SANA-1.5-1.6B, we use a group size of 32. For joint second-moment reordering, we search α from $\{0, 0.2, 0.4, 0.6, 0.8, 1.0\}$ and select the value with the smallest calibration quantization error. The acceptance threshold τ is set to 0 by default. For ConvRot, the group size of grouped rotations is set to 64. For SmoothQuant, we compute the smoothing scales following its original formulation. For each model, the smoothing strength α is selected by grid search on the calibration set. For SVDQuant, we set the rank of the low-rank branch to 32, following the original SVDQuant setting.

F Limitations

PermuQuant still has several limitations. First, the current reordering strategy is heuristic. Although our analysis provides an approximation guarantee under reasonable assumptions, the resulting permutation is not guaranteed to be globally optimal. Future work may formulate channel reordering

as an explicit optimization problem, enabling better permutation search or even optimal permutation design under practical constraints. Second, PermuQuant relies on calibration data to estimate channel statistics and decide whether to apply reordering. As in other PTQ methods, the quality of these estimates may depend on the representativeness of the calibration set. Finally, our current method mainly focuses on linear layers in diffusion transformers. Applying reordering to attention quantization remains unexplored. Extending the reordering to these attention-specific quantization steps is an interesting and practical direction for future work.

G Broader Impact

PermuQuant improves the efficiency of large diffusion models by reducing memory usage and inference latency under low-bit quantization. This may lower the hardware barrier for deploying text-to-image generation models and reduce the energy cost of inference. Therefore, our method can benefit researchers and practitioners with limited computational resources.

However, more efficient image generation may also lower the cost of misuse, such as generating misleading, harmful, or copyrighted visual content. Our work does not directly address these risks. Practical deployment should therefore be accompanied by appropriate safeguards, including content filtering, provenance tracking, watermarking, and responsible dataset governance.

Table 3: Ablation study of reordering and Hadamard transformation on MJHQ30K and GenEval. For MJHQ30K, we report FID, KID$\times 10^3$, and ImageReward (IR); for GenEval, we report the standard scores. The best results among INT W3A3 variants are highlighted in **bold**.

Model	Precision	Reordering	Hadamard	MJHQ30K			GenEval						
				FID ↓	KID ↓	IR ↑	Single ↑	Two ↑	Count ↑	Color ↑	Pos. ↑	Attr. ↑	Overall ↑
Z-Image -Turbo (8 Steps)	BF16	–	–	67.6	5.7	1.001	1.00	0.97	0.71	1.00	0.63	0.83	0.86
	INT W3A3	✗	✗	250.8	183.0	-1.609	0.69	0.21	0.21	0.47	0.21	0.10	0.32
		✓	✗	134.9	49.2	-0.163	**0.96**	0.87	0.61	0.93	0.55	0.50	0.74
		✗	✓	84.2	**12.0**	0.608	**0.96**	0.95	**0.71**	**1.00**	0.61	**0.78**	**0.83**
		✓	✓	**80.4**	12.3	**0.715**	**0.96**	**0.97**	0.64	**1.00**	**0.63**	0.75	**0.83**
FLUX.1 -dev (50 Steps)	BF16	–	–	60.6	2.8	0.917	1.00	0.89	0.61	0.93	0.68	0.53	0.77
	INT W3A3	✗	✗	131.0	52.4	0.386	0.81	0.45	0.29	0.67	0.32	0.10	0.44
		✓	✗	74.9	9.1	0.726	0.96	0.84	0.50	**0.97**	0.58	0.40	0.71
		✗	✓	68.4	5.0	0.730	0.96	0.87	**0.64**	0.90	0.68	**0.63**	**0.78**
		✓	✓	**64.7**	**4.6**	**0.759**	**1.00**	**0.92**	0.54	0.93	**0.74**	0.48	0.77
SANA-1.5 -1.6B (20 Steps)	BF16	–	–	58.1	1.2	1.075	1.00	0.97	0.79	0.97	0.66	0.55	0.82
	INT W3A3	✗	✗	103.6	30.1	0.394	0.96	0.63	0.54	0.83	0.47	0.43	0.64
		✓	✗	82.8	14.7	0.700	0.92	0.76	**0.61**	0.93	0.45	0.60	0.71
		✗	✓	72.6	6.0	0.902	**1.00**	0.82	0.57	0.93	0.50	**0.75**	0.76
		✓	✓	**68.0**	**5.5**	**0.919**	0.96	**0.89**	0.57	**0.97**	**0.66**	0.73	**0.80**

H Additional Results

H.1 Ablation of Hadamard Transform.

Table 3 studies the effect of channel reordering and the Hadamard transform under INT W3A3 quantization. We compare four variants: the original W3A3 baseline, reordering only, Hadamard only, and their combination. Across all three models, both reordering and Hadamard transformation consistently improve over the direct W3A3 baseline. For example, on Z-Image-Turbo, reordering alone reduces FID from 250.8 to 134.9, while Hadamard alone further reduces it to 84.2. This shows that both operations effectively reduce low-bit quantization error.

More importantly, combining reordering with the Hadamard transform gives the best overall performance in most cases. On Z-Image-Turbo, the combined variant achieves the best FID, ImageReward, and GenEval overall score among INT W3A3 variants. Similar trends are observed on FLUX.1-dev and SANA-1.5-1.6B, where the combined variant obtains the lowest FID and KID, and also gives the strongest ImageReward. These results indicate that reordering is complementary to Hadamard-based rotation. While Hadamard transformation redistributes channel information globally, our reordering further improves the induced per-group partitions after rotation.

The two components also differ in efficiency cost. Reordering introduces almost no additional inference overhead, since the permutations can be folded into adjacent modules or pre-applied to the weights offline. By contrast, Hadamard transformation cannot always be fully fused in modern diffusion architectures. This is because these models commonly use AdaLN-based modulation [41], where part of the transformation interacts with input-dependent scaling and shifting operations. As a result, some Hadamard matrices must be applied during inference, which increases the end-to-end latency by about 8% in our implementation. Thus, Hadamard transformation introduces an efficiency–quality trade-off in practice. Reordering alone is preferable when latency is critical, while the combined variant provides better quality when a small latency increase is acceptable.

H.2 Comparison with Grouping and Reordering Methods

IGQ-ViT [37] and RPTQ [63] group channels with similar activation statistics to share quantization parameters. IGQ-ViT employs instance-aware dynamic grouping, whereas RPTQ uses offline range-based grouping. Their groups can contain unequal numbers of channels, which makes them difficult to map directly to standard fixed-size per-group quantization kernels. In contrast, PermuQuant optimizes channel ordering under an equal-sized contiguous grouping constraint. This preserves compatibility with standard per-group kernels. Our theoretical analysis further provides an error upper bound and an approximation guarantee for the ordering problem under the stated assumptions.

Table 4: Comparison with grouping and reordering baselines under INT W3A3 quantization. We report FID, KID$\times 10^3$, and ImageReward (IR) on MJHQ30K, and the GenEval score (GE). PermuQuant is evaluated without Hadamard transformation.

Model	Method	Kernel Compat.	FID ↓	KID ↓	IR ↑	GE ↑
Z-Image -Turbo	RTN	Yes	250.8	183.0	-1.609	0.32
	IGQ-ViT	No	140.2	53.7	-0.190	0.72
	RPTQ	No	206.1	129.4	-0.983	0.45
	PermuQuant	Yes	**134.9**	**49.2**	**-0.163**	**0.74**
FLUX.1 -dev	RTN	Yes	131.0	52.4	0.386	0.44
	IGQ-ViT	No	85.9	23.5	0.581	0.68
	RPTQ	No	88.1	25.6	0.607	0.53
	PermuQuant	Yes	**74.9**	**9.1**	**0.726**	**0.71**
SANA-1.5 -1.6B	RTN	Yes	103.6	30.1	0.394	0.64
	IGQ-ViT	No	90.3	22.6	0.515	0.65
	RPTQ	No	97.4	26.9	0.479	0.67
	PermuQuant	Yes	**82.8**	**14.7**	**0.700**	**0.71**

Table 5: Validation of $\rho(G_k)$ across all layers and diffusion timesteps. We report the fraction of groups with $\rho(G_k) < 1.2$ and its P95/P99. The worst layer and timestep are selected separately by the largest P99 to highlight unfavorable subsets. Layer identifiers are 8, `single_blocks.9`, and `blocks.8` for the three models, respectively.

Model	Scope	ID	$\Pr(\rho < 1.2)$ ↑	P95 ↓	P99 ↓
Z-Image -Turbo	Overall	–	99.796%	0.8009	1.0607
	Worst layer	8	99.483%	0.7978	1.1225
	Worst timestep	4	99.587%	0.8087	1.0872
FLUX.1 -dev	Overall	–	99.730%	0.8030	1.1110
	Worst layer	9	99.289%	0.9340	1.1590
	Worst timestep	24	99.317%	0.9260	1.1480
SANA-1.5 -1.6B	Overall	–	99.760%	0.7720	1.0310
	Worst layer	8	99.445%	0.8780	1.1260
	Worst timestep	10	99.426%	0.8890	1.1240

We compare PermuQuant with IGQ-ViT and RPTQ baselines on Z-Image-Turbo, FLUX.1-dev, and SANA-1.5-1.6B. For each model, we use the same quantizer, bit-width, calibration data, and number of groups across methods. While the number of groups is matched, PermuQuant uses equally sized groups, whereas IGQ-ViT and RPTQ baselines allow group sizes to vary. All comparisons use INT W3A3 quantization, and PermuQuant is evaluated without the Hadamard transformation to isolate the contribution of channel reordering. As shown in Table 4, PermuQuant achieves the best FID, KID, ImageReward, and GenEval overall scores among the compared methods on all three models. These results demonstrate the effectiveness of channel reordering under the fixed-size grouping constraint.

H.3 Further Empirical Validation of the Extremal-Control Assumption

We further validate the extremal-control assumption in Equation (5) by examining $\rho(G_k)$ across all layers and diffusion timesteps of Z-Image-Turbo, FLUX.1-dev, and SANA-1.5-1.6B. For each model, we report the fraction of groups satisfying $\rho(G_k) < 1.2$, together with the 95th and 99th percentiles (P95 and P99) of $\rho(G_k)$. In addition to the overall statistics, we report the layer and timestep with the largest P99, selected separately to reveal unfavorable subsets that may be obscured by aggregation.

As shown in Table 5, more than 99% of groups satisfy $\rho(G_k) < 1.2$ in every reported setting, including the worst layer and timestep for each model. These results provide empirical support for a small extremal-control constant across the evaluated models, layers, and timesteps.

H.4 Visual Quality Results

We provide additional visual comparisons in Figs. 7, 8, and 9. These results complement the qualitative examples in the main paper and cover three representative diffusion backbones: SANA-1.5-1.6B, FLUX.1-dev, and Z-Image-Turbo. For each model, the dashed line separates the BF16 reference from the quantized methods. All quantized methods are evaluated under the W3A3 setting.



Figure 7: Qualitative comparison on SANA-1.5-1.6B. The dashed line separates BF16 from the quantized methods. All quantized methods are evaluated under the W3A3 setting.

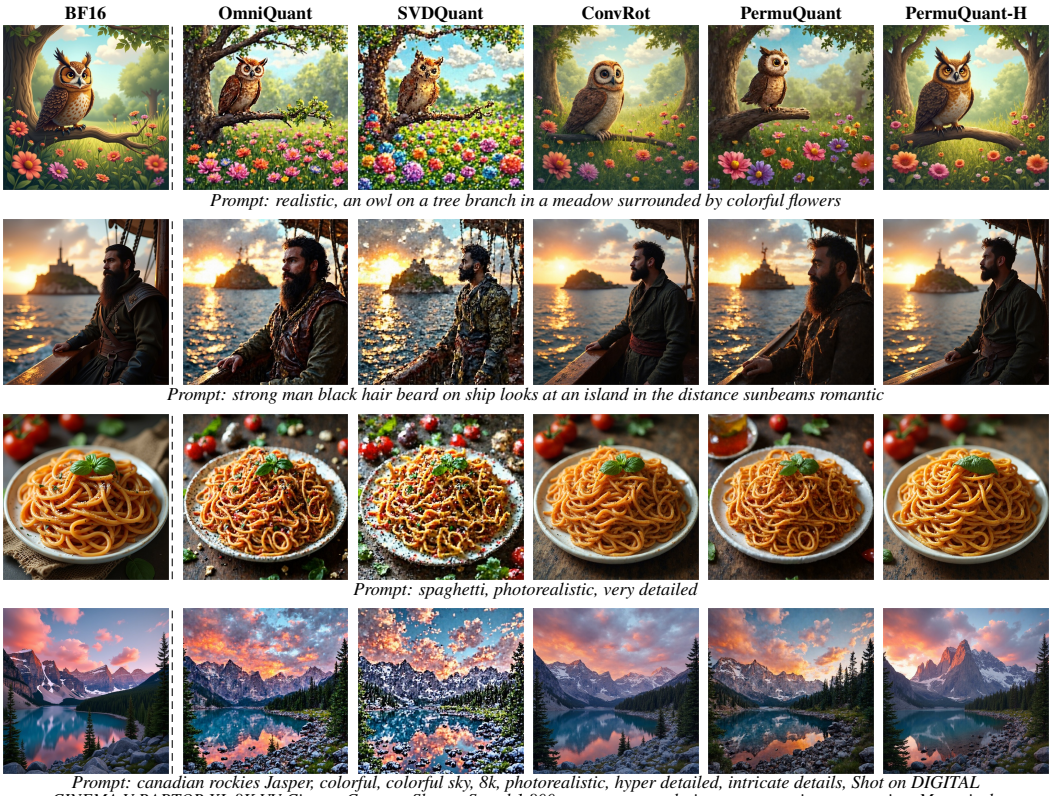


Figure 8: Qualitative comparison on FLUX.1-dev. The dashed line separates BF16 from the quantized methods. All quantized methods are evaluated under the W3A3 setting.



Figure 9: Qualitative comparison on Z-Image-Turbo. The dashed line separates BF16 from the quantized methods. All quantized methods are evaluated under the W3A3 setting.

NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: Please refer to our abstract and introduction.

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: We discuss the limitations of our work in the appendix.

   Guidelines:

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

3. **Theory assumptions and proofs**

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

   Answer: [Yes]

Justification: We state the assumptions used in our theoretical analysis in our paper, including the extremal-control condition and the standard uniform-noise approximation for quantization error. We provide the corresponding derivations and complete proofs in Appendix.

Guidelines:

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

4. **Experimental result reproducibility**

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

Answer: [Yes]

Justification: We have provided implementation details in the experiments section. We will also release all the code and models.

Guidelines:

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

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

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

   Answer: [Yes]

   Justification: Firstly, all data in the paper can be accessed publicly. Secondly, we provide very detailed instructions to reproduce our results. Thirdly, our code is available on GitHub.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: We have provided implementation details, which cover the above questions.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: Please refer to the experiment part.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).

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

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: Please refer to experiment part.

   Guidelines:

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

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: The research conducted in the paper conforms, in every respect, with the NeurIPS Code of Ethics.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Please refer to the appendix.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.

- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

    Answer: [N/A]

    Justification: This work poses no such risks.

    Guidelines:

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

12. **Licenses for existing assets**

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

    Answer: [Yes]

    Justification: We have credited most previous works in the paper. The license and terms are respected properly.

    Guidelines:

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

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

13. **New assets**

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

    Answer: [Yes]

    Justification: We will release code and models. In the paper, we have provided implementation details and other contents to reproduce our results.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper does not involve crowdsourcing nor research with human subjects.

    Guidelines:

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

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

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: The core method development in this research does not involve LLMs as any important, original, or non-standard components.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.