arXiv:2608.03057v1 [cs.CV] 4 Aug 2026

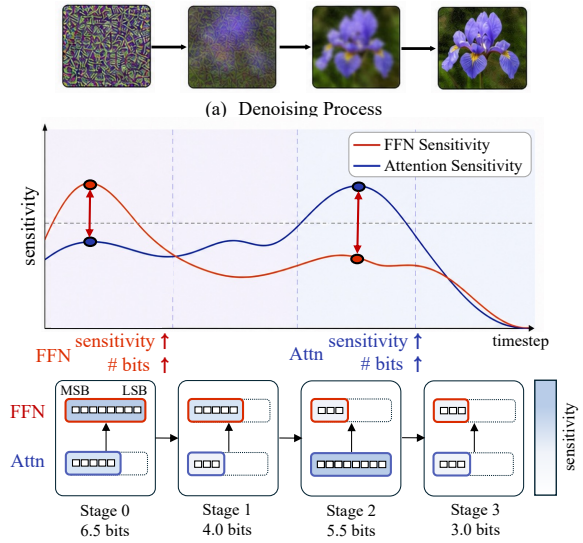
TASQ: Temporal-Adaptive Bit Sparsification Quantization for Diffusion Models

Seokho Han[1], Dongwei Wang[2], Jinhee Kim[3],
Yiran Chen[3], Kang Eun Jeon[4†], Huanrui Yang[2†], Jong Hwan Ko[1†]

[1]Department of Electrical and Computer Engineering, Sungkyunkwan University, Korea
[2]Department of Electrical and Computer Engineering, University of Arizona, USA
[3]Department of Electrical and Computer Engineering, Duke University, USA
[4]Kim Jaechul Graduate School of AI, Korea Advanced Institute of Science and Technology (KAIST)
{beppa2396, jhko}@skku.edu, kejeon@kaist.ac.kr, huanruiyang@arizona.edu

Abstract

Static quantization assigns one weight precision to every denoising step. To preserve quality, that precision must accommodate the most quantization-sensitive step, even though many other steps can tolerate fewer bits. The resulting model may satisfy its memory budget, but it repeatedly pays worst-case arithmetic throughout the denoising trajectory. We introduce **Temporal-Adaptive Bit Sparsification Quantization (TASQ)** to separate these two costs. TASQ stores one shared maximum-precision weight buffer and learns a **Temporal-Spatial LSB Mask** that selects a lower effective precision for each layer and denoising stage by truncating least-significant bits. Storage therefore remains fixed by the worst case, while BitOPs decrease at less sensitive stages without per-stage weight copies or runtime search. A **Temporal-Precision Engine** maps the learned schedule to bit-serial execution, where cycles scale with effective precision and switching precision has no measured cycle overhead. On PixArt-Σ, SANA-1.6B, and SDXL-Turbo, TASQ achieves quality comparable to static quantization with less computation. Together with the Temporal-Precision Engine, it reduces execution cycles by 25–50% over static quantization and by 6.1–7.5× over a naive static 8-bit bit-serial execution. Code is available at https://github.com/seokho-han/tasq.



(b) Diffusion model sensitivity and TASQ's learned bit allocation

Figure 1: Temporal sensitivity and TASQ bit allocation across diffusion stages: FFN sensitivity peaks at noisy steps while attention peaks mid-trajectory, and TASQ assigns higher bits where sensitivity is high.

1 Introduction

Diffusion models (Dhariwal and Nichol 2021; Ho, Jain, and Abbeel 2020; Rombach et al. 2022; Wang et al. 2025a) generate high-quality images through an iterative denoising process, repeatedly inferring the same network to transform a noisy input into a clean sample. Quantization reduces the memory and arithmetic cost of each inference (Gholami et al. 2021) by compressing diffusion weights and activations to 4 bits and lower (Li et al. 2023; Chen et al. 2025).

Meanwhile, we notice a redundancy in Diffusion computation that existing quantization methods fail to address. As shown in Figure 1, quantization sensitivity varies across both timesteps and layer types. FFN layers are most sensitive near the noisy end, whereas attention layers peak later. While mixed-precision quantization method explores layer-wise sensitivity through search-based (Wang et al. 2019; Dong et al. 2019) or learning-based (Yang et al. 2021; Xiao et al. 2023; Han et al. 2025) methods, they maintain same precision across the timesteps, failing to exploit the temporal variation. Recent works adapt precision over denoising steps, but only for activations (Zhang et al. 2026). This is limiting because activation quantization degrades sharply below 3 bits due to outliers (Li et al. 2025), while weight precision remains fixed at the worst-case timestep.

Under such static quantization, the model has to take a precision that stores enough information for the worst case of each layer across all timesteps; while wasting computation in the steps that are less sensitive.

To reduce the temporal redundancy in Diffusion computation, we propose TASQ, the first temporal-adaptive quantizer that can assign different weight precision to model layers at different diffusion steps. Changing precision across timesteps without memory overhead requires TASQ to learn multiple step-specific operating precisions from one shared

†Corresponding author.

weight representation, rather than storing the weight of each step separately. TASQ therefore takes a truncation-based method. From a shared high-precision weight, a **Temporal-Spatial LSB Mask** is proposed to determine how many least-significant bits can be omitted for each layer and stage. To avoid gradient complication for jointly training multiple-precision weights across timesteps, we further propose **Farthest-Stage-First Training** that separates the weight updates across distant stages. At inference, TASQ requires only a table lookup for precision determination and bit-plane truncation for operating weight generation, without additional weight copies, search, or repacking. Our main contributions are summarized as follows:

- We decouple storage and operating precision, allowing weight precision to vary across layers and denoising stages to reduce BitOPs.
- We propose Temporal-Spatial LSB Mask, which learns stage and layer-wise precision over a single shared weight representation.
- We design the Temporal-Precision Engine, which streams only the selected weight planes and achieves 6.1–$7.5\times$ fewer cycles than a naive static 8-bit execution while preserving FID and ImageReward.

2 Related Works

Diffusion Model Quantization. Diffusion quantization first established 8-bit PTQ baselines with Q-Diffusion (Li et al. 2023) and PTQ4DM (Shang et al. 2023). Later work explored sensitivity-aware calibration (Yang et al. 2023), timestep-aware quantization (Huang et al. 2024; He et al. 2023; Wang et al. 2024), text-to-image models (Tang et al. 2024), and DiT backbones (Wu et al. 2024; Chen et al. 2025). LoRA-based QAT methods such as EfficientDM (Hu et al. 2022; He et al. 2024) recover quality with limited retraining, while BinaryDM (Zheng et al. 2024), QuEST (Wang et al. 2025b), and BitsFusion (Sui et al. 2024) target sub-2-bit weights. These methods allocate precision spatially and reuse that allocation across timesteps.

Several low-bit pipelines protect selected components—error-sensitive tokens in ViDiT-Q (Zhao et al. 2024a), the BOS text token in MixDQ (Zhao et al. 2024b), or a 16-bit low-rank branch in SVDQuant (Li et al. 2025)—and quantize the remaining model more aggressively. The protected components and bit allocation are fixed after calibration. The most sensitive timestep therefore sets a precision floor for the rest of the trajectory.

Mixed-Precision and Bit-Level Quantization. Mixed-precision quantization exploits non-uniform layer-wise sensitivity. HAQ searches for layer bit-widths with reinforcement learning (Wang et al. 2019), while HAWQ and HAWQ-V2 use second-order sensitivity (Dong et al. 2019, 2020); both face a combinatorial allocation space. Bit-level methods learn the allocation during training instead: BSQ optimizes individual bits (Yang et al. 2021), CSQ uses a continuous relaxation (Xiao et al. 2023), and MSQ directly regularizes the LSB itself of the quantized weights without bit-splitting overhead (Han et al. 2025).

These methods nevertheless learn one fixed spatial weight-precision allocation. In contrast, TASQ learns multiple stage-specific operating precisions by applying different masks to one shared quantized weight.

Timestep-Aware Quantization. A recent line of work makes diffusion quantization timestep-aware, but not through dynamic weight precision. TCAQ-DM (Huang et al. 2025) adapts activation ranges at a uniform bit-width; MPQ-DM (Feng et al. 2025) and MPQ-Diff (Maruzzelli, Lewandowski, and Chen 2024) choose per-layer allocations that remain fixed across steps; and AdaTSQ (Zhang et al. 2026) searches per-timestep activation bits while keeping weights static to avoid memory overhead. These methods are complementary to TASQ, which learns the effective *weight* precision for each layer and stage from a single shared buffer. Table 11 summarizes this distinction.

3 Methods

We propose TASQ, a temporal-adaptive quantization framework that varies weight precision across denoising stages. TASQ uses a shared weight representation that allows lower-bit weights to be obtained by direct truncation. Because precision choices are coupled through the shared weights, we introduce Temporal-Spatial LSB Masking to learn the precision allocation, and optimize it together with the shared weights using parameter-efficient LoRA-based QAT. However, jointly updating the shared weights across neighboring stages can produce highly correlated gradients and cause training to plateau. To address this issue, Farthest-Stage-First Training separates the updates across distant denoising stages and stabilizes optimization. Since operating precision changes across the denoising trajectory, we also co-design the Temporal-Precision Engine to execute the resulting precision schedule efficiently by reusing activation bit planes and streaming only the required weight bits.

3.1 Preliminaries

Quantizer for Adaptive Precision. For a normalized weight $W \in [0, 1]$, the n-bit quantized weight and its directly truncated $(n-1)$-bit weight are defined as

$$Q_n = \lfloor W \cdot 2^n \rfloor, \qquad Q_{n-1} = \left\lfloor \frac{Q_n}{2} \right\rfloor. \tag{1}$$

Here, $\lfloor \cdot \rfloor$ denotes the floor operation. Equation (1) ensures that the lower-precision weight is obtained by removing the least-significant bit from the higher-precision weight. Switching precision therefore requires only truncation or a right shift, rather than a separately quantized weight copy.

Efficient QAT using LoRA. To bridge the gap between the performance of Quantization-Aware Training (QAT) and the efficiency of Post-Training Quantization (PTQ), recent studies (He et al. 2024; Jeon, Kim, and Kim 2025) propose integrating LoRA into the quantization. Specifically, these methods apply the quantization operator to the effective weight formed by merging learnable low-rank adapters with frozen original weights, formulated as:

$$Y = Q(X)Q(W + BA), \tag{2}$$

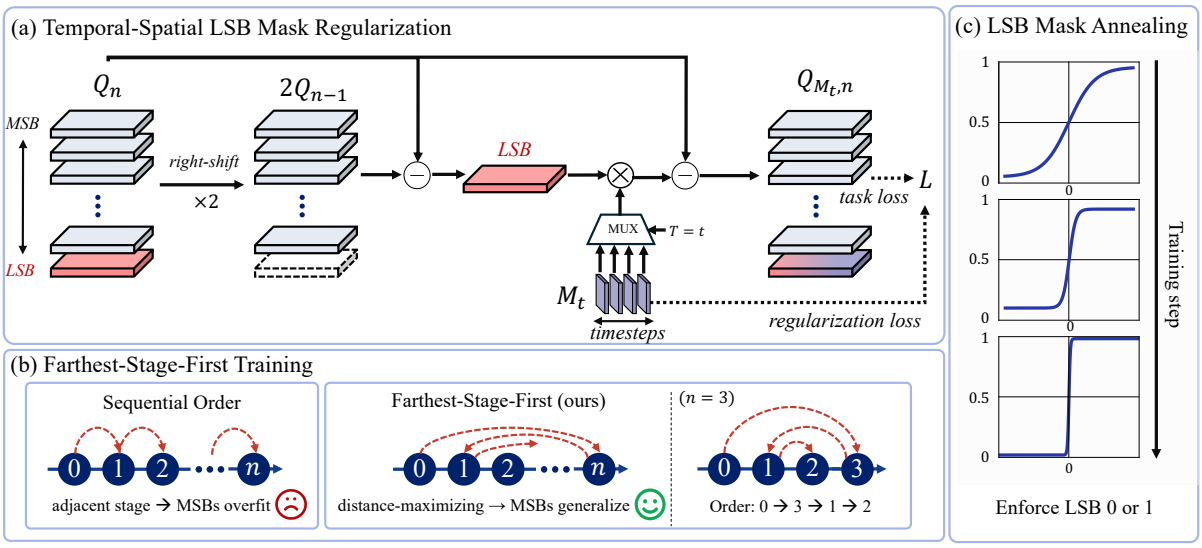


Figure 2: Process of Temporal–Spatial LSB Mask Regularization and LSB Mask Annealing.

where W represents the frozen weights, and B and A denote the trainable low-rank matrices. This yields fully quantized weights for efficient bit-wise inference while training only the low-rank parameters.

3.2 TASQ

TASQ starts from a single quantized weight representation for each layer and introduces learnable Temporal-Spatial LSB masks to decompose it into multiple stage-specific operating precisions. This allows the model to vary its operating precision without storing separate weight copies. Unlike previous bit-level methods that directly regularize the bits of quantized weights, TASQ optimizes the masks jointly with the LoRA parameters. Activation precision remains fixed in this work and can be adapted independently.

Temporal-Spatial LSB Mask For a stored n-bit quantized weight, we extract its least-significant bit from the difference between Q_n and the truncated $(n-1)$-bit weight:

$$\text{LSB} = Q_n - 2 \cdot Q_{n-1}. \tag{3}$$

Although the LSB is extracted from the shared quantized weight, it remains shared across all stages and layers. We therefore introduce a mask $M_{t,l}$ that allows the same LSB to take different sparsity states across stages and layers:

$$Q_{M,n}^{(t,l)} = Q_n - (1 - M_{t,l}) \cdot \text{LSB}. \tag{4}$$

Here, $M_{t,l}$=1 retains the bit, while $M_{t,l}$=0 removes it. Applying the same operation recursively to the remaining least-significant bits determines the operating precision while preserving the shared high-precision weight.

Training Objective. We jointly optimize the LoRA parameters and masks using the full-precision model as a teacher:

$$\mathcal{L}_{\text{total}} = \|\epsilon_q - \epsilon_{\text{fp}}\|_2^2 + \lambda_M \sum_{t,l} M_{t,l}. \tag{5}$$

The first term matches the predicted noise, while the second penalizes the retained least-significant bits. Under the straight-through estimator, Q_n and Q_{n-1} share the same weight-gradient path up to their power-of-two scale, so the extracted LSB is treated as locally constant:

$$\frac{\partial \text{LSB}}{\partial W} \approx 0. \tag{6}$$

The weight gradient therefore follows the masked quantized output:

$$\frac{\partial \mathcal{L}}{\partial W} \approx \frac{\partial \mathcal{L}}{\partial Q_{M,n}^{(t,l)}}. \tag{7}$$

Differentiating Eq. (4) with respect to $M_{t,l}$ gives

$$\frac{\partial \mathcal{L}}{\partial M_{t,l}} = \frac{\partial \mathcal{L}}{\partial Q_{M,n}^{(t,l)}} \cdot \text{LSB} + \lambda_M, \tag{8}$$

Thus, the LSB is retained when its contribution to the distillation loss outweighs the regularization λ_M; otherwise, the mask is driven toward zero. Figure 2 illustrates this process.

LSB Mask Annealing. Direct binary selection is not differentiable, so we use a continuous gate during training and anneal it to a binary decision, following the sparsification schedule of PAT (Liu et al. 2025b). For training step s and annealing horizon s_0, define

$$\tau(s) = \begin{cases} \dfrac{1}{1 - \dfrac{\ln(s)}{\ln(s_0)}}, & s < s_0, \\ \epsilon^{-1}, & \text{otherwise} \end{cases} \qquad \beta(s) = \begin{cases} -\dfrac{s}{s_0} + 0.5, & s < s_0/2, \\ 0, & \text{otherwise} \end{cases} \tag{9}$$

The continuous gate is

$$M_{t,l} = \mathcal{G}(s, m_{t,l}) = \frac{1}{1 + e^{-\tau(s) \cdot m_{t,l}}} + \beta(s), \tag{10}$$

where $m_{t,l}$ is learned. The bias initializes the gate near one, and increasing temperature binarizes it.

Figure 3: Qualitative Image Generation Results on PixArt-Σ, SANA-1.6B, SDXL-Turbo.

Table 1: Bit-plane dataflows for L weight and R=4 activation planes. Keeping activations on chip reduces the fetch count to $L+R$, bit-exactly. Algorithm 2 gives the schedules.

	Fetch planes		Total cycles		
Schedule	model	L=8	L=3	L=4	L=8
Naive (per-pass refetch)	$2LR$	64	13,853	18,413	36,653
Weight-stationary	$L+LR$	40	8,723	11,573	22,973
Activation-stationary (ours)	**L+R**	**12**	**4,313**	**4,883**	**7,163**

Stage-wise Training. Adjacent timesteps learn similar bit allocations, consistent with the temporal redundancy observed in diffusion trajectories (Whalen et al. 2025; Wang et al. 2025c). We therefore partition the trajectory into contiguous stages and share $M_{t,l}$ within each stage. This reduces the number of mask parameters while retaining the temporal variation; the elbow falls at four stages (Appendix F).

Farthest-Stage-First Training. Because all stages share the same MSBs, training neighboring stages in succession leads to highly correlated updates and caused the loss to plateau early in our experiments. We therefore train stages in a farthest-stage-first order, placing consecutive updates as far apart as possible along the trajectory. For four stages, the order is $0 \to 3 \to 1 \to 2$. This exposes the shared MSBs to more diverse parts of the trajectory while preserving a separate LSB mask for each stage.

3.3 Algorithm–Hardware Co-design

TASQ requires hardware whose execution cost scales with the selected precision. We therefore co-design the **Temporal-Precision Engine** to execute the learned weight schedule. In bit-serial GEMM, an L-bit weight and an R-bit activation require $L \times R$ binary operations. A naive implementation reloads an operand for each pass, which limits the benefit of reducing weight precision.

Since TASQ varies only the weight precision, our engine keeps the activation bit planes on chip and streams only the weight planes selected for the current layer and stage. This reduces the number of fetched planes from $2LR$ to $L+R$ while preserving bit-exact outputs. We implement this dataflow in software based on BISMO accelerator (Umuroglu, Rasnayake, and Själander 2018). As shown in Table 1, it reduces execution cycles by 3.2–5.1× over a naive bit-plane loop and introduces no measured cycle overhead when precision changes.

4 Experiments

We report generation quality and BitOPs in Table 2, inspect the learned allocation in Figure 4, ablate the mask and training schedule, and measure hardware cost in §4.3. To separate the effect of temporal adaptivity from that of SVDQuant initialization, we evaluate TASQ under two settings. Table 2 compares the original SVDQuant model, the same model further adapted through QAT, and the model trained with TASQ. Table 4 then compares QAT and TASQ when both are trained without SVDQuant initialization. Together, these experiments evaluate whether TASQ remains effective both with and without a strong PTQ starting point.

4.1 Experimental Setup

Models and Denoising Schedules. We evaluate the DiT models PixArt-Σ (0.6B) (Chen et al. 2024) and SANA-1.6B (Xie et al. 2024), and the U-Net model SDXL-Turbo (2.6B) (Podell et al. 2023; Sauer et al. 2023). PixArt-Σ and SANA use 20 denoising steps grouped into four stages. SDXL-Turbo uses four steps, so each step has its own stage.

Table 2: Quality–compute comparison. "Eff. W" denotes the average operating weight precision and "Rel. BitOPs" the compute cost normalized to static W8A8. TASQ stores one shared 8-bit weight buffer across all stages. W4 rows are compute-matched, while W8 TASQ retains W8 quality with 74–75% compute. Non-SVDQuant controls are reported in Table 4.

Backbone	Model	A-Bit	Eff. W-Bit	Rel. BitOPs	Method	MJHQ				sDCI			
						FID (↓)	IR (↑)	LPIPS (↓)	PSNR (↑)	FID (↓)	IR (↑)	LPIPS (↓)	PSNR (↑)
DiT	PixArt-Σ (20 Steps)	16	16	4.00	FP	16.6	0.944	–	–	24.8	0.966	–	–
		8	8.00	1.00	ViDiT-Q	15.7	0.944	0.137	22.5	**23.5**	**0.974**	0.163	20.4
			8.00	1.00	SVDQuant	16.3	**0.955**	**0.109**	**23.7**	24.2	0.969	**0.129**	**21.8**
			5.91	**0.74**	**SVDQuant+TASQ**	**15.6**	**0.955**	0.112	**23.7**	23.9	0.968	0.132	**21.8**
		8	4.00	0.50	ViDiT-Q	37.3	0.573	0.611	12.0	40.6	0.600	0.629	11.2
			4.00	0.50	SVDQuant	17.8	0.915	0.290	19.2	24.6	0.942	0.315	17.8
			4.00	0.50	SVDQuant+QAT	17.0	0.928	0.268	19.8	24.2	0.955	0.198	18.5
			4.00	**0.50**	**SVDQuant+TASQ**	**16.5**	**0.942**	**0.242**	**20.5**	**23.8**	**0.968**	**0.173**	**19.2**
		4	4.00	0.25	ViDiT-Q	412	-2.27	0.854	6.44	425	-2.28	0.838	6.70
			4.00	0.25	SVDQuant	19.2	0.878	0.323	17.6	25.9	0.918	0.352	16.5
			4.00	0.25	SVDQuant+QAT	17.8	0.901	0.308	18.0	24.8	0.938	0.318	17.3
			3.98	**0.25**	**SVDQuant+TASQ**	**16.1**	**0.932**	**0.282**	**18.6**	**23.1**	**0.966**	**0.263**	**18.8**
	SANA -1.6B (20 Steps)	16	16	4.00	FP	16.2	1.10	–	–	22.4	1.07	–	–
		8	8.00	1.00	SVDQuant	**15.9**	1.095	**0.243**	**18.4**	22.4	1.016	**0.221**	**17.5**
			5.98	**0.75**	**SVDQuant+TASQ**	**15.9**	**1.096**	0.274	17.6	**22.2**	**1.066**	0.291	16.3
		8	4.00	0.50	SVDQuant	17.7	1.018	0.241	17.7	22.5	1.018	**0.264**	16.3
			4.00	0.50	SVDQuant+QAT	16.8	1.048	0.252	17.7	22.5	1.032	0.271	16.4
			3.99	**0.50**	**SVDQuant+TASQ**	**15.9**	**1.096**	0.264	17.8	**22.4**	**1.060**	0.280	**16.6**
		4	4.00	0.25	RTN	20.5	0.894	0.339	15.3	28.6	0.807	0.371	13.8
			4.00	0.25	SVDQuant	19.3	0.935	**0.220**	17.8	28.1	0.846	**0.242**	**16.2**
			4.00	0.25	SVDQuant+QAT	18.5	0.952	0.248	17.5	26.4	0.878	0.263	15.9
			3.99	**0.25**	**SVDQuant+TASQ**	**16.6**	**1.074**	0.297	17.1	**21.7**	**1.060**	0.320	15.8
UNet	SDXL-Turbo (4 Steps)	16	16	4.00	FP	24.3	0.845	–	–	24.7	0.847	–	–
		8	8.00	1.00	MixDQ	24.1	0.834	0.147	21.7	25.0	0.690	0.157	21.6
			8.00	1.00	SVDQuant	24.3	**0.845**	**0.100**	24.0	24.8	0.701	**0.110**	**23.7**
			5.95	**0.74**	**SVDQuant+TASQ**	**24.0**	**0.845**	0.122	**24.2**	**24.5**	**0.730**	0.121	23.5
		8	4.00	0.50	MixDQ	27.7	0.708	0.402	15.7	25.9	0.610	0.415	15.7
			4.00	0.50	SVDQuant	24.5	0.835	0.225	19.0	25.1	0.692	0.232	19.1
			4.00	0.50	SVDQuant+QAT	24.2	0.840	0.215	19.4	24.9	0.698	0.220	19.4
			3.97	**0.50**	**SVDQuant+TASQ**	**23.8**	**0.845**	**0.198**	**19.76**	**24.7**	**0.703**	**0.205**	**19.77**
		4	4.00	0.25	MixDQ	353	-2.26	0.685	11.0	373	-2.287	0.686	11.37
			4.00	0.25	SVDQuant	24.6	0.816	0.262	18.11	25.2	0.671	0.272	18.0
			4.00	0.25	SVDQuant+QAT	24.2	0.823	0.255	18.25	25.0	0.678	0.264	18.15
			3.98	**0.25**	**SVDQuant+TASQ**	**23.6**	**0.836**	**0.244**	**18.50**	**24.7**	**0.691**	**0.252**	**18.45**

Quantization and Methods. We report W8A8, W4A8, and W4A4; Appendix A.1 gives the quantization details. The PTQ baselines are ViDiT-Q, MixDQ, RTN, and SVDQuant. The fine-tuned baselines are QAT and SVDQuant+QAT, with SVDQuant+QAT serving as the non-temporal control for SVDQuant+TASQ. SVDQuant uses a rank-16 low-rank branch at the average 6-bit setting and rank 32 at 4 bits. TASQ uses rank-32 LoRA adapters B, A in all settings. Appendix B compares the LSB mask with a standard sigmoid gate.

Training. We train TASQ under two settings, using either pure LoRA-QAT for 5.0k iterations or SVDQuant-initialized adaptation for 1.4k iterations. In both settings, we optimize the LoRA parameters and masks with $\lambda_M{=}5 \times 10^{-5}$ and prune the masks every 0.1k iterations. We divide the denoising trajectory into four temporal stages based on the elbow-point analysis in Appendix F and apply the cached-feature teacher–student distillation described in Appendix A.2. Per-pipeline hyperparameters and the shared calibration and training costs of SVDQuant, QAT, and TASQ are detailed in Appendix C.

Datasets and Metrics. Training prompts are sampled from COCO Captions (Chen et al. 2015). For evaluation, we use 5K prompts each from MJHQ-30K (Li et al. 2024) and sDCI (Urbanek et al. 2024), covering stylized and densely captioned image distributions. We report FID (↓) and ImageReward (↑) for quality, and LPIPS (↓) and PSNR (↑) for

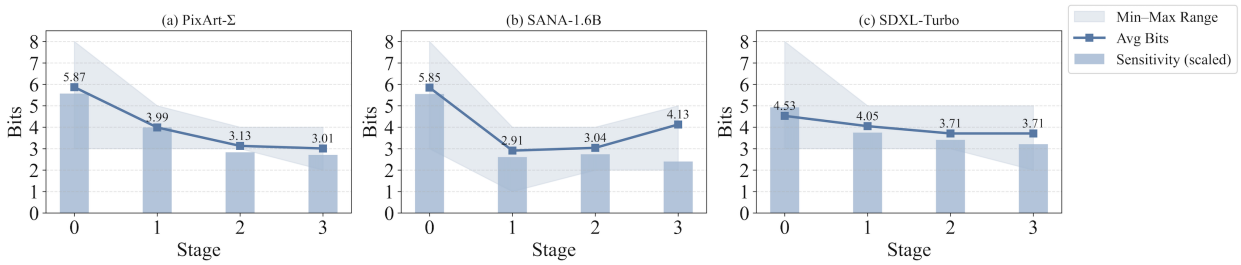


Figure 4: Effective weight precision across temporal stages under the W4A4 operating budget. The static allocation must cover the most sensitive stage, whereas TASQ follows the stage-dependent sensitivity and uses fewer bits elsewhere.

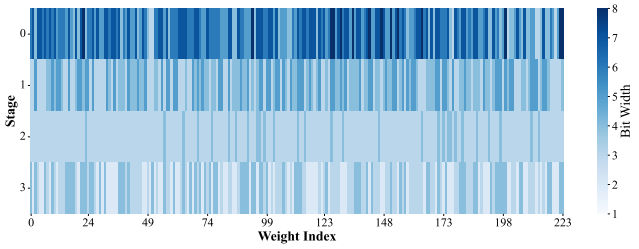


Figure 5: TASQ-learned operating precision across temporal stages and layers for PixArt-Σ under W4A4.

fidelity to the full-precision model, on both datasets.

Analysis of Temporal-Adaptiveness. The learned precision follows the sensitivity profile in Figure 4. The noisiest stage generally receives more bits, whereas later stages require fewer, and the variation within each stage shows that layers do not share the same precision requirement. A static allocation must follow the upper envelope of these demands, whereas TASQ adapts the precision to each stage and layer.

This trend reflects how quantization error propagates through the denoising process. Errors introduced at early, noisy stages affect more subsequent updates and are further amplified by the noise schedule, whereas errors introduced later pass through fewer updates and can tolerate lower weight precision. However, this temporal trend is not uniform across layers. Attention and FFN layers are most sensitive at different parts of the trajectory, as shown in Figure 1, while the wide min–max bands in Figure 4 indicate substantial layer-wise variation within each stage. TASQ therefore learns both the overall temporal trend and the layer-specific deviations. A timestep-only schedule cannot capture the latter, while a spatial-only schedule obscures the former.

4.2 Ablation Study

Effectiveness of Temporal–Spatial Adaptation. We examine whether temporal and spatial adaptation provide complementary benefits by comparing spatial-only, temporal-only, and joint temporal–spatial precision allocation. All settings store the same 8-bit model and use the same average 4-bit operating precision on PixArt-Σ. Spatial-only allocation achieves an FID of 17.1, while temporal-only allocation achieves 18.4. Combining both forms of adaptation improves the FID to **16.1**. These results show that allocating precision jointly across layers and denoising stages is more effective than adapting along either dimension alone.

Effect of the LSB Mask Design. Replacing the LSB mask with a sigmoid timestep gate increases the PixArt-Σ W4A4 FID from 16.1 to 21.4 (Appendix B), suggesting that gradually suppressing the LSB signal before the model adapts is harmful.

TASQ without SVDQuant Initialization. To determine whether the gain depends on the SVDQuant starting point, Table 4 compares standard LoRA-QAT with TASQ when both are trained directly from the same initialization. TASQ improves ImageReward in every setting and lowers FID in all but one case, indicating that the benefit comes from temporal adaptivity rather than the initialization itself.

4.3 Efficiency Discussion

Operating regime. TASQ does not directly target reducing the stored model size. The most sensitive stage still determines the shared weight precision, which is stored once throughout inference. Instead, TASQ reduces computation by executing less sensitive stages with fewer weight bits. Table 2 therefore reports Rel. BitOPs, not storage. On precision-scalable datapaths, lower BitOPs translate directly into fewer execution cycles, whereas fixed INT4/INT8 kernels cannot execute below their native precision.

Deployment procedure. Efficient deployment must consider both memory usage and inference speed. We therefore first choose the storage precision according to the available memory budget and then use TASQ to reduce the repeated denoising computation. The model is quantized and stored once at the highest precision that fits within the memory budget. TASQ is then trained to select among the lower precisions supported by the target device. The stored weight buffer remains unchanged throughout inference, and TASQ only determines how many bit planes each layer reads at each denoising stage. For example, an 8-bit deployment stores a single 8-bit model, while selected layers can execute at 3-bit or 2-bit precision by reading fewer bit planes. This procedure fixes the memory footprint while allowing TASQ to improve inference speed through stage- and layer-specific allocation.

Training efficiency. The masks add little optimization cost. With the same 5.0k-iteration schedule, TASQ matches QAT in memory and wall-clock time. Starting from SVDQuant, the masks and LoRA adapters require another

Table 3: Ablation of temporal and spatial precision adaptation on PixArt-Σ. All settings store the same 8-bit model and use the same average 4-bit operating precision. **Temporal** and **Spatial** indicate whether precision varies across denoising stages and layers, respectively.

		Adaptation		MJHQ				sDCI			
Method	Eff. W-Bit	Temporal	Spatial	FID (↓)	IR (↑)	LPIPS (↓)	PSNR (↑)	FID (↓)	IR (↑)	LPIPS (↓)	PSNR (↑)
Spatial only	4.00	✗	✓	17.1	0.904	0.297	18.3	24.2	0.926	0.283	17.2
Temporal only	4.00	✓	✗	18.4	0.909	0.290	18.5	25.3	0.929	0.333	17.0
TASQ	3.98	✓	✓	**16.1**	**0.932**	**0.282**	**18.6**	**23.1**	**0.966**	**0.263**	**18.8**

Table 4: TASQ versus non-temporal QAT without SVDQuant initialization.

			MJHQ		sDCI	
Backbone	Setting	Method	FID (↓)	IR (↑)	FID (↓)	IR (↑)
PixArt-Σ	W4A8	QAT	18.5	0.908	25.0	0.940
	W4A8	TASQ	**17.2**	**0.925**	**24.3**	**0.952**
	W4A4	QAT	19.0	0.885	25.5	0.922
	W4A4	TASQ	**17.5**	**0.908**	**24.6**	**0.935**
SANA-1.6B	W4A8	QAT	17.5	1.025	22.8	1.010
	W4A8	TASQ	**17.0**	**1.042**	**22.5**	**1.028**
	W4A4	QAT	19.2	0.940	27.0	0.855
	W4A4	TASQ	**18.2**	**0.968**	**26.0**	**0.885**
SDXL-Turbo	W4A8	QAT	24.9	0.830	**24.7**	0.679
	W4A8	TASQ	**24.3**	**0.838**	25.0	**0.695**
	W4A4	QAT	25.1	0.810	25.4	0.665
	W4A4	TASQ	**24.0**	**0.828**	**24.9**	**0.682**

Table 5: Training cost at W4A8. "+TASQ" is the additional adaptation after SVDQuant.

Model	Method	Iter.	Time (h)	Mem. (GB)	FID
SANA-1.6B	SVDQuant	–	6.8	18.6	17.7
	+TASQ	+1.4k	+0.5	9.8	**15.9**
	QAT	5.0k	1.8	9.1	17.5
	TASQ	5.0k	1.8	9.1	17.0
SDXL-Turbo	SVDQuant	–	10.4	12.4	24.5
	+TASQ	+1.4k	+2.1	16.7	**23.8**
	QAT	5.0k	8.2	11.6	24.9
	TASQ	5.0k	8.2	11.6	24.3

1.4k iterations: 0.5 hours on SANA-1.6B and 2.1 hours on SDXL-Turbo (Table 5).

Inference efficiency. We measure bit-serial GEMM on our Temporal-Precision Engine, whose execution cost scales with the selected precision. As shown in Table 6, execution cycles scale linearly with the product of weight and activation bit widths, while switching precision introduces no measured cycle overhead. The execution cost of a mixed-precision schedule therefore equals the sum of the measured cycles at each precision. Under this measured cost model, TASQ retains static W8 quality while reducing execution cycles by approximately 25%, corresponding to an average operating weight precision of about 6 bits. At the same average W4A4 computational budget, TASQ improves MJHQ FID by 1.0–3.1 points over static quantization. The Temporal-Precision Engine therefore realizes TASQ's adaptive precision schedule with execution time proportional to the selected precision and no measured overhead for precision switching.

Table 6: Measured four-tile GEMM cost on the Temporal-Precision Engine with A4. The mixed schedule incurs no precision-switching overhead.

	Static (one precision × 4 tiles)						**Mixed**
4 tiles, A=4	W1	W2	W3	W4	W6	W8	**1, 2, 3, 4**
Cycles	528	1,056	1,584	2,112	3,168	4,224	**1,320** (+0)
Latency (μs)	2.89	5.78	8.67	11.57	17.35	23.13	**7.23**
Energy (μJ)	1.89	3.78	5.67	7.56	11.35	15.13	**4.73**

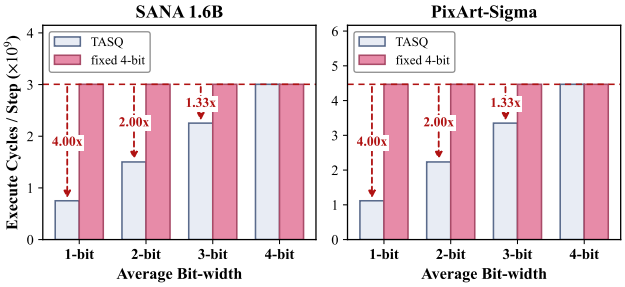


Figure 6: Execute cycles per denoising step under the measured bit-serial cycle law: TASQ executes at its average effective bit-width, while a fixed 4-bit datapath cannot descend below 4-bit.

5 Conclusion

We introduced TASQ, a temporal weight-precision method that separates stored precision from the precision used during denoising. Its stage- and layer-specific LSB masks operate on one shared buffer, avoiding duplicated checkpoints while adapting computation to the denoising trajectory. Across DiT and U-Net backbones, TASQ retained the quality of static 8-bit quantization with less computation and improved generation quality under the same average compute budget. The Temporal-Precision Engine further translated the reduced bit-plane computation into measured cycle savings on precision-scalable hardware. These results show that temporal weight allocation improves the quality–compute tradeoff while preserving a single shared weight representation.

References

Askarihemmat, M.; Wagner, S.; Bilaniuk, O.; Hariri, Y.; Savaria, Y.; and David, J.-P. 2023. BARVINN: Arbitrary Precision DNN Accelerator Controlled by a RISC-V CPU. In *Proceedings of the 28th Asia and South Pacific Design Automation Conference (ASP-DAC)*.

Chen, J.; Ge, C.; Xie, E.; Wu, Y.; Yao, L.; Ren, X.; Wang, Z.; Luo, P.; Lu, H.; and Li, Z. 2024. PixArt-Sigma: Weak-to-Strong Training of Diffusion Transformer for 4K Text-to-Image Generation. arXiv:2403.04692.

Chen, L.; Meng, Y.; Tang, C.; Ma, X.; Jiang, J.; Wang, X.; Wang, Z.; and Zhu, W. 2025. Q-DiT: Accurate Post-Training Quantization for Diffusion Transformers. In *Proceedings of the Computer Vision and Pattern Recognition Conference (CVPR)*, 28306–28315.

Chen, X.; Fang, H.; Lin, T.-Y.; Vedantam, R.; Gupta, S.; Dollar, P.; and Zitnick, C. L. 2015. Microsoft COCO Captions: Data Collection and Evaluation Server. *arXiv preprint arXiv:1504.00325*.

Dhariwal, P.; and Nichol, A. 2021. Diffusion models beat GANs on image synthesis. In *Proceedings of the 35th International Conference on Neural Information Processing Systems*, NIPS '21. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713845393.

Dong, Z.; Yao, Z.; Arfeen, D.; Gholami, A.; Mahoney, M. W.; and Keutzer, K. 2020. HAWQ-V2: Hessian Aware Trace-Weighted Quantization of Neural Networks. In *Advances in Neural Information Processing Systems (NeurIPS)*, volume 33, 18518–18529.

Dong, Z.; Yao, Z.; Gholami, A.; Mahoney, M. W.; and Keutzer, K. 2019. HAWQ: Hessian Aware Quantization of Neural Networks with Mixed-Precision. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 293–302.

Feng, W.; Qin, H.; Yang, C.; An, Z.; Huang, L.; Diao, B.; Wang, F.; Tao, R.; Xu, Y.; and Magno, M. 2025. MPQ-DM: Mixed Precision Quantization for Extremely Low Bit Diffusion Models. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Gholami, A.; Kim, S.; Dong, Z.; Yao, Z.; Mahoney, M. W.; and Keutzer, K. 2021. A Survey of Quantization Methods for Efficient Neural Network Inference. *arXiv preprint arXiv:2103.13630*.

Han, S.; Yoon, S.; Kim, J.; Wang, D.; Jeon, K. E.; Yang, H.; and Ko, J. H. 2025. MSQ: Memory-Efficient Bit Sparsification Quantization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 21885–21894.

He, Y.; Liu, J.; Wu, W.; Zhou, H.; and Zhuang, B. 2024. EfficientDM: Efficient Quantization-Aware Fine-Tuning of Low-Bit Diffusion Models. In *International Conference on Learning Representations (ICLR)*.

He, Y.; Liu, L.; Liu, J.; Wu, W.; Zhou, H.; and Zhuang, B. 2023. PTQD: Accurate Post-Training Quantization for Diffusion Models. *arXiv preprint arXiv:2305.10657*.

Ho, J.; Jain, A.; and Abbeel, P. 2020. Denoising diffusion probabilistic models. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20. Red Hook, NY, USA: Curran Associates Inc. ISBN 9781713829546.

Hu, E. J.; Shen, Y.; Wallis, P.; Allen-Zhu, Z.; Li, Y.; Wang, S.; Wang, L.; and Chen, W. 2022. LoRA: Low-Rank Adaptation of Large Language Models. In *International Conference on Learning Representations*.

Huang, H.; Chen, J.; Guo, J.; Zhan, R.; and Wang, Y. 2025. TCAQ-DM: Timestep-Channel Adaptive Quantization for Diffusion Models. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

Huang, Y.; Gong, R.; Liu, J.; Chen, T.; and Liu, X. 2024. TFMQ-DM: Temporal Feature Maintenance Quantization for Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

Jeon, H.; Kim, Y.; and Kim, J.-j. 2025. L4Q: Parameter Efficient Quantization-Aware Fine-Tuning on Large Language Models. arXiv:2402.04902.

Li, D.; Kamko, A.; Akhgari, E.; Sabet, A.; Xu, L.; and Doshi, S. 2024. Playground v2.5: Three Insights towards Enhancing Aesthetic Quality in Text-to-Image Generation. arXiv:2402.17245.

Li, M.; Lin, Y.; Zhang, Z.; Cai, T.; Li, X.; Guo, J.; Xie, E.; Meng, C.; Zhu, J.-Y.; and Han, S. 2025. SVDQuant: Absorbing Outliers by Low-Rank Components for 4-Bit Diffusion Models. In *The Thirteenth International Conference on Learning Representations*.

Li, X.; Liu, Y.; Lian, L.; Yang, H.; Dong, Z.; Kang, D.; Zhang, S.; and Keutzer, K. 2023. Q-Diffusion: Quantizing Diffusion Models. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 17535–17545.

Liu, S.; Zeng, C.; Yan, C.; Peng, X.; Wang, X.; Chen, F.; and Mei, X. 2025a. Error Propagation Mechanisms and Compensation Strategies for Quantized Diffusion. *arXiv preprint arXiv:2508.12094*.

Liu, Y.; Yang, H.; Chen, Y.; Zhang, R.; Wang, M.; Du, Y.; and Du, L. 2025b. PAT: Pruning-Aware Tuning for Large Language Models. *Proceedings of the AAAI Conference on Artificial Intelligence*, 39(23): 24686–24695.

Maruzzelli, R. M.; Lewandowski, B.; and Chen, L. Y. 2024. MPQ-Diff: Mixed Precision Quantization for Diffusion Models. arXiv:2412.00144.

Podell, D.; English, Z.; Lacey, K.; Blattmann, A.; Dockhorn, T.; Muller, J.; Penna, J.; and Rombach, R. 2023. SDXL: Improving Latent Diffusion Models for High-Resolution Image Synthesis. *ArXiv*, abs/2307.01952.

Rombach, R.; Blattmann, A.; Lorenz, D.; Esser, P.; and Ommer, B. 2022. High-Resolution Image Synthesis With Latent Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 10684–10695.

Sauer, A.; Lorenz, D.; Blattmann, A.; and Rombach, R. 2023. Adversarial Diffusion Distillation. *arXiv preprint arXiv:2311.17042*.

Shang, Y.; Yuan, Z.; Xie, B.; Wu, B.; and Yan, Y. 2023. Post-Training Quantization on Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 1972–1981.

Sui, Y.; Li, Y.; Kag, A.; Idelbayev, Y.; Cao, J.; Hu, J.; Sagar, D.; Yuan, B.; Tulyakov, S.; and Ren, J. 2024. BitsFusion: 1.99 bits Weight Quantization of Diffusion Model. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *Advances in Neural Information Processing Systems*, volume 37, 76775–76818. Curran Associates, Inc.

Tang, S.; Wang, X.; Chen, H.; Guan, C.; Wu, Z.; Tang, Y.; and Zhu, W. 2024. Post-Training Quantization with Progressive Calibration and Activation Relaxing for Text-to-Image Diffusion Models. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Umuroglu, Y.; Rasnayake, L.; and Själander, M. 2018. BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing. In *28th International Conference on Field Programmable Logic and Applications (FPL)*, 307–314.

Urbanek, J.; Bordes, F.; Astolfi, P.; Williamson, M.; Sharma, V.; and Romero-Soriano, A. 2024. A Picture is Worth More Than 77 Text Tokens: Evaluating CLIP-Style Models on Dense Captions. arXiv:2312.08578.

Wang, C.; Peng, H.-Y.; Liu, Y.-T.; Gu, J.; and Hu, S.-M. 2025a. Diffusion Models for 3D Generation: A Survey. *Comput. Vis. Media*, 11(1): 1–28.

Wang, C.; Wang, Z.; Xu, X.; Tang, Y.; Zhou, J.; and Lu, J. 2024. Towards Accurate Post-training Quantization for Diffusion Models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 16026–16035.

Wang, H.; Shang, Y.; Yuan, Z.; Wu, J.; Yan, J.; and Yan, Y. 2025b. QuEST: Low-bit Diffusion Model Quantization via Efficient Selective Finetuning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, 15542–15551.

Wang, K.; Liu, Z.; Lin, Y.; Lin, J.; and Han, S. 2019. HAQ: Hardware-Aware Automated Quantization with Mixed Precision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 8612–8620.

Wang, K.; Shi, M.; Zhou, Y.; Li, Z.; Yuan, Z.; Shang, Y.; Peng, X.; Zhang, H.; and You, Y. 2025c. A Closer Look at Time Steps is Worthy of Triple Speed-Up for Diffusion Model Training. *arXiv preprint arXiv:2405.17403*.

Whalen, L.; Du, Z.; You, H.; Li, C.; Li, S.; and Lin, Y. 2025. Early-Bird Diffusion: Investigating and Leveraging Timestep-Aware Early-Bird Tickets in Diffusion Models for Efficient Training. *arXiv preprint arXiv:2504.09606*.

Wu, J.; Wang, H.; Shang, Y.; Shah, M.; and Yan, Y. 2024. PTQ4DiT: Post-training Quantization for Diffusion Transformers. In Globerson, A.; Mackey, L.; Belgrave, D.; Fan, A.; Paquet, U.; Tomczak, J.; and Zhang, C., eds., *Advances in Neural Information Processing Systems*, volume 37, 62732–62755. Curran Associates, Inc.

Xiao, L.; Yang, H.; Dong, Z.; Keutzer, K.; Du, L.; and Zhang, S. 2023. Csq: Growing mixed-precision quantization scheme with bi-level continuous sparsification. In *2023 60th ACM/IEEE Design Automation Conference (DAC)*, 1–6. IEEE.

Xie, E.; Chen, J.; Chen, J.; Cai, H.; Tang, H.; Lin, Y.; Zhang, Z.; Li, M.; Zhu, L.; Lu, Y.; and Han, S. 2024. SANA: Efficient High-Resolution Image Synthesis with Linear Diffusion Transformers. *arXiv preprint arXiv:2410.10629*.

Yang, H.; Duan, L.; Chen, Y.; and Li, H. 2021. BSQ: Exploring bit-level sparsity for mixed-precision neural network quantization. *arXiv preprint arXiv:2102.10462*.

Yang, Y.; Dai, X.; Wang, J.; Zhang, P.; and Zhang, H. 2023. Efficient Quantization Strategies for Latent Diffusion Models. *arXiv preprint arXiv:2312.05431*.

Zhang, S.; Ding, Z.; Yang, K.; Wu, J.; Yan, X.; Li, X.; Duan, B.; Fang, J.; and Zhang, Y. 2026. AdaTSQ: Pushing the Pareto Frontier of Diffusion Transformers via Temporal-Sensitivity Quantization. arXiv:2602.09883.

Zhao, T.; Fang, T.; Huang, H.; Liu, E.; Wan, R.; Soedarmadji, W.; Li, S.; Lin, Z.; Dai, G.; Yan, S.; Yang, H.; Ning, X.; and Wang, Y. 2024a. ViDiT-Q: Efficient and Accurate Quantization of Diffusion Transformers for Image and Video Generation. *arXiv preprint arXiv:2406.02540*.

Zhao, T.; Ning, X.; Fang, T.; Liu, E.; Huang, G.; Lin, Z.; Yan, S.; Dai, G.; and Wang, Y. 2024b. MixDQ: Memory-Efficient Few-Step Text-to-Image Diffusion Models with Metric-Decoupled Mixed Precision Quantization. In *Proceedings of the European Conference on Computer Vision (ECCV)*.

Zheng, X.; Qin, H.; Ma, X.; Zhang, M.; Hao, H.; Wang, J.; Zhao, Z.; Guo, J.; and Liu, X. 2024. BinaryDM: Towards Accurate Binarization of Diffusion Model. *arXiv preprint arXiv:2404.05662*.

A Implementation Details

A.1 Quantization

Weight Quantization The quantization process is formally defined in Equation (11):

$$Q(w,n) = \text{clamp}\left(\left\lfloor \frac{w}{\alpha} + z \right\rfloor, 0, 2^n - 1\right), \quad (11)$$

where $\lfloor \cdot \rfloor$ denotes the floor operation. We use 2^n, rather than 2^n-1, in the scale so that the quantization grid aligns with the truncation boundaries:

$$\alpha = \frac{\max(w) - \min(w)}{2^n}, \qquad z = -\frac{\min(w)}{\alpha}. \quad (12)$$

The corresponding dequantization is

$$\hat{w} = \frac{2^n}{2^n - 1} \cdot \alpha \cdot (Q(w,n) - z). \quad (13)$$

Activation Quantization The quantization process is formally defined in Equation (14):

$$Q(x,n) = \text{clamp}\left(\left\lfloor \frac{x}{\alpha} + z \right\rceil, 0, 2^n - 1\right), \quad (14)$$

where $\lfloor \cdot \rceil$ denotes the rounding-to-nearest operation. We calculate the scaling factor α using the denominator $2^n - 1$ to fully utilize the representable range of the n-bit integer:

$$\alpha = \frac{\max(x) - \min(x)}{2^n - 1}, \qquad z = -\frac{\min(x)}{\alpha}. \quad (15)$$

The activation is dequantized as

$$\hat{x} = \alpha \cdot (Q(x,n) - z). \quad (16)$$

A.2 Teacher–Student Distillation

We sample training prompts from MS-COCO and precompute their conditional features c_t and unconditional features uc_t. The student can then be trained without keeping the teacher, VAE, and text encoder active in memory.

A.3 Overall Training Algorithm

The overall training algorithm is summarized in Algorithm 1.

Algorithm 1 Overall Training Algorithm of TASQ

Require: training data X, total diffusion timesteps T, number of temporal stages n, pruning interval I
Ensure: quantized model G

Initialize parameters F_1, F_2, $m_{t,l}$
Initialize regularization strength λ_M
Divide T into n stages: $T = \{T_1, T_2, \ldots, T_n\}$
Initialize mask training step $s = 0$, mask training interval $s_0 = I/n$
for iteration $= 0 \ldots i$ **do**
    Sample a timestep $t \sim \mathcal{U}(T)$ corresponding to current stage
    Compute $M_{t,l} = \mathcal{G}(s, m_{t,l})$
    LSB masked weight: $Q_{M,n}^{(t,l)} = Q_n - (1 - M_{t,l}) \cdot \text{LSB}$
    Teacher–Student Distillation
    Quantized model $\epsilon_q = F_q(x_t, t; Q_{M,n}^{(t,l)})$
    Full-precision teacher $\epsilon_{\text{fp}} = F_{\text{fp}}(x_t, t; W)$
    Distillation loss: $L_{\text{distill}} = \|\epsilon_q - \epsilon_{\text{fp}}\|^2$
    Total loss: $L_{\text{total}} = L_{\text{distill}} + \lambda_M \sum_{t,l} M_{t,l}$
    Update LoRA weights F_1, F_2 and mask parameter $m_{t,l}$ via gradient descent
    if iteration > 0 and iteration % $n == 0$ **then**
        $s = s + 1$
    end if
    if $s == s_0$ **then**
        if $m_{t,l} < 0$ **then**
            Prune corresponding LSB in layer l at timestep t
            reset $m_{t,l} = 0$
        end if
        reset $s = 0$
    end if
end for

B Robustness of the LSB Mask

Table 7 compares our Continuous LSB Mask (CLM) with the temperature-controlled sigmoid gate used in CSQ (Xiao et al. 2023). A standard sigmoid gate starts at 0.5, so it attenuates every LSB at the beginning of training. CLM instead starts at one, keeps the LSBs intact during the initial updates, and later anneals each gate toward zero or one. On PixArt-Σ W4A4, this change lowers FID from 21.4 to 16.1 on MJHQ and from 27.8 to 23.1 on sDCI.

C Training Details

All operating points share the same $b_{\max}$=8 buffer design and are compared at matched average effective bit-width (BitOPs), each fine-tuned for its setting; under a memory constraint, $b_{\max}$ is instead set to the affordable precision, giving storage identical to a native model at that width while TASQ still executes below it on average.

This section details the training configuration used for each pipeline evaluated above. We separate the description into three parts: (i) the optimization hyperparameters of the fine-tuning loop (Table 8), (ii) the calibration / training data budget per backbone (Table 9), and (iii) the resulting training cost in wall-clock time and peak memory (Table 5).

Optimization hyperparameters. Table 8 compares the four pipelines studied above along four axes: total iterations, LSB pruning interval (applicable only to TASQ-based pipelines), residual LoRA rank, and batch size. Pure QAT and pure TASQ are trained from the FP-initialized model for 5.0k iterations, whereas SVDQuant-initialized variants (SVDQuant+QAT, SVDQuant+TASQ) start from a calibrated SVDQuant checkpoint and only require 1.4k additional fine-tuning iterations. TASQ-based pipelines additionally maintain the temporal–spatial LSB mask, which is annealed and pruned every 0.1k iterations. The trainable residual LoRA branch is fixed at rank 32 across all variants; SVDQuant-initialized pipelines additionally carry a frozen SVD low-rank branch (rank 32 for the 4-bit setting, rank 16 for the 6-bit setting) inherited from the PTQ initialization.

Table 7: Mask formulation on PixArt-Σ. Initializing CLM at one avoids attenuating LSBs at the start of training.

Backbone	Model	(Avg) Precision	Mask Type	FID (↓)	IR (↑)	LPIPS (↓)	PSNR (↑)	FID (↓)	IR (↑)	LPIPS (↓)	PSNR (↑)
DiT	PixArt-Σ	INT W4A4	Sigmoid	21.4	0.863	0.372	15.9	27.8	0.882	0.398	14.6
	(20 Steps)	INT W4A4	CLM (Ours)	**16.1**	**0.932**	**0.282**	**18.6**	**23.1**	**0.966**	**0.263**	**18.8**

Table 8: Training-setting comparison across QAT, TASQ, SVDQuant+QAT, and SVDQuant+TASQ. "Pruning Int." is the LSB pruning interval, applicable only to TASQ-based pipelines. "Residual LoRA Rank" refers to the trainable LoRA branch; SVDQuant-initialized pipelines additionally carry a frozen SVD low-rank branch (rank 32 for 4-bit, rank 16 for 6-bit) inherited from the PTQ initialization.

Method	Iter.	Pruning Int.	Trainable LoRA Rank	Batch Size
QAT	5.0k	–	32	4
TASQ	5.0k	0.1k	32	4
SVDQuant + QAT	1.4k	–	32	4
SVDQuant + TASQ	1.4k	0.1k	32	4

Table 9: Calibration / training data budget for SVDQuant, QAT, and TASQ on each backbone. The pool size is $128 \times T$, where T is the number of denoising steps. All three methods share the identical pool to isolate the contribution of the optimization procedure from any difference in supervision volume.

Model	Steps (T)	SVDQuant	QAT	TASQ
PixArt-Σ	20	$128 \times 20 = 2{,}560$	$128 \times 20 = 2{,}560$	$128 \times 20 = 2{,}560$
SANA-1.6B	20	$128 \times 20 = 2{,}560$	$128 \times 20 = 2{,}560$	$128 \times 20 = 2{,}560$
SDXL-Turbo	4	$128 \times 4 = 512$	$128 \times 4 = 512$	$128 \times 4 = 512$

Calibration / training data. To ensure a controlled comparison, all three methods (SVDQuant, QAT, TASQ) draw from the same pool of $128 \times T$ (prompt, timestep) supervision pairs, where T is the number of denoising steps used by each backbone. Concretely, we sample 128 distinct text prompts from MS-COCO Captions and pre-compute the conditional and unconditional teacher features c_t and uc_t at every denoising timestep, yielding the cached pool described in Appendix A.2. SVDQuant uses this pool for scale calibration and the SVD decomposition of its low-rank branch; QAT and TASQ sample from it for cached teacher–student distillation. Table 9 summarizes the pool size for each backbone.

Training cost. Table 5 reports wall-clock time, peak memory, and the resulting FID for every pipeline at W4A8. Pure TASQ matches pure QAT exactly in both time and memory at the same 5.0k iterations, so the temporal–spatial masks add no measurable optimization cost over a non-temporal LoRA-QAT baseline. Starting from a calibrated SVDQuant checkpoint, TASQ needs 1.4k further iterations, which costs $+0.5$ hours on SANA-1.6B and $+2.1$ hours on SDXL-Turbo and improves FID from 17.7 to 15.9 and from 24.5 to 23.8 respectively. Peak memory during the SVDQuant-initialized adaptation is lower than SVDQuant's own calibration pass on SANA-1.6B (9.8 against 18.6 GB) because only the LoRA branch and the masks carry gradients.

D Interpretation of Timestep Sensitivity

D.1 Global Timestep Sensitivity

This section provides a mechanism-level interpretation of the timestep sensitivity observed in Figure 4. Following the denoising order used by TASQ, Stage 0 denotes the noisiest timesteps (large t), whereas the final stage corresponds to the cleanest timesteps. Under this convention, the theoretically derived sensitivity aligns almost perfectly with TASQ's learned bit allocation.

Error propagation in quantized DDIM. The DDIM update can be written as

$$x_{t-1} = \frac{\sqrt{\bar{\alpha}_{t-1}}}{\sqrt{\bar{\alpha}_t}} x_t + B_t\, \epsilon_\theta(x_t, t), \quad B_t = \sqrt{1-\bar{\alpha}_{t-1}} - \frac{\sqrt{\bar{\alpha}_{t-1}(1-\bar{\alpha}_t)}}{\sqrt{\bar{\alpha}_t}}. \tag{17}$$

When the denoiser is quantized as $\tilde{\epsilon}_\theta = \epsilon_\theta + \vartheta_t$, defining the cumulative deviation $\delta_t := \tilde{x}_t - x_t$ with $\delta_T = 0$, a first-order Taylor expansion yields the recursive error propagation

$$\delta_{t-1} = A_t\, \delta_t + B_t\, \vartheta_t, \qquad A_t = \frac{\sqrt{\bar{\alpha}_{t-1}}}{\sqrt{\bar{\alpha}_t}} I + B_t J_{x_t}, \tag{18}$$

where J_{x_t} denotes the Jacobian of $\epsilon_\theta(x_t, t)$ with respect to x_t.

Closed-form final output error. Unrolling Equation (18) from $\delta_T = 0$ to δ_0, following the closed-form DDIM error propagation analysis of (Liu et al. 2025a), gives

$$\delta_0 = \sum_{k=1}^{T} \underbrace{\left(\prod_{j=1}^{k-1} A_j \right) B_k}_{C_k}\, \vartheta_k. \tag{19}$$

Each coefficient C_k captures how the per-step quantization error ϑ_k injected at timestep k is amplified through the subsequent denoising trajectory before reaching the final output x_0.

Closed-form timestep sensitivity. Applying the approximation $J_{x_t} \approx 0$ used in prior analyses of quantized diffusion error propagation (Liu et al. 2025a), each A_t reduces to $\frac{\sqrt{\bar{\alpha}_{t-1}}}{\sqrt{\bar{\alpha}_t}} I$, and the product telescopes as

$$\prod_{j=1}^{k-1} A_j \approx \frac{\sqrt{\bar{\alpha}_0}}{\sqrt{\bar{\alpha}_{k-1}}} I \approx \frac{1}{\sqrt{\bar{\alpha}_{k-1}}} I. \tag{20}$$

This yields the following closed-form timestep sensitivity that depends only on the noise schedule:

$$\mathcal{S}(k) := \|C_k\| \approx \frac{|B_k|}{\sqrt{\bar{\alpha}_{k-1}}}. \tag{21}$$

Table 10: Stage-wise noise-schedule sensitivity and learned weight precision. Stage 0 contains the noisiest timesteps.

Model	Stage	Sensitivity	TASQ bits
PixArt-Σ	0 (noisy)	**0.71649**	**5.87**
	1	0.06710	3.99
	2	0.01131	3.13
	3 (clean)	0.00396	3.01
SDXL-Turbo	0 (noisy)	**0.08837**	**4.53**
	1	0.01734	4.05
	2	0.00537	3.71
	3 (clean)	0.00259	3.71

The noise schedule makes both factors larger near the noisy end: $\bar{\alpha}_{k-1}$ becomes smaller, while $|B_k|$ increases. An error introduced there is therefore amplified more strongly and propagates through more subsequent steps before reaching x_0. Errors introduced near the clean end have fewer steps in which to accumulate.

Verification against TASQ bit allocation. We evaluate $\mathcal{S}(k)$ on each model's noise schedule and average it within the same stages used by TASQ. Table 10 reports Pearson correlations of 0.97 for PixArt-Σ and 0.96 for SDXL-Turbo between sensitivity and learned precision. Stage 0 is the most sensitive stage in both models and receives the most bits.

D.2 Layer-wise Timestep Sensitivity

Figure 7 visualizes the activation dynamics of SANA-1.6B over 20 denoising steps, averaged over 10 prompts. Rows correspond to the four self-attention projections, Q, K, V, and O, while columns show five evenly spaced transformer blocks. Q, K, and V share the residual-branch input and therefore exhibit nearly identical trajectories. In contrast, the output projection O receives the post-attention hidden state and shows both larger activation magnitude and stronger step-to-step variation.

The activation trajectory also changes substantially with depth. In early blocks, the RMS tends to decay monotonically across denoising, whereas deeper blocks can rise sharply on the noisy side and peak around intermediate timesteps. This depth- and projection-conditioned non-stationarity motivates per-stage quantization, since a single static scale may fail to cover both ends of the activation distribution, especially in deeper output projections.

D.3 Learned Bit Allocation across Layers and Stages

Figure 8 shows the operating precision TASQ learns for every quantized weight tensor at every temporal stage, for all three backbones at the average W4A4 setting. The main paper reproduces the PixArt-Σ panel; the SANA-1.6B and SDXL-Turbo panels are given here.

Three properties are visible in all three models. First, Stage 0, the noisiest stage, is uniformly the darkest row: it retains the most least-significant planes, matching the closed-form schedule sensitivity of §D.1. Second, precision does not decay monotonically along the trajectory. In PixArt-Σ the lightest row is Stage 2 rather than Stage 3, and in SDXL-Turbo Stages 1 and 2 are close to each other, so the stage-wise averages in Table 10 hide non-monotonic structure that a hand-designed decay schedule would miss. Third, within every row the allocation varies strongly from tensor to tensor, and the high-precision tensors are not the same ones across stages. This is the component that a purely temporal schedule cannot express: it would have to assign one precision to an entire row, and therefore follow the upper envelope of that row rather than its actual distribution.

The number of weight indices differs per backbone (224 for PixArt-Σ, 160 for SANA-1.6B, 560 for SDXL-Turbo) because it counts the quantized linear and convolutional tensors of each architecture. Indices are ordered by network depth.

Table 11: Positioning against timestep-aware quantization: only TASQ adapts *weight* precision per denoising step (W), learns the allocation end-to-end, keeps a single shared weight buffer ($1\times$ store), and measures the resulting compute saving on a bit-serial accelerator. [§]single per-layer allocation; [†]per-layer + temporal distillation; [‡]activation *ranges*, not precision; "–" marks methods whose allocation is fixed across denoising steps, for which the per-step storage question does not arise.

	Per-step prec.				
Method	W	A	Learned	$1\times$ store	HW meas.
MPQ-Diff[§] (2024)	✗	✗	✗	–	✗
MPQ-DM[†] (2025)	✗	✗	✗	–	✗
TCAQ-DM[‡] (2025)	✗	✗	✗	✓	✗
AdaTSQ (2026)	✗	✓	✗	✓	✗
TASQ (ours)	✓	✗	✓	✓	✓

E Additional Comparison with Temporal-Awareness

Table 3 established that temporal and spatial adaptation are complementary, and Table 4 that the gain survives without SVDQuant initialization. This section adds the two comparisons that do not fit there: Table 11 places TASQ against timestep-aware quantizers along the axes that distinguish them, and Table 12 compares against timestep-aware baselines on CIFAR-10.

Table 12 compares TASQ, implemented on EfficientDM, with PTQ4DM, Q-Diffusion (Li et al. 2023), TFMQ-DM (Huang et al. 2024), and EfficientDM on a 100-step CIFAR-10 DDIM model. The baselines keep weight precision fixed across timesteps, whereas TASQ learns a timestep-dependent weight allocation. At A8, TASQ matches Effi-

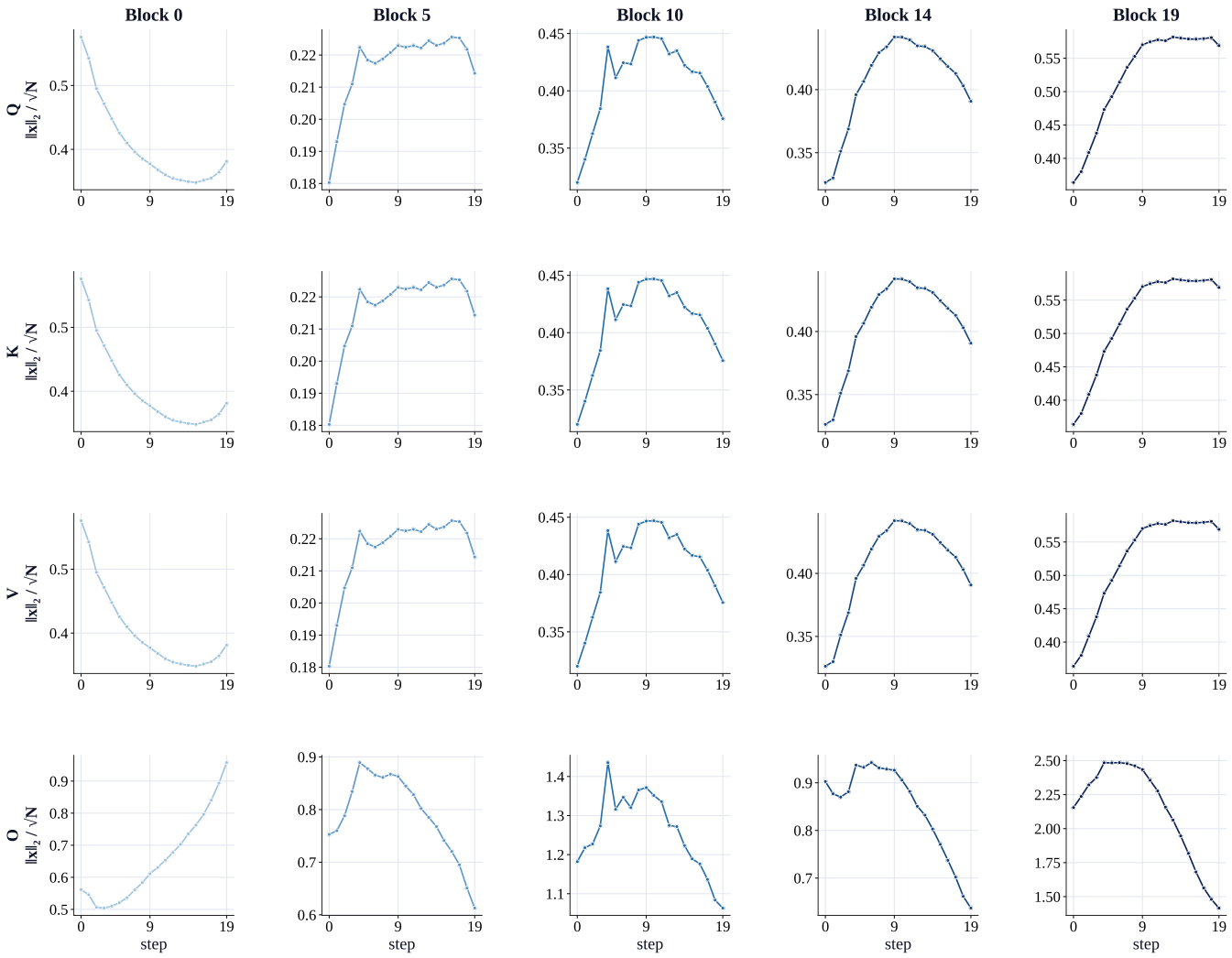


Figure 7: Per-projection × per-block input-activation RMS, $\|x\|_2/\sqrt{N}$, of SANA-1.6B across denoising steps.

cientDM's IS of 9.41 and lowers FID from 3.80 to 3.65. At A4, it raises IS from 9.37 to 9.40 and lowers FID from 3.91 to 3.90.

F Number of Stages

Figure 9 varies the number of stages on MJHQ. SDXL-Turbo improves as the stage count increases from one to four, matching its four denoising steps. PixArt-Σ also improves up to four stages, with little change beyond that point. We therefore use four stages in the remaining experiments.

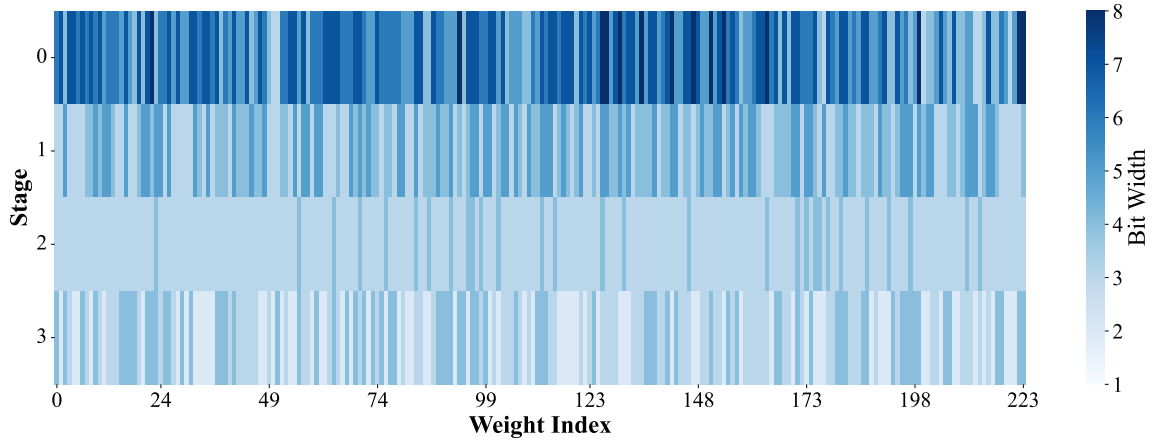
G Hardware Deployment Details

This section gives the deployment details behind the bit-serial measurements in §4.3: we give the bit-plane schedules the engine executes, describe how the operating precision maps to the native precision set of a target device, and discuss the restriction imposed by static-graph accelerators.

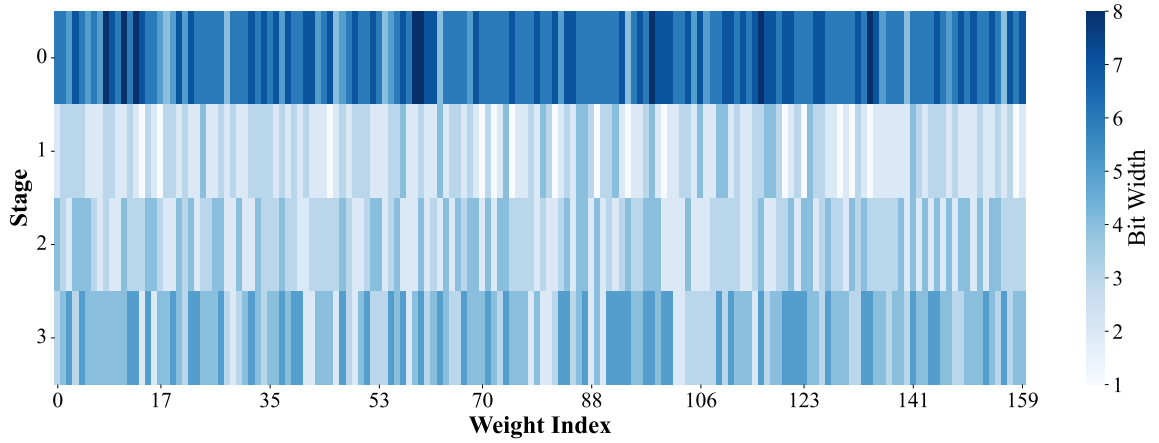
G.1 Hardware-Agnostic Design Principle

TASQ represents adaptive precision independently of a particular kernel. Each quantized layer stores one weight buffer at the maximum bit-width $b_{\max}$ found across its temporal stages, plus a small bit-allocation map of size $\mathcal{O}(L \cdot S)$ (number of layers × number of stages). For the configurations evaluated in this paper, this map adds less than 0.005% to the model's weight footprint. At inference time, switching the effective precision of a layer is implemented by reading fewer least-significant planes from the same buffer. It does not require model reloading, kernel re-launch, or a runtime precision search.

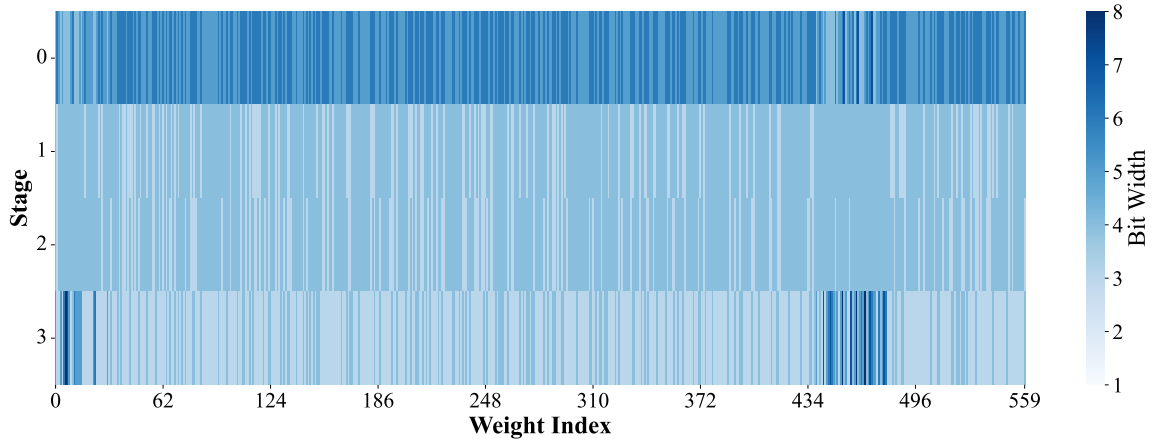
The compiled kernel handles every stage and takes the number of bit planes as an argument; the measured switch itself costs no cycles. Deployment then depends on the precision set supported by the target hardware. The next section describes how we align TASQ's operating precisions with that set.



(a) PixArt-Σ (0.6B, DiT, 20 steps)



(b) SANA-1.6B (DiT, 20 steps)



(c) SDXL-Turbo (2.6B, U-Net, 4 steps)

Figure 8: TASQ-learned operating weight precision across temporal stages (rows) and weight tensors (columns) at the average W4A4 setting. Darker is higher precision. Stage 0 is the noisiest stage. All three backbones keep the most planes at Stage 0, but the allocation within each stage varies strongly across tensors, and the per-stage ordering is not monotonic in the denoising direction.

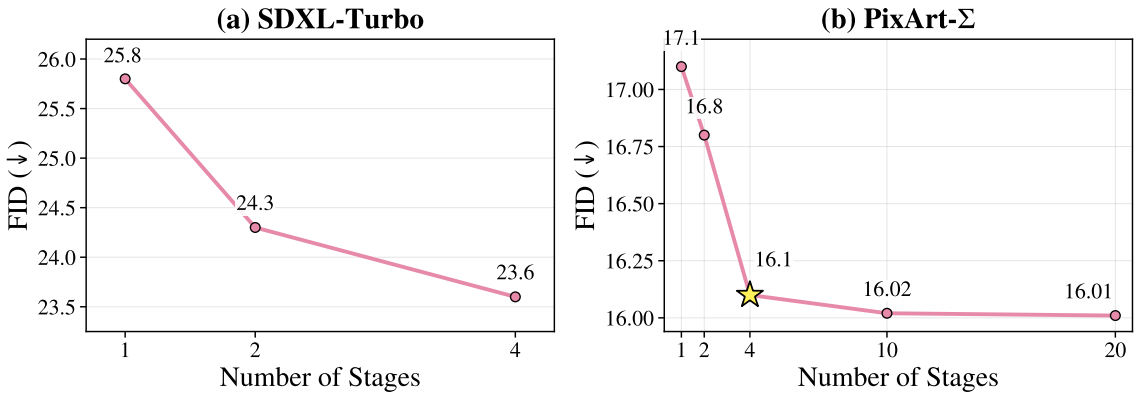


Figure 9: Ablation on the number of temporal stages on MJHQ for (a) SDXL-Turbo and (b) PixArt-Σ. For SDXL-Turbo, performance improves steadily up to four stages, matching its four denoising steps. For PixArt-Σ, four stages form the elbow point, after which the gains become marginal.

Table 12: CIFAR-10 results for a 100-step DDIM model. TASQ matches the best IS and gives the lowest FID at both activation precisions.

Method	A-bit	W-bit	IS (↑)	FID (↓)
Full Precision	32	32	9.12	4.14
PTQ4DM	8	4.00	9.31	10.12
Q-Diffusion		4.00	9.12	4.93
TFMQ-DM		4.00	9.13	4.78
EfficientDM		4.00	**9.41**	3.80
TASQ		3.91	**9.41**	**3.65**
PTQ4DM	4	4.00	0.45	375.12
Q-Diffusion		4.00	0.71	384.21
TFMQ-DM		4.00	3.19	236.63
EfficientDM		4.00	9.37	3.91
TASQ		3.92	**9.40**	**3.90**

G.2 Adapting to Native Hardware Instructions

Given a target device with a set of natively supported integer precisions $\mathcal{B}_{\text{native}} = \{b_1, b_2, \dots\}$, the user selects $b_{\max} = \max \mathcal{B}_{\text{native}}$ as the storage precision and trains TASQ to operate over $\mathcal{B}_{\text{native}}$. After training, every weight is physically stored at $b_{\max}$ as a single $Q_{b_{\max}}$ buffer (Eq. (1)), and per-stage operating precision is realized at inference time by truncating its least significant bits.

Generalizing the one-bit right-shift in Eq. (1) to a k-bit right-shift, the b_j-bit operand obtained from the stored b_i-bit code ($b_i > b_j$) is

$$Q_{b_j} = \left\lfloor \frac{Q_{b_i}}{2^{\,k_{i\to j}}} \right\rfloor, \qquad k_{i\to j} = b_i - b_j, \tag{22}$$

Algorithm 2 Bit-plane schedules for one GEMM tile with L weight and R activation planes (plane-fetch counts in comments). The Temporal-Precision Engine is the activation-stationary schedule plus per-stage precision control; activations are re-fetched for every tile.

Naive bit-plane loop $2LR$ fetches
for $i = 1$ **to** L **do**
  for $j = 1$ **to** R **do**
    fetch W_i; fetch A_j; Exec$(W_i, A_j,\ \text{shift}{=}i{+}j)$
  end for
end for

Weight-stationary $L + LR$ fetches
for $i = 1$ **to** L **do**
  fetch W_i ▷ held resident
  for $j = 1$ **to** R **do**
    fetch A_j; Exec$(W_i, A_j,\ i{+}j)$
  end for
end for

Temporal-Precision Engine (ours) $L_s + R$ fetches
for stage $s = 1$ **to** S **do**
  $L_s \leftarrow$ per-stage weight precision ▷ one register write, 0 cycles
  for $j = 1$ **to** R **do**
    fetch A_j ▷ resident for the whole tile
  end for
  for $i = 1$ **to** L_s **do**
    fetch W_i ▷ streamed once
    for $j = 1$ **to** R **do**
      Exec$(W_i, A_j,\ i{+}j)$
    end for
  end for
end for

so that $k_{i\to j}$ corresponds to the number of LSB planes dropped from the stored buffer. The discarded $k_{i\to j}$-bit LSB block, generalizing the single-LSB extraction in Eq. (3), is

$$\mathrm{LSB}^{(k_{i\to j})} \;=\; Q_{b_i} \;-\; 2^{\,k_{i\to j}} \cdot Q_{b_j} \;=\; \sum_{p=0}^{k_{i\to j}-1} 2^{\,p}\, Q_{b_i}[p], \tag{23}$$

where $Q_{b_i}[p] \in \{0,1\}$ denotes the p-th bit-plane of Q_{b_i} (least-significant first). At inference, the trained mask $M_{t,l}$ has annealed to a binary indicator that determines, per stage and per layer, which planes are kept; the operating precision is then $b_j = b_{\max} - k_{i\to j}$, and Eq. (22) is realized by reading only the top b_j planes of the stored buffer. No re-quantization, model swap, or per-stage checkpoint is needed.

Importantly, the same trained TASQ checkpoint can be re-deployed across backends with different native precision sets simply by selecting a different operating precision subset — the stored weight buffer is unchanged. This decouples training from any specific deployment target, which is the property that motivates our hardware-agnostic claim.

G.3 Static-Graph Compilation Targets

A class of mobile and edge accelerators, including the Apple Neural Engine through Core ML, Qualcomm Hexagon, and Edge TPU, relies on ahead-of-time compilation of a static computation graph in which the precision of each layer is fixed at compile time. On these targets, the runtime cannot vary the number of bit-planes consumed by a kernel call, because the graph compiler has already lowered each op to a fixed-precision tensor primitive. The same restriction applies to any method that changes layer precision at runtime. Targets that dispatch work at run time rather than lowering it ahead of time do not impose this constraint, since the number of bit-planes consumed by a call can then be chosen per stage.

H Limitations and Future Directions

Commodity-hardware support. The learned precision schedule can be evaluated at a matched operation budget on any platform, but cycle savings require hardware whose cost changes with operand precision (Umuroglu, Rasnayake, and Själander 2018; Askarihemmat et al. 2023). Datapaths whose lowest supported precision is four bits do not expose sub-4-bit execution, so a 2- or 3-bit TASQ layer still pays the four-bit cost. Supporting TASQ on emerging FP4 and microscaling formats will require production kernels that expose their finer precision choices.

TASQ targets the cost of each denoising step and can be used with faster samplers or distilled models; SDXL-Turbo shows that stage-wise allocation still helps on a four-step schedule. We leave prompt-conditioned masks and joint weight–activation precision to future work. Appendix G.3 discusses runtime support.

I Additional Image Quality Results

Figures 10 and 11 provide additional comparisons with SVDQuant. In these examples, TASQ retains more of the objects and attributes present in the full-precision output, although both quantized models can differ visibly from that reference.

Figure 10: Qualitative Image Generation Results on SANA-1.6B

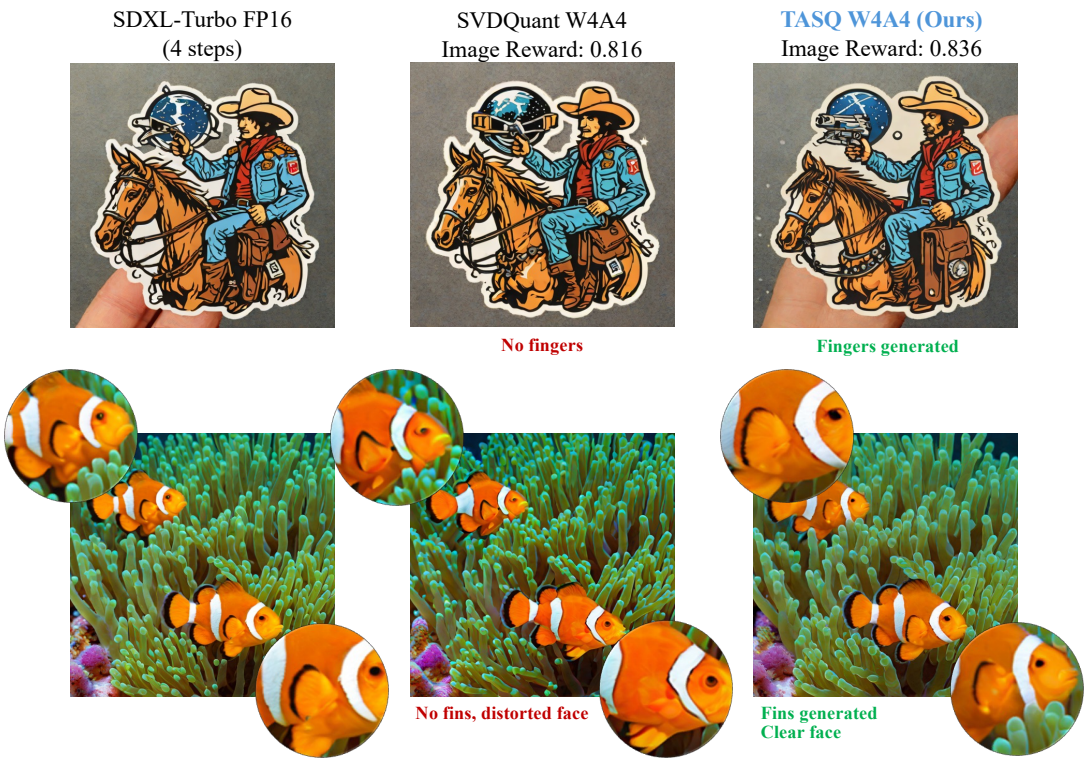


Figure 11: Qualitative Image Generation Results on SDXL-Turbo and PixArt-Σ