PYRAQUANT: SPATIAL MIXED-PRECISION QUANTIZATION FOR HIGH-RESOLUTION DIFFUSION MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

High-resolution diffusion generation remains expensive because denoising cost grows rapidly with image resolution. Progressive generation reduces this cost by first generating at a low resolution and then progressively increasing the resolution, while existing diffusion quantization methods mainly allocate precision across layers, channels, or denoising steps without exploiting the stage-wise and spatial structure of this process. We find that progressive refinement exhibits two forms of non-uniformity: quantization tolerance varies across resolution stages, and refinement changes differ across spatial regions. Based on these observations, we propose PyraQuant, a training-free precision allocation framework for quantized progressive diffusion models. PyraQuant keeps the base quantizer at the lowest resolution. At each subsequent resolution transition, it assigns each region a precision according to how much it changes, carrying these decisions to the next resolution. The lower-precision configuration is derived directly from the original quantized representation without additional calibration or model copies, and regions requiring limited refinement reuse cached predictions. Experiments show that on FLUX.1-schnell with the SVDQuant W4A4 base quantizer, PyraQuant reduces effective precision from 4.48 to 2.00 bits while maintaining comparable generation quality, and achieves a $1.81\times$ end-to-end speedup on UltraHR-eval4K.

1 INTRODUCTION

High-resolution image generation has become increasingly important in applications such as digital content creation, visual design, and image synthesis (Rombach et al., 2022; Podell et al., 2024). However, generating such images remains expensive because the computation of diffusion models grows rapidly with image resolution. Each denoising step needs to process a much larger number of image tokens, making high-resolution inference a major bottleneck. Recent high-resolution diffusion systems often adopt progressive generation to reduce this cost (Wu et al., 2025a; Qiu et al., 2025; Bu et al., 2025; Koh et al., 2025). Instead of generating the final image directly, these methods first produce a lower-resolution image and then upsample and refine it through higher-resolution stages, avoiding costly early-stage computation at high resolution. However, progressive generation also introduces a non-uniform computation process. Different resolution stages have different computation costs and precision requirements, and the information added during higher-resolution transitions varies across spatial regions.

Quantization provides a complementary way to reduce diffusion inference cost by using lower precision for weights and activations (Shang et al., 2023; Li et al., 2023). Diffusion quantization methods have explored precision assignment across layers, channels, and denoising steps (He et al., 2023; Zhao et al., 2024; Feng et al., 2025). However, these methods treat the generation pipeline as a fixed computation graph and do not consider how precision should change during progressive refinement. As a result, the stage-wise and spatial structure of progressive generation remains unused.

This raises an important question: **How should precision be allocated during progressive high-resolution diffusion?** We first observe a mismatch between computational cost and quantization sensitivity across resolution stages. As shown in Fig. 1, the 4K stage dominates computation cost, while being less sensitive to quantization than earlier stages. The most expensive stage is therefore not necessarily the stage that requires the highest precision. Beyond the stage dimension, higher-resolution refinement also affects different spatial regions unevenly. Some regions receive larger

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

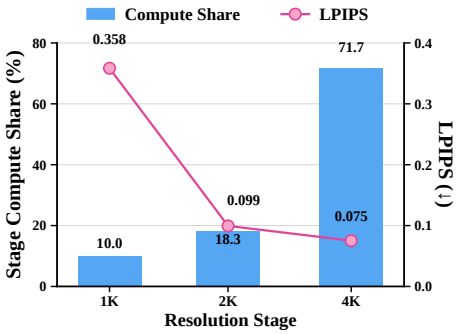

Figure 1: Computation share of each resolution stage (bars) and its quantization sensitivity (line) on FLUX.1-schnell. Sensitivity is the LPIPS between the final outputs obtained with all stages at W4 (SVDQuant) and with only that stage lowered to W3.

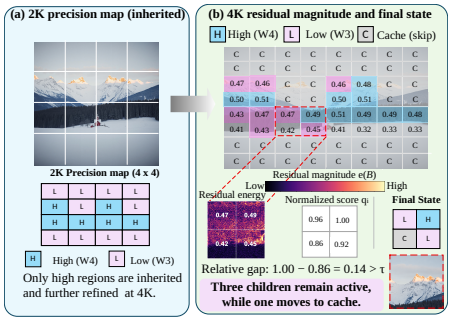

Figure 2: Residual-guided allocation on FLUX.1-schnell ($\tau = 0.1$). (a) Only high-precision 2K regions are inherited and refined at 4K. (b) Within each inherited sibling group, a child whose normalized residual magnitude falls more than τ below the group maximum moves to cache, and the remaining children are split into high and low precision.

updates, and skipping refinement in these regions introduces larger reconstruction errors, while regions with smaller updates remain closer to the inherited prediction. These observations suggest that progressive diffusion requires a precision and computation policy that changes across both resolution stages and spatial regions.

Based on this observation, we propose PyraQuant, a training-free precision allocation framework for quantized progressive diffusion models. PyraQuant operates on top of an existing base quantizer. The lowest-resolution stage uses the base quantizer over the full image, while higher-resolution stages progressively introduce lower-precision computation and cached reuse. At the first higher-resolution stage, residual magnitude assigns regions to high or lower precision. At the next resolution, inherited high-precision regions remain candidates for finer allocation, while the remaining regions can reuse cached predictions. A nested lower-precision configuration is derived directly from the existing quantized representation without storing another model copy or introducing additional calibration.

We evaluate PyraQuant on progressive high-resolution diffusion pipelines with both DiT and UNet backbones. On FLUX.1-schnell, PyraQuant reduces the effective precision of SVDQuant from 4.48 to 2.00 bits and lowers executed computation from 25.3% to 13.6% of the corresponding 16-bit pipeline, while maintaining comparable generation quality and achieving a $1.81\times$ end-to-end speedup. On SDXL, it reduces executed computation from 33.8% to 24.3% and achieves a $1.33\times$ speedup. Across different base quantizers on both backbones, PyraQuant further reduces executed computation by up to 46% relative to the corresponding uniform quantizers while maintaining comparable generation quality.

Our contributions are summarized as follows:

- We propose PyraQuant, a training-free precision allocation framework for quantized progressive diffusion models that extends quantizers across resolution stages and spatial regions.
- We introduce a cascaded allocation strategy that carries spatial decisions across successive resolutions, coupling stage-wise precision assignment with spatial refinement and allowing regions to progressively move from high precision to low precision or cached reuse.
- We design a residual-magnitude-guided spatial assignment rule that uses the changes produced during progressive refinement to determine where different computation options should be applied.

2 Related Work

2.1 Post-Training Quantization for Diffusion Models

Post-training quantization(PTQ) reduces the numerical cost of diffusion inference without retraining the original model. Early diffusion PTQ methods mainly address distribution changes across denoising steps through calibration, reconstruction, or timestep-aware quantization (Shang et al., 2023; Li et al., 2023; He et al., 2023; Huang et al., 2024; Yao et al., 2024). Quantization-aware training can further reduce diffusion weights to 1–2 bits (Sui et al., 2024; Zheng et al., 2025; Lu et al., 2024), but requires retraining the model. These methods focus on learning lower-bit model representations, while our work studies how precision should be allocated during progressive generation.

Recent work further improves the quantizer itself to support lower bit-widths. SVDQuant shifts activation outliers into the weights and absorbs them with a low-rank branch, quantizing the rest to low bits, enabling practical W4A4 inference on modern diffusion models (Li et al., 2025). Other studies reduce quantization error through rotation, selective bit assignment, or specialized calibration (Yang et al., 2025; Mills et al., 2025; Zhang et al., 2026).A separate line of work enables a single model to operate at multiple bit-widths through joint training or shared multi-precision representations (Jin et al., 2020; Yu et al., 2021; Park et al., 2024; Nair et al., 2025). These methods mainly focus on constructing or training a single model that supports multiple deployment precisions.

These studies also show that one quantization setting is not equally suitable for every part of the denoising process. This motivates sensitivity-aware precision allocation. ViDiT-Q studies quantization behavior across transformer layers and denoising steps, while MixDQ assigns different bit-widths to model components according to their sensitivity (Zhao et al., 2025b; 2024). Other methods adapt quantization according to activation statistics or module properties (Ryu et al., 2025; Wu et al., 2024). Their decisions, however, are still mainly made along dimensions such as layers, channels, and denoising timesteps.

Spatially varying precision has also been explored in diffusion models and related vision tasks, where different numerical formats are assigned to tokens or image regions according to local importance (Wang et al., 2025; Hong et al., 2022; Kim et al., 2026). However, these decisions are usually made within a fixed-resolution model or a single forward process. Progressive diffusion introduces an additional structure because the same image continues to evolve across successive resolution stages.

2.2 Quantization-Based Feature Reuse For Diffusion Inference

Recent studies further investigate the interaction between numerical compression and feature reuse. CacheQuant combines quantization with feature caching for diffusion models and analyzes the coupled effect of quantization errors and cached features during inference (Liu et al., 2025). By jointly considering precision reduction and feature reuse, it demonstrates the potential of combining numerical and computational optimization in diffusion acceleration. Similar ideas have also been explored in video generation, where QuantCache introduces importance-guided quantization together with hierarchical latent and layer caching (Wu et al., 2025b). However, these methods mainly exploit redundancy within a denoising trajectory, where adaptation decisions are made across timesteps or intermediate representations. Progressive high-resolution diffusion introduces a different dependency structure, where each resolution stage inherits and refines the previous output. PyraQuant leverages this cross-resolution hierarchy to allocate precision and computation across resolution stages and spatial regions according to refinement changes.

3 Method

Given a progressive diffusion model with an existing base quantizer Q, PyraQuant performs training-free allocation across resolution stages and spatial regions without redesigning or recalibrating Q. Instead, it decides how different computation options should be used as generation moves from low to high resolution. Our design uses two properties of progressive generation. First, different resolution stages play different roles in the coarse-to-fine process. Second, the changes introduced during each resolution transition are not uniform across the image.

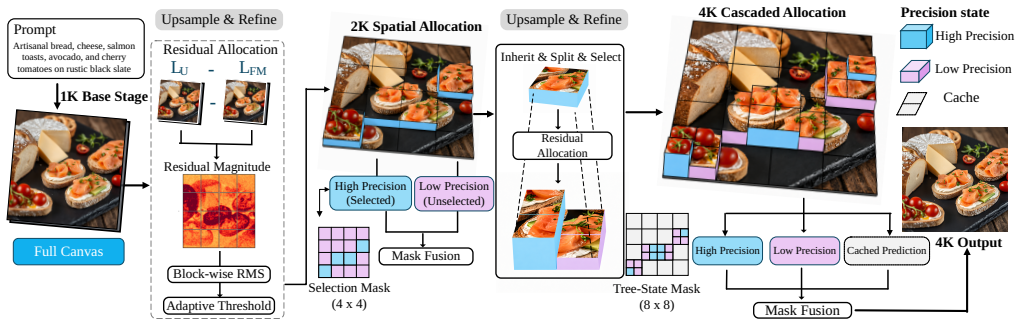

Figure 3: Overview of the PyraQuant framework.

As shown in Fig. 3, PyraQuant follows the coarse-to-fine structure of the progressive pipeline. The 1K stage uses the base quantizer over the full canvas. During the transition to 2K, PyraQuant computes the residual between the latent produced by frequency mixing and the upsampled latent from the previous stage. It then uses the residual magnitude in each region to assign high or low precision before denoising at the new resolution. This map is then carried to 4K, where inherited high-precision regions are further allocated according to the smallest region the backbone can process separately, while regions outside the inherited path directly use cache. The final map contains high-precision, low-precision, and cached regions.

3.1 Stage-Aware Computation Reduction

Progressive generation first establishes the coarse image and then adds information at higher resolutions. PyraQuant therefore expands the available computation options as the resolution increases instead of applying one quantization setting throughout the pipeline.

Let Q_{hi} denote the high-precision configuration of the base quantizer, Q_{lo} the derived lower-precision configuration, and C cached reuse. For the three-stage setting in Fig. 3, we define

$$\mathcal{S}_{1K} = \{Q_{\mathrm{hi}}\}, \qquad \mathcal{S}_{2K} = \{Q_{\mathrm{hi}}, Q_{\mathrm{lo}}\}, \qquad \mathcal{S}_{4K} = \{Q_{\mathrm{hi}}, Q_{\mathrm{lo}}, C\}. \tag{1}$$

The base stage uses the original quantized model over the full image, while later stages progressively introduce lower-precision computation and cached reuse. The two numerical configurations are derived from the same base quantizer,

$$Q_{\mathrm{hi}} = Q, \qquad Q_{\mathrm{lo}} = \mathcal{N}(Q_{\mathrm{hi}}), \tag{2}$$

where $\mathcal{N}$ is a nested mapping over the quantized codes, in the spirit of any-precision and Matryoshka quantization (Jin et al., 2020; Yu et al., 2021; Park et al., 2024; Nair et al., 2025). It forms a smaller codebook by merging nearby levels of the base code without introducing an independently calibrated quantization scale. The lower-precision configuration therefore requires neither another copy of the weights nor calibration. Other parts of the base quantizer remain unchanged. This allows PyraQuant to work with different underlying quantization methods without changing its allocation rule.

Cache represents a distinct computation option rather than another lower-precision representation. A cached region reuses information already available from the progressive pipeline and avoids backbone evaluation at the current stage. On FLUX, we find that W3 recomputation remains more accurate than cached reuse, while W2 recomputation becomes less accurate than reuse despite still requiring backbone computation. We therefore use W3 as the lowest numerical precision and cached reuse as the terminal option for regions that require little further refinement. Detailed results are provided in Appendix C.2.

3.2 Residual-Guided Spatial Assignment

Stage-aware computation options define the available choices, but they do not specify where high or low precision should be used. PyraQuant obtains this spatial signal directly from the resolution transition. Spatially adaptive diffusion methods have explored region-level computation decisions based on local information (Liu et al., 2026; Lin et al., 2026).

At each resolution transition, the progressive pipeline upsamples the previous-stage latent to L_U and constructs the starting latent L_{FM} through latent frequency mixing before denoising at the new resolution (Koh et al., 2025). Since both latents are already available, we define their difference as the transition residual and summarize its magnitude within each region by the root-mean-square over its D latent channels,

$$R = L_{\mathrm{FM}} - L_U, \qquad e(B) = \sqrt{\frac{1}{|B|D} \sum_{p \in B} \sum_{c=1}^{D} R_{c,p}^2}. \tag{3}$$

A larger residual magnitude indicates that the current resolution transition changes region B more strongly relative to the inherited prediction. We use this quantity as a signal of how much further refinement a region requires, rather than as a direct measure of quantization sensitivity. In a diagnostic study, the within-image Spearman correlation between $e(B)$ and the reconstruction error caused by skipping the corresponding region is 0.71 on average and positive for every image, and the selection ablation on 500 prompts (Section 4.5) confirms this. Regions with smaller residual magnitude are therefore better candidates for reuse, while regions with larger residual magnitude should remain active for further refinement. The score is computed from existing pipeline tensors without an additional model forward pass or a learned predictor. Additional analysis is provided in Appendix C.3.

Because the absolute residual scale varies across prompts and stages, PyraQuant compares nearby regions relative to one another. For four sibling regions with residual scores $\{e_i\}_{i=1}^{4}$, we normalize them within the sibling group,

$$q_i = \frac{e_i}{\max_j e_j}. \tag{4}$$

A sibling group is separated only when

$$1 - \min_i q_i \geq \tau. \tag{5}$$

At the 2K stage, this relative-gap rule determines the two numerical configurations. When the condition in Eq. 5 holds, a child uses Q_{hi} if $q_i > 1 - \tau$, while the remaining children use Q_{lo}. Otherwise, the entire sibling group remains at Q_{hi}. This local rule avoids relying on one absolute residual scale and compares only regions that share the same parent.

The threshold τ is selected on an independent calibration set to meet the execution budget and fixed during evaluation. It only adjusts the allocation policy and does not recalibrate the base quantizer.

All regions are recomputed at 2K, so the rule produces a high- and low-precision map rather than a cache mask. Fig. 2(a) shows this 2K allocation, where the high-precision regions define the spatial path inherited by the next resolution.

3.3 Cascaded Allocation and Cached Reuse

PyraQuant does not construct a new precision map over the full canvas at every resolution. Only regions retained by the previous stage continue into finer allocation. Let $\mathcal{H}_s$ denote the high-precision regions at stage s. Their children form the candidate set at the next resolution,

$$\mathcal{C}_{s+1} = \bigcup_{B \in \mathcal{H}_s} \mathrm{Split}(B). \tag{6}$$

where $\mathrm{Split}(B)$ denotes the four children of region B at the next resolution. This inheritance couples the resolution-stage decision with spatial allocation. Regions outside the inherited candidate set move directly to cached reuse. On the DiT pipeline, the inherited children are further selected at the finer spatial granularity using the residual at the new resolution. On the UNet pipeline, the minimum executable unit contains one complete sibling group, so all children of an inherited high-precision region remain active and are only divided between Q_{hi} and Q_{lo}.

For the DiT pipeline, Fig. 2(b) shows the transition from 2K to 4K. Gray cells mark regions that are no longer active, while remaining cells show the residual magnitude measured at 4K. The inset expands one inherited 2×2 sibling group. In this example, the relative gap is larger than τ. The child with a sufficiently smaller relative score therefore moves to cached reuse, while the other three children remain active. Among the regions that remain active at 4K, PyraQuant further assigns the

two numerical configurations according to their residual magnitude. Let $\mathcal{A}_{4K}$ denote the active set. We assign

$$B \in \mathcal{H}_{4K} \iff e(B) \geq \underset{B' \in \mathcal{A}_{4K}}{\text{median}}\, e(B'), \tag{7}$$

while the remaining active regions use Q_{lo}.

Cached reuse participates directly in the normal diffusion update. Feature reuse methods have shown that cached intermediate information can reduce repeated diffusion computation by avoiding redundant model execution (Ma et al., 2024; Zou et al., 2025; Liu et al., 2025). Let $\hat{x}_0^{\text{cache}}$ denote the cached clean-latent reference, initialized with L_{FM} at the start of the stage. At timestep t, let z_t denote the noisy latent, Π_t the scheduler-dependent mapping, m_C the cache mask, and y_t^{model} the prediction from the computed regions. The cached and final predictions are

$$y_t^{\text{cache}} = \Pi_t(z_t, \hat{x}_0^{\text{cache}}), \qquad \hat{y}_t = (1 - m_C) \odot y_t^{\text{model}} + m_C \odot y_t^{\text{cache}}. \tag{8}$$

For flow matching (Lipman et al., 2023; Liu et al., 2023), $\Pi_t(z_t, x) = (z_t - x)/\sigma_t$. Cached regions therefore bypass the backbone and retain the stage-entry clean-latent reference, while active regions are evaluated with their assigned precision. The two outputs are combined before the standard diffusion update. Computation is restricted to active regions through *routing*: only crops of active regions, each padded with a small halo for boundary context, are processed by the backbone and written back to the canvas, and the halo computation forms the routing overhead (Appendix A.3).

4 Experiments

4.1 Experimental Setup

We evaluate PyraQuant on two official ScaleDiff pipelines (Koh et al., 2025), built on FLUX.1-schnell (Black Forest Labs, 2024) and SDXL (Podell et al., 2024), covering DiT (Peebles & Xie, 2023) and UNet backbones. Both generate images progressively from 1K to 2K and 4K, using $4/2/2$ and $50/20/20$ denoising steps, respectively. We evaluate the final 4096^2 outputs on the 2,000 prompts of UltraHR-eval4K (Zhao et al., 2025a). All methods use the same prompt and seed for each sample. On FLUX, PyraQuant is built on the released SVDQuant W4A4 model (Li et al., 2025). On SDXL, we use a round-to-nearest (RTN) W8A8 base quantizer with group size 32.

We report FID (Heusel et al., 2017), Kernel Inception Distance (KID) (Bińkowski et al., 2018), Inception Score (IS) (Salimans et al., 2016), CLIPScore (Hessel et al., 2021), HPSv2 (Wu et al., 2023), and ImageReward (Xu et al., 2023) for generation quality. For quality evaluation, the high- and low-precision configurations are simulated using quantization-dequantization (QDQ), while cached regions reuse the cached predictions defined by the allocation map. Latency is measured with a separate deployment executor that uses packed INT8 kernels for active regions, so QDQ operations are not included in the reported runtime. Unless otherwise stated, we report executed bit operations (BOPs) based on the computation actually sent to the backbone, including routing overhead, normalized by the corresponding 16-bit pipeline. Quality samples are generated on NVIDIA A5000 GPUs with CPU offloading. End-to-end latency and peak GPU memory are measured on a single NVIDIA L40S 48 GB GPU over 20 prompts after one warm-up run, with model weights resident in GPU memory and without CPU offloading. All baselines are training-free post-training quantizers, and each retains its original activation quantization scheme and calibration procedure when adapted to the ScaleDiff pipeline.

4.2 Compression and Quality Evaluation

Existing quantizers already provide strong low-precision representations for diffusion models. We ask whether the progressive structure provides additional room for compression after the base quantizer has been fixed. In particular, we ask whether PyraQuant can reduce the effective precision used during generation without requiring another model copy or causing substantial image-quality degradation. Table 1 summarizes this comparison on both progressive pipelines.

The main result is that PyraQuant further reduces the effective precision and computation of the base quantizer. On ScaleDiff-FLUX.1-schnell, SVDQuant uses 4.48 effective bits and 25.3% BOPs, while PyraQuant reduces them to 2.00 and 13.6%, respectively. On ScaleDiff-SDXL, PyraQuant reduces

Table 1: Compression and quality comparison on UltraHR-eval4K with 2,000 prompts at 4096^2. Storage bit denotes average weight precision, while Effective bit denotes computation-weighted precision after allocation and caching. BOPs are normalized to the corresponding 16-bit pipeline. For FID and KID, "r" uses UltraHR references and "b" uses matched 16-bit outputs. Best results among quantized methods are bolded.

Method	Precision W / A	Storage (bit)	Effective bit ↓ (bit)	BOPs (%)	FID ↓ r / b	KID ($\times 10^3$) ↓ r / b	IS ↑	CLIP ↑	HPSv2 ↑	IR ↑
ScaleDiff-FLUX.1-schnell										
ScaleDiff-FLUX.1-schnell	bf16	16.00	16.00	100.0	45.6 / –	12.2 / –	12.15	27.04	0.2800	0.648
SVDQuant (Li et al., 2025)	W4 / A4	4.48	4.48	25.3	44.0 / **15.6**	10.7 / −0.1	12.46	27.10	0.2806	0.648
ViDiT-Q-style (Zhao et al., 2025b)	W4/W8 / A8	6.08	6.08	33.9	45.7 / 19.2	12.2 / 0.3	12.00	27.01	0.2809	0.635
SegQuant-G (Zhang et al., 2026)	W4 / A8	4.55	4.55	31.7	44.0 / 16.6	11.1 / −0.2	12.39	27.09	0.2812	**0.670**
PyraQuant on SVDQuant	W4 | W3 | C / A4	**4.48**	**2.00**	**13.6**	**39.7** / 17.5	**7.0** / 1.0	**12.52**	**27.11**	**0.2833**	0.658
ScaleDiff-SDXL										
ScaleDiff-SDXL	fp16	16.00	16.00	100.0	37.3 / –	3.8 / –	12.40	28.20	0.2929	0.749
SmoothQuant (Xiao et al., 2023)	W8 / A8	8.01	8.01	34.9	**36.1** / 18.5	3.7 / 1.8	11.99	**28.51**	0.2860	0.697
SegQuant-G (Zhang et al., 2026)	W8 / A8	9.05	9.05	51.9	37.1 / 13.1	3.7 / −0.3	12.21	28.22	**0.2928**	0.742
CacheQuant-style (Liu et al., 2025)	W8 / A8	8.02	**1.60**	**7.0**	112.1 / 110.5	65.3 / 64.6	7.08	19.00	0.2205	−1.886
RTN-W8	W8 / A8	8.45	8.45	33.8	37.1 / **10.5**	**3.6** / −0.2	**12.65**	28.22	0.2925	0.739
PyraQuant on RTN-W8	W8 | W4 | C / A8	8.45	5.33	24.3	37.3 / 11.9	3.8 / 0.0	12.40	28.06	0.2904	**0.768**
PyraQuant on RTN-W4	W4 | W3 | C / A8	**4.97**	3.88	20.0	36.8 / 18.7	4.0 / 0.6	12.30	28.02	0.2865	0.731

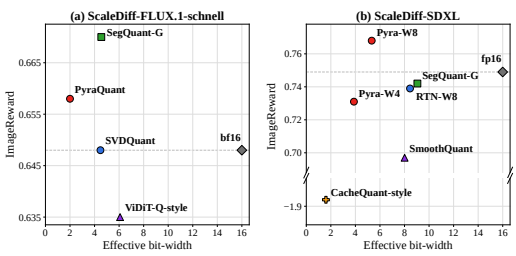

Figure 4: Compression-quality tradeoff on FLUX.1-schnell and SDXL. Left-panel quantized methods mainly use INT4. In the right panel, Pyra-W8 and Pyra-W4 denote PyraQuant on the RTN-W8 and RTN-W4 bases, while the other quantized methods use INT8.

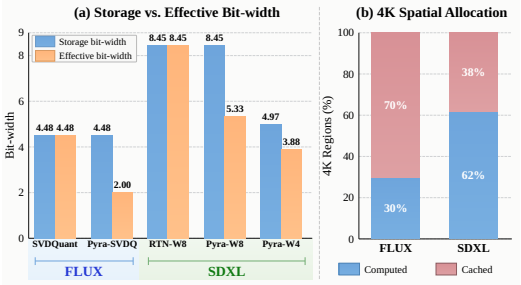

Figure 5: Sources of compression in PyraQuant. (a) Storage and effective bit-width before and after allocation. (b) 4K spatial allocation between recomputed and cached regions. Pyra-SVDQ denotes PyraQuant on SVDQuant.

the effective bit of RTN-W8 from 8.45 to 5.33 and its BOPs from 33.8% to 24.3%. Using RTN-W4 as the base further lowers the effective bit to 3.88 and BOPs to 20.0%.

This reduction in effective precision results in only limited changes in generation quality. On FLUX, PyraQuant reaches 2.00 effective bits while HPSv2 remains at 0.2833, compared with 0.2806 for SVDQuant and 0.2800 for the bf16 reference. On SDXL, the 5.33-bit configuration reaches 0.2904 HPSv2, close to the fp16 reference at 0.2929. The more aggressive 3.88-bit configuration shows a larger quality change, which is expected as both stored and effective precision are reduced further. The results therefore show a gradual compression-quality tradeoff.

Figure 4 compares the quality–compression tradeoff using ImageReward. On FLUX, PyraQuant reduces the effective bit-width to 2.00 while achieving an ImageReward of 0.658, above both the bf16 reference and SVDQuant. On SDXL, PyraQuant with the RTN-W8 base achieves the highest ImageReward at 5.33 effective bits. The lower-storage RTN-W4 configuration further reduces the effective bit-width to 3.88 while preserving similar generation quality at lower effective precision. In contrast, the very low effective precision of CacheQuant-style is accompanied by a substantial quality degradation. These results show that PyraQuant shifts the quality–compression tradeoff toward lower effective precision rather than reducing bit-width at the expense of generation quality.

The amount of additional compression differs across the two pipelines, but the same allocation principle remains effective. FLUX permits a more aggressive reduction because a smaller fraction of the high-resolution computation remains active under the final allocation. SDXL retains more high-resolution computation and therefore reaches a higher effective bit-width. Its two PyraQuant configurations also expose the expected tradeoff between model storage and generation quality. These results suggest that PyraQuant does not prescribe one fixed precision recipe. Instead, it adapts the use of the available numerical configurations to the structure of the progressive pipeline.

Table 2: Effect of applying PyraQuant to different base quantization settings. BOPs is the ratio to the uniform base, Δ values are paired changes after adding PyraQuant. Details in Table 5.

Pipeline	Base	BOPs	ΔCLIP	ΔHPSv2	ΔIR
FLUX	RTN-W4	$0.65\times$	$+0.10$	$+0.0022$	-0.006
FLUX	SVDQuant	$0.54\times$	$+0.00$	$+0.0027$	$+0.010$
SDXL	RTN-W8	$0.72\times$	-0.15	-0.0021	$+0.029$
SDXL	RTN-W4	$0.78\times$	-0.09	-0.0035	$+0.015$

Table 3: Practical deployment results on one NVIDIA L40S GPU. Latency is measured for the PyraQuant executor. Memory reports peak GPU memory during the final 4K stage.

Pipeline	Method	Latency (s)	Memory (GiB)
FLUX	bf16	44.2	35.8
	PyraQuant	24.4	17.3
SDXL	fp16	79.3	11.9
	PyraQuant	59.5	8.9

The main experiments use SVDQuant on FLUX and RTN-W8 on SDXL. To test whether PyraQuant depends on a base configuration, we further apply the same allocation strategy to alternative base quantization settings. Table 2 reports the BOPs ratio and paired quality changes after adding PyraQuant to each uniform base. Across all settings, the changes in CLIP and HPSv2 remain small, at most 0.15 and 0.0035, respectively. The base quantization setting mainly determines the generation quality, while PyraQuant reduces executed BOPs by 22–46%. These results show that PyraQuant generalizes across base quantization settings.

4.3 Analysis of Effective Precision and Spatial Reuse

The main results show that PyraQuant reduces effective precision and backbone computation beyond the underlying quantizers. We analyze two sources of this reduction: lower-precision computation for active regions and cached reuse for regions with limited refinement.

The first source of compression comes from the difference between stored and effective precision. Figure 5(a) compares these quantities for the base quantizers and PyraQuant configurations. On FLUX, SVDQuant stores the model at 4.48 bits per parameter and uses the same precision throughout generation. PyraQuant keeps the same storage requirement but reduces the effective bit-width to 2.00. The same trend appears on SDXL, where the RTN-W8 base remains at 8.45 stored bits while the effective bit-width decreases to 5.33. The lower-storage RTN-W4 configuration further reaches 3.88 effective bits with 4.97-bit storage. Therefore, the reduction in effective precision comes from the allocation strategy rather than from storing additional quantized models.

Spatial reuse provides the second source of compression. At the final 4K stage, PyraQuant only recomputes regions that remain on the active refinement path and reuses cached predictions elsewhere. Figure 5(b) shows that only 30% of the 4K regions are recomputed on FLUX, while 62% remain active on SDXL. FLUX removes a larger fraction of high-resolution computation, while SDXL retains more active regions due to its execution granularity and refinement process.

The two sources of compression determine the computation cost. As reported in Table 1, PyraQuant reduces executed BOPs from 25.3% to 13.6% on FLUX compared with SVDQuant, and from 33.8% to 24.3% on SDXL compared with RTN-W8. The lower-storage RTN-W4 configuration reduces the executed cost to 20.0%. On FLUX, most of the BOP reduction comes from cached reuse, while the lower-precision configuration refines the precision of regions that remain active. Cached regions also let FLUX fall below 19.1%, the share of 16-bit attention BOPs that bounds any full-canvas method.

Together, these results show that the base quantizer determines model representation, while PyraQuant controls its use across resolution stages and spatial regions. The compression comes from both lower precision in active regions and reduced computation over the high-resolution canvas.

4.4 Inference Latency and Memory Usage

The previous analysis shows that PyraQuant reduces effective precision and executed computation under the progressive generation schedule. We next evaluate whether these allocation decisions translate into practical deployment benefits. We measure end-to-end latency using the final PyraQuant executor on a single NVIDIA L40S GPU with model weights resident in memory. Peak GPU memory is measured during the final 4K stage.

As shown in Table 3, PyraQuant reduces both runtime and memory usage on the two progressive pipelines. On ScaleDiff-FLUX.1-schnell, the latency decreases from 44.2 seconds to 24.4 seconds, achieving a $1.81\times$ speedup, while the peak GPU memory decreases from 35.8 GiB to 17.3 GiB. On

Table 4: Ablation studies on the first 500 UltraHR prompts. BOPs are executed BOPs (%) relative to the corresponding 16-bit pipeline. (a) and (c) use FLUX with the SVDQuant base, (b) uses SDXL with the RTN-W8 base. In (a), w/o routing runs the backbone on the full canvas with the allocation.

(a) Removing components

Variant	BOPs	HPSv2	IR	CLIP
PyraQuant	13.6	0.2827	0.620	26.90
w/o cache	24.4	0.2811	0.622	26.87
w/o low bit	13.8	0.2822	0.617	26.85
w/o routing	24.4	0.2827	0.622	26.91

(b) Region selection rule

Selection	BOPs	HPSv2	IR	CLIP
Residual	22.9	0.2890	0.750	27.82
Random	23.0	0.2886	0.745	27.75
Inverted	22.7	0.2885	0.734	27.71

(c) Inheriting 2K decisions

Variant	BOPs	HPSv2	IR	CLIP
PyraQuant	13.6	0.2827	0.620	26.90
No inheritance	21.2	0.2824	0.632	26.92

ScaleDiff-SDXL, PyraQuant reduces the latency from 79.3 seconds to 59.5 seconds and lowers the peak GPU memory from 11.9 GiB to 8.9 GiB, achieving a $1.33\times$ speedup. The latency executor uses INT8 kernels for quantized execution, while the additional speedup comes mainly from routed execution and cached reuse rather than specialized W4/W3 kernels.

4.5 Ablation and Analysis

All variants use the same base quantizer as the full recipe, and the selection variants share a fixed median quota so that they use the same budget. Table 4a studies the effects of low-bit computation, cached reuse, and routing. On FLUX, cached reuse provides the main reduction in executed BOPs: removing cache increases the cost from 13.6% to 24.4%, while removing the low-precision configuration increases it only to 13.8%. This difference is expected because 16-bit attention dominates the per-token BOPs of the W4A4 base. On SDXL, removing the low-precision configuration has a larger effect, increasing BOPs from 24.3% to 26.5%. In both cases, quality changes remain small. Using the same allocation map without routing preserves generation quality but still evaluates the full canvas, increasing executed BOPs to 24.4%. The allocation map itself allocates 12.3% BOPs, while routed execution realizes this reduction with 1.3 percentage points of routing overhead.

We further evaluate residual-guided spatial assignment. Table 4b compares residual-based selection with random and inverted selection under the same budget. Replacing residual guidance with random selection reduces HPSv2 from 0.2890 to 0.2886, ImageReward from 0.750 to 0.745, and CLIP from 27.82 to 27.75. Inverting the residual signal further reduces these metrics to 0.2885, 0.734, and 27.71, respectively, while the executed BOPs remain nearly unchanged. These results show that residual magnitude provides a meaningful signal for spatial allocation rather than merely imposing a fixed computation budget.

Finally, we study cascaded inheritance. Table 4c removes inheritance on FLUX and re-selects all 4K regions independently. Executed BOPs increase from 13.6% to 21.2%, while HPSv2 and CLIP remain nearly unchanged. This shows that inheriting previous spatial decisions avoids unnecessary high-resolution computation.

5 Conclusion

PyraQuant shows that progressive high-resolution diffusion exposes two forms of non-uniformity that existing quantization methods do not exploit: quantization tolerance differs across resolution stages, and the changes introduced by high-resolution refinement are concentrated in a limited set of spatial regions. Exploiting both, PyraQuant performs stage- and region-aware computation allocation on top of a fixed base quantizer, guided by residual magnitude and carried across successive resolutions, with cached reuse for regions requiring limited refinement. On ScaleDiff-FLUX.1-schnell, it lowers the effective precision of SVDQuant from 4.48 to 2.00 bits and executed BOPs from 25.3% to 13.6%. On ScaleDiff-SDXL, executed BOPs drop from 33.8% to 24.3%, with preserved generation quality on both backbones. These savings carry over to deployment: on a single NVIDIA L40S, the PyraQuant executor achieves $1.81\times$ and $1.33\times$ end-to-end speedups over the 16-bit pipelines and reduces final-stage peak GPU memory on FLUX from 35.8 GiB to 17.3 GiB. Because the allocation operates on top of the base quantizer, these gains are complementary to future advances in quantizers themselves and are expected to grow as generation moves to higher resolutions.

Code Availability. The anonymous code repository of PyraQuant is available at: `https://anonymous.4open.science/r/PyraQuant-FC54/README.md`.

AI USE STATEMENT

Large language model assistants were used to support literature discovery and to improve the clarity and presentation of the manuscript. All research ideas, method design, experiments, analysis, and conclusions were developed and verified by the authors.

REFERENCES

Mikołaj Bińkowski, Danica J. Sutherland, Michael Arbel, and Arthur Gretton. Demystifying MMD GANs. In *International Conference on Learning Representations*, 2018.

Black Forest Labs. FLUX. https://github.com/black-forest-labs/flux, 2024.

Jiazi Bu, Pengyang Ling, Yujie Zhou, Pan Zhang, Tong Wu, Xiaoyi Dong, Yuhang Zang, Yuhang Cao, Dahua Lin, and Jiaqi Wang. HiFlow: Training-free high-resolution image generation with flow-aligned guidance. In *Advances in Neural Information Processing Systems*, volume 38, pp. 144317–144351, 2025. doi: 10.52202/085713-4831.

Lucy Chai, Michaël Gharbi, Eli Shechtman, Phillip Isola, and Richard Zhang. Any-resolution training for high-resolution image synthesis. In *European Conference on Computer Vision*, pp. 170–188, 2022.

Ruoyi Du, Dongliang Chang, Timothy Hospedales, Yi-Zhe Song, and Zhanyu Ma. DemoFusion: Democratising high-resolution image generation with no $$$. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6159–6168, 2024.

Weilun Feng, Haotong Qin, Chuanguang Yang, Zhulin An, Libo Huang, Boyu Diao, Fei Wang, Renshuai Tao, Yongjun Xu, and Michele Magno. MPQ-DM: Mixed precision quantization for extremely low bit diffusion models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2025.

Elias Frantar, Saleh Ashkboos, Torsten Hoefler, and Dan Alistarh. OPTQ: Accurate quantization for generative pre-trained transformers. In *International Conference on Learning Representations*, 2023.

Yefei He, Luping Liu, Jing Liu, Weijia Wu, Hong Zhou, and Bohan Zhuang. PTQD: Accurate post-training quantization for diffusion models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 13237–13249, 2023. doi: 10.52202/075280-0580. URL https://papers.nips.cc/paper_files/paper/2023/hash/2aab8a76c7e761b66eccaca0927787de-Abstract-Conference.html.

Jack Hessel, Ari Holtzman, Maxwell Forbes, Ronan Le Bras, and Yejin Choi. CLIPScore: A reference-free evaluation metric for image captioning. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, 2021.

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 *Advances in Neural Information Processing Systems*, volume 30, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/8a1d694707eb0fefe65871369074926d-Abstract.html.

Cheeun Hong, Sungyong Baik, Heewon Kim, Seungjun Nah, and Kyoung Mu Lee. CADyQ: Content-aware dynamic quantization for image super-resolution. In *European Conference on Computer Vision*, pp. 367–383, 2022.

Yushi Huang, Ruihao Gong, Jing Liu, Tianlong Chen, and Xianglong Liu. TFMQ-DM: Temporal feature maintenance quantization for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7362–7371, 2024. doi: 10.1109/CVPR52733.2024.00703. URL https://openaccess.thecvf.com/content/CVPR2024/html/Huang_TFMQ-DM_Temporal_Feature_Maintenance_Quantization_for_Diffusion_Models_CVPR_2024_paper.html.

Qing Jin, Linjie Yang, and Zhenyu Liao. AdaBits: Neural network quantization with adaptive bit-widths. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020.

Seeyeon Kim, Jaehun Lee, Sungyeob Yoo, and Joo-Young Kim. MASQ: Accelerating masked diffusion via stage-wise multi-precision quantization, 2026. URL https://arxiv.org/abs/2605.23226.

Sungho Koh, SeungJu Cha, Hyunwoo Oh, Kwanyoung Lee, and Dong-Jin Kim. ScaleDiff: Higher-resolution image synthesis via efficient and model-agnostic diffusion. In *Advances in Neural Information Processing Systems*, volume 38, pp. 95571–95596, 2025. doi: 10.52202/085713-3196.

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 component for 4-bit diffusion models. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vWR3KuiQur.

Xiuyu Li, Yijiang Liu, Long Lian, Huanrui Yang, Zhen Dong, Daniel Kang, Shanghang Zhang, and Kurt Keutzer. Q-Diffusion: Quantizing diffusion models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 17535–17545, 2023.

Bowen Lin, Fanjiang Ye, Yihua Liu, Zhenghui Guo, Boyuan Zhang, Weijian Zheng, Yufan Xu, Tiancheng Xing, Yuke Wang, and Chengming Zhang. SDiT: Semantic region-adaptive for diffusion transformers, 2026. URL https://arxiv.org/abs/2601.12283.

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 *European Conference on Computer Vision*, volume 8693 of *Lecture Notes in Computer Science*, pp. 740–755. Springer, 2014. doi: 10.1007/978-3-319-10602-1_48.

Yaron Lipman, Ricky T. Q. Chen, Heli Ben-Hamu, Maximilian Nickel, and Matt Le. Flow matching for generative modeling. In *International Conference on Learning Representations*, 2023.

Xingchao Liu, Chengyue Gong, and Qiang Liu. Flow straight and fast: Learning to generate and transfer data with rectified flow. In *International Conference on Learning Representations*, 2023.

Xuewen Liu, Zhikai Li, and Qingyi Gu. CacheQuant: Comprehensively accelerated diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 23269–23280, 2025.

Ziming Liu, Yifan Yang, Chengruidong Zhang, Yiqi Zhang, Lili Qiu, Yang You, and Yuqing Yang. Region-adaptive sampling for diffusion transformers. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2346–2356, 2026.

Xudong Lu, Aojun Zhou, Ziyi Lin, Qi Liu, Yuhui Xu, Renrui Zhang, Xue Yang, Junchi Yan, Peng Gao, and Hongsheng Li. TerDiT: Ternary diffusion models with transformers. *arXiv preprint arXiv:2405.14854*, 2024.

Xinyin Ma, Gongfan Fang, and Xinchao Wang. DeepCache: Accelerating diffusion models for free. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 15762–15772, 2024.

Keith G. Mills, Mohammad Salameh, Ruichen Chen, Negar Hassanpour, Wei Lu, and Di Niu. Qua2SeDiMo: Quantifiable quantization sensitivity of diffusion models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 6153–6163, 2025. doi: 10.1609/aaai.v39i6.32658.

Pranav Nair, Puranjay Datta, Jeff Dean, Prateek Jain, and Aditya Kusupati. Matryoshka quantization. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, 2025.

Yeonhong Park, Jake Hyun, SangLyul Cho, Bonggeun Sim, and Jae W. Lee. Any-precision LLM: Low-cost deployment of multiple, different-sized LLMs. In *Proceedings of the 41st International Conference on Machine Learning*, 2024.

William Peebles and Saining Xie. Scalable diffusion models with transformers. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 4195–4205, 2023.

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. In *International Conference on Learning Representations*, 2024. URL https://proceedings.iclr.cc/paper_files/paper/2024/hash/081b08068e4733ae3e7ad019fe8d172f-Abstract-Conference.html.

Haonan Qiu, Shiwei Zhang, Yujie Wei, Ruihang Chu, Hangjie Yuan, Xiang Wang, Yingya Zhang, and Ziwei Liu. FreeScale: Unleashing the resolution of diffusion models via tuning-free scale fusion. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 16893–16903, 2025.

Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10684–10695, 2022.

Hyogon Ryu, NaHyeon Park, and Hyunjung Shim. DGQ: Distribution-aware group quantization for text-to-image diffusion models. In *International Conference on Learning Representations*, 2025.

Tim Salimans, Ian Goodfellow, Wojciech Zaremba, Vicki Cheung, Alec Radford, and Xi Chen. Improved techniques for training GANs. In *Advances in Neural Information Processing Systems*, 2016.

Yuzhang Shang, Zhihang Yuan, Bin Xie, Bingzhe Wu, and Yan Yan. Post-training quantization on diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1972–1981, 2023.

Yang Sui, Yanyu Li, Anil Kag, Yerlan Idelbayev, Junli Cao, Ju Hu, Dhritiman Sagar, Bo Yuan, Sergey Tulyakov, and Jian Ren. BitsFusion: 1.99 bits weight quantization of diffusion model. In *Advances in Neural Information Processing Systems*, 2024.

Weitian Wang, Shubham Rai, Cecilia De la Parra, and Akash Kumar. MixA-Q: Revisiting activation sparsity for vision transformers from a mixed-precision quantization perspective. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 22143–22152, 2025.

Zhou Wang, Alan C. Bovik, Hamid R. Sheikh, and Eero P. Simoncelli. Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4): 600–612, 2004.

Haoning Wu, Shaocheng Shen, Qiang Hu, Xiaoyun Zhang, Ya Zhang, and Yanfeng Wang. MegaFusion: Extend diffusion models towards higher-resolution image generation without further tuning. In *Proceedings of the Winter Conference on Applications of Computer Vision*, pp. 3944–3954, 2025a.

Junyi Wu, Haoxuan Wang, Yuzhang Shang, Mubarak Shah, and Yan Yan. PTQ4DiT: Post-training quantization for diffusion transformers. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/72d32f4fe0b7af03732bd227bf1c4a5f-Abstract-Conference.html.

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 *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 15035–15044, 2025b.

Xiaoshi Wu, Yiming Hao, Keqiang Sun, Yixiong Chen, Feng Zhu, Rui Zhao, and Hongsheng Li. Human preference score v2: A solid benchmark for evaluating human preferences of text-to-image synthesis, 2023. URL https://arxiv.org/abs/2306.09341.

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 *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 38087–38099, 2023.

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 *Advances in Neural Information Processing Systems*, 2023.

Lianwei Yang, Haokun Lin, Tianchen Zhao, Yichen Wu, Hongyu Zhu, Ruiqi Xie, Zhenan Sun, Yu Wang, and Qingyi Gu. LRQ-DiT: Log-rotation post-training quantization of diffusion transformers for image and video generation, 2025. URL https://arxiv.org/abs/2508.03485.

Yuzhe Yao, Feng Tian, Jun Chen, Haonan Lin, Guang Dai, Yong Liu, and Jingdong Wang. Timestep-aware correction for quantized diffusion models. In *Computer Vision – ECCV 2024*, pp. 215–232, 2024. doi: 10.1007/978-3-031-72848-8_13. URL https://www.ecva.net/papers/eccv_2024/papers_ECCV/papers/08312.pdf.

Haichao Yu, Haoxiang Li, Humphrey Shi, Thomas S. Huang, and Gang Hua. Any-precision deep neural networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, 2021.

Jiaji Zhang, Ruichao Sun, Hailiang Zhao, Jiaju Wu, Peng Chen, Hao Li, Yuying Liu, Kingsum Chow, Gang Xiong, and Shuiguang Deng. SegQuant: A semantics-aware and generalizable quantization framework for diffusion models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 43717–43726, 2026.

Richard Zhang, Phillip Isola, Alexei A. Efros, Eli Shechtman, and Oliver Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 586–595, 2018.

Chen Zhao, En Ci, Yunzhe Xu, Tiehan Fan, Shanyan Guan, Yanhao Ge, Jian Yang, and Ying Tai. UltraHR-100K: Enhancing UHR image synthesis with a large-scale high-quality dataset. In *Advances in Neural Information Processing Systems*, 2025a.

Tianchen Zhao, Xuefei Ning, Tongcheng Fang, Enshu Liu, Guyue Huang, Zinan Lin, Shengen Yan, Guohao Dai, and Yu Wang. MixDQ: Memory-efficient few-step text-to-image diffusion models with metric-decoupled mixed precision quantization. In *Computer Vision – ECCV 2024*, pp. 285–302, 2024. doi: 10.1007/978-3-031-72630-9_17.

Tianchen Zhao, Tongcheng Fang, Haofeng Huang, Rui Wan, Widyadewi Soedarmadji, Enshu Liu, Shiyao Li, Zinan Lin, Guohao Dai, Shengen Yan, Huazhong Yang, Xuefei Ning, and Yu Wang. ViDiT-Q: Efficient and accurate quantization of diffusion transformers for image and video generation. In *International Conference on Learning Representations*, 2025b. URL https://openreview.net/forum?id=E1N1oxd63b.

Xingyu Zheng, Xianglong Liu, Haotong Qin, Xudong Ma, Mingyuan Zhang, Haojie Hao, Jiakai Wang, Zixiang Zhao, Jinyang Guo, and Michele Magno. BinaryDM: Accurate weight binarization for efficient diffusion models. In *International Conference on Learning Representations*, 2025.

Chang Zou, Xuyang Liu, Ting Liu, Siteng Huang, and Linfeng Zhang. Accelerating diffusion transformers with token-wise feature caching. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/hash/bbe024e0517fe12ac3a8d388b19ff9fe-Abstract-Conference.html.

A Implementation Details

This section provides additional implementation details of PyraQuant, including the construction of nested precision states, the routing executor, and the computation accounting.

A.1 Quantization Details

PyraQuant extends an existing quantized representation with nested lower-precision configurations for stage- and region-aware allocation.

For the DiT pipeline, we use the released SVDQuant W4A4 model as the base quantizer. The high-precision configuration directly follows the original quantized weights:

$$Q_{\mathrm{hi}} = Q. \tag{9}$$

The lower-precision configuration is obtained by merging nearby quantization codes, following the nesting of integer codes used by any-precision quantization (Park et al., 2024; Nair et al., 2025):

$$Q_{\mathrm{lo}} = \mathcal{N}(Q_{\mathrm{hi}}). \tag{10}$$

The nested mapping operates directly on stored quantized codes and keeps the original scaling factors unchanged. Therefore, the lower-precision configuration does not require an additional weight copy or calibration procedure.

For the UNet pipeline, the base quantizer uses RTN W8A8 with group size 32 for linear layers and W8 convolution weights. The nested lower-precision configuration is obtained from the stored W8 codes using a smaller codebook while preserving the original quantization scale. A low-storage configuration further uses RTN W4A8 as the base quantizer and derives a nested W3 configuration.

PyraQuant preserves the model-weight storage of the base quantizer: its numerical configurations share one stored representation and do not require separate model copies.

A.2 Baseline Configurations

All baselines use the same ScaleDiff pipelines, progressive schedules, prompts, seeds, and output resolution as PyraQuant. When an official implementation or released quantized model is available, we use it directly or adapt it to the corresponding ScaleDiff backbone. Otherwise, we reproduce the reported quantization or reuse strategy within the same host pipeline. These adapted variants are marked as "-style" in Table 1. We keep the weight and activation precisions listed in Table 1 and do not modify the denoising schedule. This setup isolates the quantization or reuse strategy from differences in the underlying generation pipeline. SegQuant-G denotes the GPTQ-calibrated (Frantar et al., 2023) variant of SegQuant, following the naming of the original paper (Zhang et al., 2026).

A.3 Routing and Cache Executor

The progressive pipeline uses 64×64 latent regions as the basic allocation unit. This corresponds to a 4×4 grid at 2K resolution and an 8×8 grid at 4K resolution.

At the 2K stage, PyraQuant assigns each region to either the high-precision or low-precision configuration using the residual-guided selection rule. The resulting high-precision regions define the candidate regions for the next resolution.

At 4K, PyraQuant applies cascaded allocation. Only children of inherited high-precision regions remain candidates for further computation. Regions outside this candidate set directly use cached reuse.

For the main DiT configuration, computed regions are merged into local crops and expanded with a 2-token halo. The local outputs are fused back into the full canvas.

For the UNet pipeline, the executable unit is constrained by the minimum crop granularity of the backbone. Each executable region contains the four children of one 2K region. Therefore, at 4K

the UNet pipeline keeps all children of inherited high-precision regions active, while children of low-precision regions use cached reuse.

The cache prediction is mapped into the current diffusion output space through the scheduler-dependent transformation:

$$y_t^{\text{cache}} = \Pi_t(z_t, \hat{x}_0^{\text{cache}}). \tag{11}$$

The final prediction combines computed and cached regions:

$$\hat{y}_t = (1 - m_C) \odot y_t^{\text{model}} + m_C \odot y_t^{\text{cache}}. \tag{12}$$

A.4 Accounting and Simulation

We report three related quantities: storage bit, effective bit, and executed bit operations (BOPs).

Storage bit measures the average number of bits required to store the model parameters, including quantization scales and additional low-rank factors when applicable.

Effective bit measures the computation-weighted precision assigned during progressive generation:

$$B_{\text{eff}} = \frac{\sum_i n_i b_i}{\sum_i n_i}, \tag{13}$$

where n_i denotes the number of tokens assigned to state i and b_i denotes the bit cost of that state, including quantization scales and low-rank factors when applicable. Cached regions use $b_C = 0$ and are included in the denominator.

For full-canvas quantization methods without cached reuse, effective bit is identical to storage bit. For PyraQuant, the difference reflects the allocation of lower-precision configurations and cached reuse across the progressive schedule. The numerical backend used for runtime measurement is described in Appendix E.3.

We distinguish allocated BOPs from executed BOPs. Allocated BOPs are determined by the computation map before routing. Executed BOPs count the computation actually evaluated on routed regions, including routing overhead such as halo regions and the text tokens processed with every routed crop. Cached regions contribute zero backbone BOPs. Attention remains at 16-bit precision, and auxiliary low-rank branches are included at their executed precision.

Quality evaluation uses quantize–dequantize (QDQ) simulation. The runtime-measurement path is described in Appendix E.3.

B Additional Experimental Results

This section provides additional experimental results, including plug-in generality, threshold selection, and hyper-parameter analysis.

B.1 Plug-in Generality

PyraQuant is designed as an allocation framework on top of an existing quantizer. We therefore evaluate whether the same allocation strategy remains effective with different base quantizers.

Table 5 compares uniform quantization with the corresponding PyraQuant configuration under four base quantization settings. On the DiT pipeline, we evaluate RTN W4A8 and SVDQuant W4A4. On the UNet pipeline, we evaluate RTN W8A8 and RTN W4A8.

Across the four tested base quantization settings, PyraQuant reduces executed BOPs by 22–46% relative to the corresponding uniform bases, with absolute HPSv2 changes of at most 0.0035. HPSv2 improves on both FLUX settings and ImageReward improves on three of the four settings. These paired comparisons support applying the allocation framework across different base quantization settings on both DiT and UNet pipelines.

Table 5: Plug-in generality across different base quantization settings. BOPs Ratio is the executed BOPs of each PyraQuant configuration divided by those of its corresponding uniform base, under the accounting of Table 1. The FLUX RTN W4A8 rows use the first 200 prompts and an 8-token routing halo (larger than the 2-token halo of the main configuration, so their BOPs ratio is conservative), and all other rows use the 2,000 prompts. The Δ rows report paired per-prompt changes after adding PyraQuant to the corresponding base configuration, computed from unrounded scores.

Pipeline	Configuration	BOPs Ratio	HPSv2	IR	CLIP
FLUX	RTN W4A8	1.00×	0.2769	0.681	26.74
	RTN W4A8 + PyraQuant	0.65×	0.2790	0.675	26.84
	Δ		+0.0022	-0.006	+0.10
FLUX	SVDQuant W4A4	1.00×	0.2806	0.648	27.10
	SVDQuant + PyraQuant	0.54×	0.2833	0.658	27.11
	Δ		+0.0027	+0.010	+0.00
SDXL	RTN W8A8	1.00×	0.2925	0.739	28.22
	RTN W8A8 + PyraQuant	0.72×	0.2904	0.768	28.06
	Δ		-0.0021	+0.029	-0.15
SDXL	RTN W4A8	1.00×	0.2900	0.716	28.11
	RTN W4A8 + PyraQuant	0.78×	0.2865	0.731	28.02
	Δ		-0.0035	+0.015	-0.09

Table 6: Threshold sweep results on the 32 calibration prompts. Allocated BOPs follow the accounting of Table 1. 4K Computed Regions is the fraction of the 8×8 4K regions that are recomputed rather than cached, averaged over the calibration prompts. The selected threshold is $\tau = 0.10$ for both pipelines. For SDXL, τ governs the 2K sibling test in every row, the rows from 0.03 to 0.15 additionally apply the sibling test at 4K, whereas the final recipe keeps all inherited children active at 4K to match the executable unit (Section 3.3) and is reported in the last row at the same threshold.

Pipeline	τ	Allocated BOPs	4K Computed Regions	Notes
FLUX	0.05	9.4%	14%	
FLUX	0.10	11.3%	24%	Selected
FLUX	0.20	15.7%	49%	
SDXL	0.03	11.5%	12%	
SDXL	0.05	12.8%	18%	
SDXL	0.10	16.6%	35%	
SDXL	0.15	20.0%	50%	
SDXL	0.10	20.7%	54%	Selected (final recipe)

B.2 Threshold Analysis

The allocation rule of PyraQuant exposes a single threshold, the relative-gap τ, and the leaf size and the halo are implementation parameters and are analyzed in Appendix B.3. We select τ on a calibration set of 32 COCO captions (Lin et al., 2014) that are disjoint from the evaluation prompts.

Table 6 reports the threshold sweep results. The threshold controls how aggressively regions are separated into different precision states.

Increasing τ retains more computed regions at 4K and increases allocated BOPs on both pipelines. For SDXL, the final configuration retains all inherited children to match the backbone's executable sibling-group granularity, resulting in 54% computed regions and 20.7% allocated BOPs at $\tau = 0.10$ on the calibration prompts. The corresponding fractions over the 2,000 evaluation prompts are shown in Fig. 5.

We select $\tau = 0.10$ from this sweep as the operating point for both main configurations and keep it fixed in all evaluations. Alternative thresholds are identified explicitly in the corresponding experiments.

Table 7: Effect of spatial leaf size on FLUX (SVDQuant base), first 100 UltraHR prompts; executed BOPs. The default setting uses 64 latent pixels; the same configuration over all 2,000 prompts executes 13.6% (Table 1).

Leaf Size	BOPs	HPSv2	IR
32	19.1%	0.2834	0.685
64	13.5%	0.2835	0.672
128	14.4%	0.2838	0.675

Table 8: Effect of halo size on routed execution. BOPs use the configuration maps of the main FLUX run on the 2,000 prompts with the same accounting as Table 1. Latency is the median over 20 prompts on one L40S. The seam ratio is the mean absolute image gradient across crop boundaries divided by the mean absolute gradient inside crops, measured on the 4K outputs of 8 diagnostic images.

Halo	Executed BOPs	Seam Ratio	Latency
0	12.7%	1.083	23.6 s
2	13.6%	0.996	24.4 s
4	14.5%	0.997	25.8 s
8	16.4%	0.998	27.6 s

B.3 Leaf Size and Halo Analysis

We further evaluate two implementation parameters: the spatial allocation granularity and the halo size used by the routed executor.

B.3.1 Leaf Size

Table 7 evaluates different leaf sizes.

The default 64-latent-pixel leaf achieves the lowest executed BOPs among the tested sizes (13.5%), while HPSv2 varies by only 0.0004 across the three settings. This configuration balances spatial allocation flexibility and routing overhead.

B.3.2 Halo Size

Table 8 evaluates halo sizes for routed execution.

The 2-token halo is the smallest setting whose boundary statistics match those of the larger halos (seam ratio 0.996 versus 0.997 and 0.998 for 4 and 8 tokens), at an additional 0.9 percentage points of executed BOPs and 0.8 s of latency over no halo. Larger halos increase executed BOPs and latency without further change in the seam ratio.

C Additional Ablation Studies

This section provides additional ablation studies on the main components of PyraQuant. Unless otherwise stated, variants use the same base quantizer as the full recipe and are evaluated on the first 500 UltraHR prompts. The targeted diagnostic analyses in Tables 10 and 11 use 16 FLUX images.

C.1 Computation State Ablation

PyraQuant combines lower-precision computation, cached reuse, and spatial routing. We evaluate the contribution of each component.

Removing cached reuse raises executed BOPs from 13.6% to 24.4% on FLUX and from 24.3% to 29.4% on SDXL, while the absolute HPSv2 change is 0.0016 on each pipeline. Removing the low-precision state raises executed BOPs to 13.8% and 26.5%, respectively, with small changes in the reported quality metrics. The two states therefore provide complementary computation savings.

Table 9: Ablation of computation ablation on the first 500 UltraHR prompts with the main base quantizers. The w/o routing rows send the full canvas to the backbone with the same configuration map (cached leaves evaluated at the low-precision configuration) while still applying the cache blend of Eq. 8; they therefore execute the same BOPs as w/o cache. Their allocated BOPs are 12.3% (FLUX) and 22.4% (SDXL).

Variant	Executed BOPs	HPSv2	IR	CLIP
FLUX (SVDQuant W4A4)				
Full recipe	13.6%	0.2827	0.620	26.90
w/o cache	24.4%	0.2811	0.622	26.87
w/o low bit	13.8%	0.2822	0.617	26.85
w/o routing	24.4%	0.2827	0.622	26.91
SDXL (RTN-W8A8)				
Full recipe	24.3%	0.2892	0.756	27.87
w/o cache	29.4%	0.2908	0.725	28.04
w/o low bit	26.5%	0.2897	0.751	27.91
w/o routing	29.4%	0.2892	0.747	27.90

Table 10: Comparison of lower computation states on 16 diagnostic FLUX images. The 1K and 2K stages are identical, and the entire 4K stage is run in the listed state. PSNR, SSIM (Wang et al., 2004), and LPIPS (Zhang et al., 2018) are computed on the 4K outputs, downsampled to 1024^2, against the output obtained with the 4K stage run in bf16; the 4K backbone column gives the fraction of 4K backbone computation executed.

4K state	4K backbone	PSNR ↑	SSIM ↑	LPIPS ↓
W3 recompute	100%	**30.5**	**0.886**	**0.170**
Cache	0	28.0	0.792	0.198
W2 recompute	100%	24.8	0.694	0.459

With the same allocation map, routed execution reduces executed BOPs from 24.4% to 13.6% on FLUX and from 29.4% to 24.3% on SDXL relative to the full-canvas executor, with unchanged HPSv2 in both comparisons.

C.2 Lower-Precision State Analysis

Table 10 supports the choice of W3 and cached reuse as complementary computation states on FLUX. W3 recomputation provides the highest reconstruction fidelity among the tested states, while cached reuse outperforms W2 recomputation in PSNR, SSIM, and LPIPS on all 16 images, improving PSNR from 24.8 to 28.0 and reducing LPIPS from 0.459 to 0.198. At the region level, cached reuse also gives a lower reconstruction error than W2 recomputation on 96.4% of the $16 \times 8 \times 8 = 1{,}024$ 4K regions. Because attention remains at 16-bit precision, lowering the numerical configuration from W3 to W2 would reduce per-token BOPs only from 95% to 90% of the W4 state, whereas cached regions bypass backbone execution entirely. Cached reuse therefore dominates W2 recomputation in both fidelity and cost, which is why the state hierarchy stops at W3 and uses cached reuse as the terminal state.

C.3 Residual-Guided Selection Ablation

We first examine whether residual magnitude reflects the need for further refinement. At the 4K stage, we replace selected regions with the inherited prediction while keeping the remaining regions unchanged, and measure the resulting reconstruction error against the fully refined output. Across 16 diagnostic images, the within-image Spearman correlation between $e(B)$ and the error caused by skipping the corresponding region is 0.71 on average, with positive correlation on all 16 images.

Table 11 further compares three choices that skip the same 25% of regions. Skipping regions with the lowest residual magnitude produces substantially smaller error than either random selection or

Table 11: Region-level validation of residual magnitude on 16 diagnostic FLUX images. At the 4K stage, 25% of the 64 regions of each image are replaced by the inherited prediction. Block MSE is the mean squared error (8-bit RGB scale) of the replaced regions against the fully refined bf16 output; PSNR is measured on the full output against the same reference, with the remaining regions computed either in bf16 or in the W3 state.

Skipped regions	Block MSE ↓	PSNR, others bf16 ↑	PSNR, others W3 ↑
Lowest $e(B)$ 25%	**79**	**37.2**	**28.8**
Random 25%	187	32.3	28.2
Highest $e(B)$ 25%	379	29.1	27.2

Table 12: Ablation of the spatial selection signal on the first 500 UltraHR prompts. All variants select the same number of regions per image (a fixed median quota) in place of the relative-gap rule, so their allocated BOPs are identical; the Residual rows are therefore quota-matched variants rather than the full recipe of Table 9. LPIPS_q (Zhang et al., 2018) and PSNR_q are computed against the base quantizer's own output for the same prompt and seed.

Selection	Pipeline	HPSv2	IR	CLIP	LPIPS_q ↓	PSNR_q ↑
Residual	FLUX	0.2827	0.621	26.86	0.146	26.0
Random	FLUX	0.2822	0.629	26.97	0.157	24.8
Inverted	FLUX	0.2816	0.638	26.90	0.164	24.0
Residual	SDXL	0.2890	0.750	27.82	0.172	23.6
Random	SDXL	0.2886	0.745	27.75	0.178	22.9
Inverted	SDXL	0.2885	0.734	27.71	0.181	22.4

skipping regions with the highest residual magnitude. The block MSE increases from 79 to 379 between the lowest- and highest-residual groups. These results indicate that residual magnitude captures how strongly a region depends on the current refinement stage.

We then evaluate whether this signal improves computation-state allocation by comparing residual-based selection with random and inverted selection under the same allocation budget.

Residual-based selection achieves the highest HPSv2 and remains closest to the base quantizer's output on both pipelines. Relative to random selection, it improves PSNR_q from 24.8 to 26.0 on FLUX and from 22.9 to 23.6 on SDXL, while reducing LPIPS_q from 0.157 to 0.146 and from 0.178 to 0.172, respectively. ImageReward and CLIPScore also favor residual-based selection on SDXL; on FLUX the three selections lie within 0.017 ImageReward and 0.11 CLIPScore of one another, whereas the reconstruction metrics separate them monotonically. Together with the region-level diagnostic, these results support transition residual magnitude as a signal for allocating computation to regions that need further refinement.

C.4 Cascade Inheritance Ablation

The cascade design avoids rebuilding a complete precision map at every resolution. We compare it with re-selection strategies at the final resolution.

Cascade inheritance carries spatial decisions across resolutions and avoids recomputing regions that can reuse cached predictions. At the same threshold, it lowers executed BOPs from 21.2% to 13.6% on FLUX with comparable quality metrics. On SDXL, where the executable unit is a full sibling group (Appendix A.3), the two variants execute nearly the same computation (24.3% versus 24.8%) with comparable quality, and the cascade avoids the additional selection step; the $\tau = 0.15$ sibling test provides a further operating point at 27.8% executed BOPs.

D Additional Quality Analysis

This section reports additional distribution metrics, discusses the CacheQuant-style row of Table 1, and provides qualitative comparisons.

Table 13: Ablation of cascaded inheritance on the first 500 UltraHR prompts. On FLUX, the no-inheritance variant re-scores all 4K regions with the same τ. On SDXL, the 4K sibling test applies Eq. 5 inside each inherited group at $\tau = 0.10$; the last row applies the same sibling test at $\tau = 0.15$.

Variant	Executed BOPs	HPSv2	IR	CLIP
FLUX (SVDQuant W4A4)				
Cascade	13.6%	0.2827	0.620	26.90
No inheritance	21.2%	0.2824	0.632	26.92
SDXL (RTN-W8A8)				
Cascade	24.3%	0.2892	0.756	27.87
4K sibling test	24.8%	0.2885	0.757	27.87
4K sibling test, $\tau = 0.15$	27.8%	0.2895	0.750	27.88

Table 14: Distribution metrics for 4096^2 outputs on the 2,000 UltraHR-eval4K prompts of Table 1. Subscripts r and b denote the UltraHR reference images and matched 16-bit outputs, respectively, and subscript p denotes patch-level metrics (Chai et al., 2022; Du et al., 2024) computed on five fixed 1024^2 crops without resizing; all KID values are scaled by 10^3.

Method	KID_r	KID_b	IS	$\text{FID}_{p,r}$	$\text{KID}_{p,r}$	IS_p
FLUX						
bf16	12.2	–	12.15	50.4	21.8	9.93
SVDQuant	10.7	-0.1	12.46	39.4	15.1	9.74
ViDiT-Q-style	12.2	0.3	12.00	35.5	12.3	9.55
SegQuant-G	11.1	-0.2	12.39	42.3	16.8	9.76
PyraQuant	7.0	1.0	12.52	23.6	5.7	8.91
SDXL						
fp16	3.8	–	12.40	49.3	23.9	11.98
SmoothQuant	3.7	1.8	11.99	42.9	17.0	11.43
SegQuant-G	3.7	-0.3	12.21	48.7	23.3	12.23
CacheQuant-style	65.3	64.6	7.08	110.0	60.4	5.93
RTN-W8	3.6	-0.2	12.65	46.3	21.2	11.95
PyraQuant on RTN-W8	3.8	0.0	12.40	41.1	18.8	11.70
PyraQuant on RTN-W4	4.0	0.6	12.30	38.8	16.8	10.98

D.1 Additional Distribution Metrics

Table 14 complements the main results with patch-level distribution metrics and reports the corresponding whole-image KID and Inception Score for reference.

The patch-level results provide a complementary view of local image statistics. On FLUX, PyraQuant reduces $\text{FID}_{p,r}$ from 39.4 to 23.6 and $\text{KID}_{p,r}$ from 15.1 to 5.7 relative to SVDQuant. On SDXL, PyraQuant on RTN-W8 reduces $\text{FID}_{p,r}$ from 46.3 to 41.1 and $\text{KID}_{p,r}$ from 21.2 to 18.8 relative to RTN-W8, and matches the fp16 pipeline on whole-image KID_r (3.8) and IS (12.40).

D.2 CacheQuant-style Analysis

The CacheQuant-style baseline reaches the lowest effective bit-width in Table 1, but with a substantial quality decrease on the progressive SDXL pipeline. Following the protocol of Section 4.1, this baseline retains the static per-step activation quantization and the calibration procedure of the original method. Table 15 isolates its components on three diagnostic prompts: temporal reuse, with or without error correction, and W8 weight quantization each stay close to the fp16 reference, whereas adding static activation quantization to temporal reuse reproduces the degradation of the full configuration.

These diagnostics suggest that the static activation quantization used by the adapted CacheQuant-style baseline contributes to its degradation on the progressive SDXL pipeline. PyraQuant retains the base quantizer's activation quantization scheme and applies cached reuse to regions selected by the

Table 15: Component diagnostics of the CacheQuant-style baseline on three diagnostic prompts (mean CLIPScore of the 4096^2 outputs). Each row enables only the listed components of the method on the progressive SDXL pipeline.

Configuration	CLIP
fp16 pipeline	33.4
Temporal reuse only	33.4
Temporal reuse without error correction	33.5
W8 weights + temporal reuse	33.6
Static A8 activations + temporal reuse	23.4
Full CacheQuant-style configuration	21.2

Table 16: Runtime ablations on the DiT pipeline. Rows compare executor configurations; speedups are relative to the bf16 full-canvas baseline.

Executor	Latency (s)	Speedup
bf16 full canvas	44.2	1.00×
INT8 full canvas	41.0	1.08×
INT8 routed, no cache	38.1	1.16×
PyraQuant executor	24.4	1.81×
bf16 routed + cache	26.4	1.67×

transition residual, while active regions use either the base or the nested lower-precision weight configuration.

D.3 Qualitative Comparison

Figures 6 and 7 compare the main configurations of Table 1 on both pipelines.

E Runtime Analysis

This section provides detailed runtime measurements and deployment analysis.

E.1 Runtime Ablations

We evaluate executor configurations that isolate the contributions of INT8 kernels, spatial routing, and cached reuse. Latencies are medians over the same 20 prompts after one warm-up run on a single NVIDIA L40S GPU, with model weights resident in GPU memory and without CPU offloading. Speedups are measured relative to the corresponding 16-bit full-canvas pipeline.

On the DiT pipeline, adding cached reuse to the routed INT8 executor reduces latency from 38.1 s to 24.4 s, a 36.0% reduction. The complete executor achieves a $1.81\times$ speedup over the bf16 full-canvas baseline. The bf16 routed-and-cached configuration also achieves a $1.67\times$ speedup, demonstrating that spatial routing and reuse provide benefits across numerical execution backends.

The same analysis is performed on the UNet pipeline.

On the UNet pipeline, INT8 linear layers reduce full-canvas latency from 79.3 s to 68.9 s. On this backbone, routing is the mechanism that enables cached reuse rather than a source of savings by itself: without cache every region remains active, so routed execution covers the full canvas plus the 8-latent-pixel halo (78.8 s). Cached reuse then reduces the latency from 78.8 s to 59.5 s, a 24.5% reduction, yielding a $1.33\times$ speedup over the fp16 full-canvas baseline. The executor applies INT8 kernels to linear layers, while convolutions remain in fp16 and spatial routing follows the backbone's coarser crop granularity.

E.2 Detailed Memory Analysis

Table 18 reports the peak GPU memory measured during the final 4K stage.

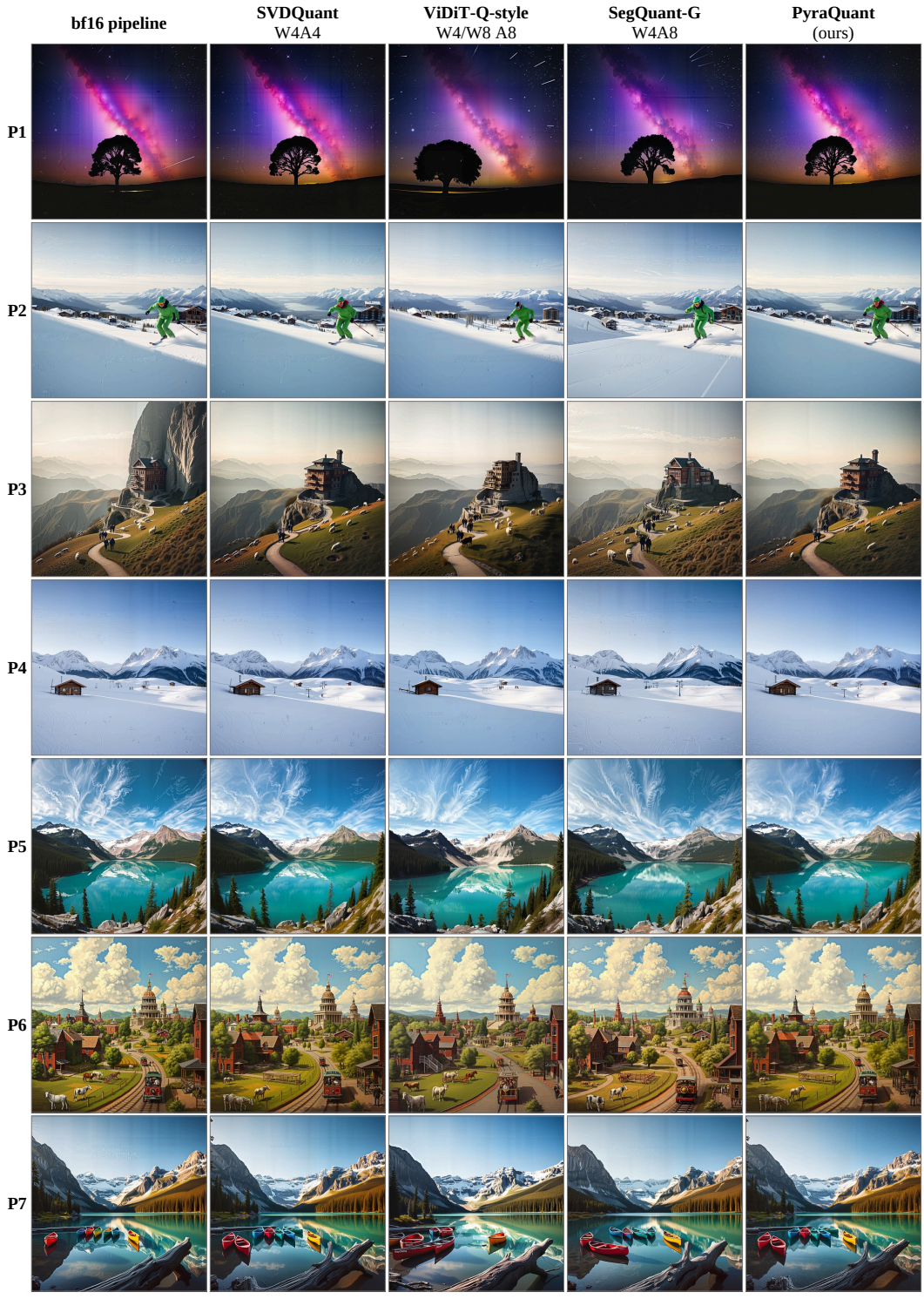

Figure 6: Qualitative comparison on the DiT pipeline at 4096^2, using the same prompt and seed for all methods in each row. PyraQuant uses 2.00 effective bits and 13.6% of the corresponding 16-bit pipeline's BOPs. ViDiT-Q-style distorts the tree crown in P1 and the building in P3; SegQuant-G thins the skier in P2.

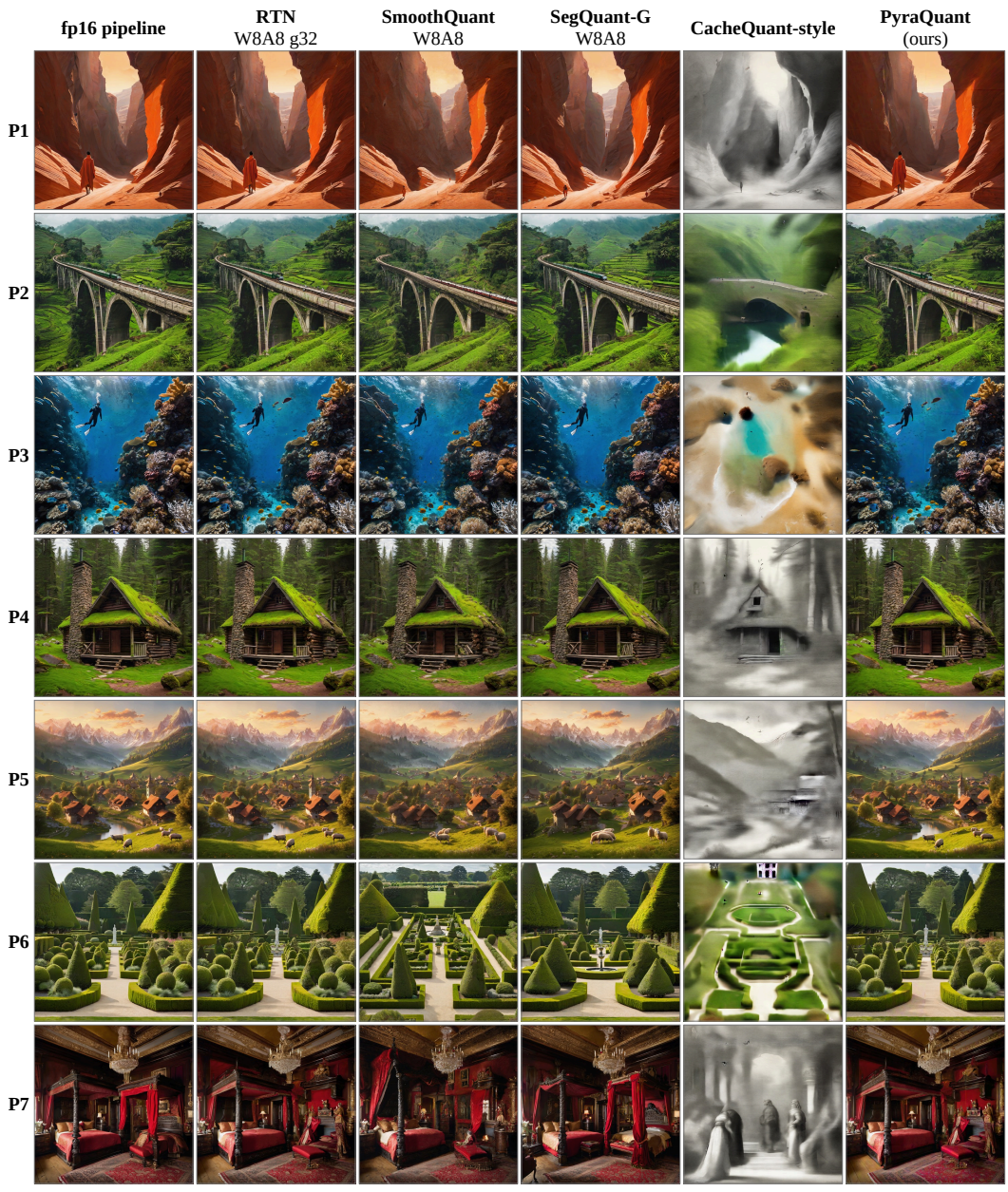

Figure 7: Qualitative comparison on the UNet pipeline at 4096^2, using the same prompt and seed for all methods in each row. PyraQuant uses 5.33 effective bits and 24.3% of the corresponding 16-bit pipeline's BOPs. SmoothQuant and SegQuant-G shrink the walking figure in P1.

Table 17: Runtime ablations on the UNet pipeline. Speedups are relative to the fp16 full-canvas baseline.

Executor	Latency (s)	Speedup
fp16 full canvas	79.3	1.00×
INT8 full canvas	68.9	1.15×
INT8 routed, no cache	78.8	1.01×
PyraQuant executor	59.5	1.33×
fp16 routed + cache	67.2	1.18×

Table 18: Peak GPU memory during the final 4K stage on one NVIDIA L40S GPU.

Pipeline	Method	Peak Memory (GiB)
FLUX	bf16 full canvas	35.8
FLUX	INT8 full canvas	28.1
FLUX	PyraQuant	17.3
SDXL	fp16 full canvas	11.9
SDXL	INT8 full canvas	9.8
SDXL	PyraQuant	8.9

INT8 full-canvas execution reduces final-stage peak GPU memory from 35.8 to 28.1 GiB on FLUX and from 11.9 to 9.8 GiB on SDXL. Combining quantized execution with spatial routing and cached reuse further lowers the peaks to 17.3 and 8.9 GiB, respectively.

E.3 Executor Implementation Details

Quality evaluation applies QDQ to the high- and low-precision states and combines their predictions according to the allocation masks, with cached predictions used in inactive regions. Runtime evaluation uses a separate packed-INT8 executor that evaluates each routed crop in a single quantized pass and bypasses backbone execution in cached regions. Its kernels are implemented in Triton: activations are quantized per token, weights use per-output-channel INT8, and a fused INT8 GEMM accumulates in int32 and applies dequantization, bias, and, for SVDQuant, the 16-bit low-rank up-projection in its epilogue; attention, convolutions, embeddings, and adaLN modulation layers remain in 16-bit precision. The high- and low-precision states share one stored quantized representation.

For the main DiT configuration, the executor uses local crops with a 2-token halo and preserves the original positional information. On the UNet pipeline, execution follows the backbone's minimum crop granularity and uses a halo of 8 latent pixels.

F Reproducibility

All quality comparisons use paired prompts and fixed seeds across methods. Efficiency measurements follow the hardware and execution protocol in Section 4.1 and Appendix E.

We release the inference code of the two main configurations, including the routing executor and the INT8 latency executor, together with the evaluation prompts and seeds, the stage-wise allocation masks, and the threshold calibration script.

PyraQuant preserves the backbone architectures and progressive denoising schedules of the host pipelines and introduces a stage-entry hook for allocation.