KroQuant: Kronecker-Structured Block Transforms for Efficient Post-Training Quantization of Diffusion Transformers

Yann Bouquet[*,†] **Alireza Khodamoradi**[†] **Kristof Denolf**[†] **Mathieu Salzmann**[*]

Abstract

Post-training quantization (PTQ) of diffusion transformers (DiTs) to W4A4 severely degrades output quality, because activations entering each linear layer contain outliers that 4-bit formats cannot represent. The standard fix applies an invertible linear transform to the activations and its inverse to the weights before quantizing both. Normalization layers between blocks force this transform to run online at every denoising step, making its inference computation cost the binding design constraint. Existing options trade quantization quality for inference cost: per-channel scaling (SmoothQuant) is computationally cheap but impacts the magnitude of the channels, which can harm quantization accuracy; fixed Hadamard transforms yield better quantization accuracy but require large block sizes that incur a high online cost; learned full-d invertible transforms calibrate best but entail a prohibitive dense $d \times d$ matrix multiplication (GEMM) per layer per step.

We propose KroQuant, a PTQ method that applies a learned Kronecker-structured invertible transform to each 32-element block of the activation, storing less than half the parameters of per-channel scaling. Offline LoRaQ weight calibration then absorbs the residual weight quantization error. On PixArt-Σ, SANA, and FLUX.1-schnell at W4A4 (MXFP4e2), KroQuant matches or improves on SVDQuant and LoRaQ for every metric on MJHQ-30K and sDCI. The block-local structure runs as small matrix-core GEMMs or as a five-stage butterfly operation, leveraging the properties of the Kronecker operator. We show on an MI350X GPU that the KroQuant quantizer kernel runs close to parity with the SmoothQuant kernel.

1 Introduction

Post-training quantization (PTQ) to 4-bit precision is the natural way to deploy billion-parameter diffusion transformers (DiTs), and is supported natively in current accelerators via the OCP Microscaling standard [Microsoft et al., 2023]. The bottleneck is activation outliers [Xiao et al., 2023]: a small subset of channels carries magnitudes orders of magnitude larger than the rest, so a 4-bit blockwise quantizer wastes most of its dynamic range on those channels and clips everything else. Consider a linear layer $O = AW^\top$ with $A \in \mathbb{R}^{m\times d}$, $W \in \mathbb{R}^{o\times d}$, and a b-bit quantizer Q_b. The main solution in both LLM and DiT quantizers is to insert an invertible activation transform $T \in \mathrm{GL}(d)$ that redistributes channel magnitudes before quantization, acting as

$$A \mapsto AT, \qquad W \mapsto WT^{-T}, \qquad AT \cdot (WT^{-\top})^\top = AT \cdot T^{-1}W^\top = AW^\top, \tag{1}$$

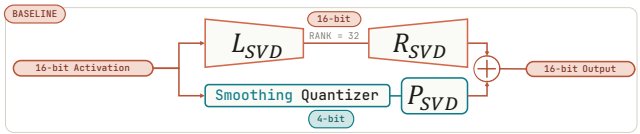
which leaves the linear map invariant while reducing the per-layer quantization error

$$\mathcal{E}(T) = Q_b(AT)\, Q_b(WT^{-\top})^\top - AW^\top. \tag{2}$$

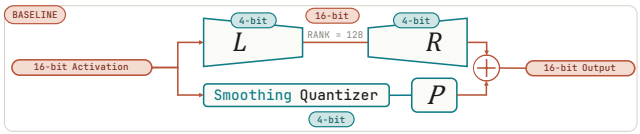
[*]EPFL, Lausanne, Switzerland. firstname.lastname@epfl.ch
[†]Advanced Micro Devices, Inc., Longmont, CO, USA. firstname.lastname@amd.com

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

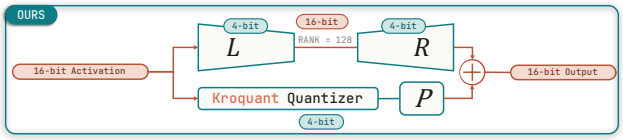
Figure 1: Pipeline comparison for W4A4 quantization of a linear layer in a DiT block. (a) SVDQuant splits the weight into a quantized residual and an SVD-truncated FP16 low-rank branch. (b) LoRaQ replaces the SVD truncation by a data-free low-rank branch that can itself be quantized below 16 bits. (c) KroQuant additionally inserts a learned block-diagonal activation transform T before quantization, complementing LoRaQ's weight-side correction. P and P_{SVD} are the 4-bit residual matrix defined as a function of the low rank components.



(a) SVDQuant.



(b) LoRaQ.



(c) KroQuant.

The compensating T^{-1} is always folded into the layer's stored weights; the *absorption trick* further folds T into the previous layer's output, so that the activation arriving at the current layer is already T-transformed and the runtime transform is free. As such, existing methods differ only in the structure of T: diagonal per-channel scaling [Xiao et al., 2023, Zhao et al., 2025, Wu et al., 2024], dense rotations [Ashkboos et al., 2025, Tseng et al., 2024, Liu et al., 2025], or general invertible matrices [Sun et al., 2024].

While generally applicable to LLMs, the absorption trick fails in DiTs, especially between transformer blocks. This is because folding T into the previous layer's output requires the normalization between blocks to commute with T:

$$\mathrm{Norm}(xT)\,T^{-1} = \mathrm{Norm}(x). \tag{3}$$

This is not satisfied by the LayerNorm and AdaLN variants sitting between every DiT block (Appendix C). Every transform a DiT quantizer applies therefore runs **online**, at every linear layer of every denoising step, and per-step cost becomes the binding design constraint. While the per-step cost is affordable with per-channel scaling transforms that entail diagonal matrices, this strategy offers only limited expressivity and thus quantization accuracy. This trade-off is taken by recent weight-splitting PTQ pipelines such as SVDQuant [Li et al., 2025] and LoRaQ [Bouquet et al., 2026], which decompose each weight into a low-bit residual and a low-rank correction branch.

By contrast, existing non-diagonal transforms yield better quantization but fall short under the online constraint. A Hadamard transform is a fixed orthogonal and symmetric matrix with normalized ± 1 entries that mixes channels via add/subtract butterflies, spreading outlier magnitudes uniformly across the block [Ashkboos et al., 2025]. At block size 32, it is cheap (a single MXFP4-aligned kernel pass) but suppresses outliers only within each 32-channel sub-vector, and eventually ends up underperforming per-channel scaling [Xiao et al., 2023]. Learned $d \times d$ invertible matrices [Liu et al., 2025, Sun et al., 2024] are expressive but store $O(d^2)$ parameters per layer and require a dense $m \times d \times d$ matrix multiplication every step, infeasible at inference. Altogether, there is therefore a need for a transform that is **learned** and 32×32 **block-diagonal**; that is, more expressive than a fixed Hadamard at the same block size, yet online-cheap and compatible with an MX quantizer.

To address this need, we propose **KroQuant**[‡], a learned T that is block-diagonal with 32×32 blocks, one per MX quantization block. Each block is a Kronecker product of five 2×2 unit-determinant factors (Section 3.2). This construction is a five-stage butterfly, the structure of the fast Hadamard transform, in which each stage mixes pairs of channels with a single learned 2×2 factor. The structure leaves two ways to execute the transform on a GPU: materializing the dense block feeds the matrix cores, while executing the butterfly stage by stage never forms the block and needs 10 instead of 32 multiply-accumulates per element, but runs on the vector units. We show that KroQuant acts as a drop-in replacement for SmoothQuant [Xiao et al., 2023] in the activation-transform stage of weight-splitting PTQ pipelines such as LoRaQ (Figure 1). On PixArt-Σ [Chen et al., 2024a], SANA [Xie et al., 2025], and FLUX.1-schnell [Black-Forest-Labs, 2024], with LoRaQ in its W4A4

[‡]Code: `https://github.com/AMDResearch/KroQuant`

configuration, LoRaQ-with-KroQuant matches or beats LoRaQ-with-SmoothQuant [Bouquet et al., 2026, Xiao et al., 2023] and the SVDQuant reference [Li et al., 2025] on every metric. On an AMD MI350X GPU, both the materialized kernels and the butterfly kernel run close to parity with per-channel scaling, the latter through an implementation specific to this architecture (Section 4.4).

Concretely, we contribute:

- **A learned block-diagonal activation transform compatible with online constraints.** The first non-diagonal online transform for DiT PTQ that is more expressive than a fixed block-Hadamard and cheaper at inference than full-d rotations [Liu et al., 2025] or invertibles [Sun et al., 2024].
- **A Kronecker-LU butterfly parametrization of 32×32 blocks.** Five 2×2 unit-determinant LU factors form a learnable relaxation of the block Hadamard, which serves as initialization, with a closed-form inverse; the resulting 15 parameters per block, $68\times$ fewer than a dense $\mathrm{GL}(32)$ block, let the block be rebuilt inside the quantizer kernel.
- **An algorithm close to parity in speed with SmoothQuant.** On an MI350X GPU, kernels that either feed the materialized block to matrix cores or leverage the five-stage butterfly run close to parity with per-channel scaling.

2 Related Work

PTQ for diffusion transformers. DiT quantization has progressed from 8-bit schemes [Shang et al., 2023] to sub-8-bit methods that adapt SmoothQuant [Xiao et al., 2023] to the timestep-conditioned activation distributions induced by AdaLN. PTQ4DiT [Wu et al., 2024] and ViDiT-Q [Zhao et al., 2025] apply per-channel diagonal scales with timestep-dependent calibration; Q-DiT [Chen et al., 2024b] and MixDQ [Zhao et al., 2024] push further with mixed-precision and sensitivity-driven allocations. All keep the activation transform T diagonal, and all degrade at W4A4, where the residual error is dominated by inter-channel correlations that no per-channel rescaling can remove. Timestep awareness and the structure of T are orthogonal axes: PTQ4DiT and ViDiT-Q account for how activation statistics vary across denoising timesteps when calibrating their per-channel scales, whereas KroQuant changes the shape of the transform and samples its calibration activations across timesteps without weighting them. The two are complementary.

The design space of activation transforms. PTQ methods that go beyond per-channel scaling differ in the structure of the invertible transform $T \in \mathrm{GL}(d)$ (Table 8 in Appendix D). SmoothQuant's diagonal T rescales channels independently; QuaRot [Ashkboos et al., 2025] and QuIP# [Tseng et al., 2024] use fixed Hadamard rotations, whose outlier-suppression strength grows with block size, applied as a fast transform of $\log_2 d$ butterfly stages; SpinQuant [Liu et al., 2025] learns a dense rotation on the Stiefel manifold, at the cost of a $d \times d$ parameter matrix and of a dense GEMM per layer whenever it runs online. FlatQuant [Sun et al., 2024] uses the Kronecker product as an approximation device: it writes a dense T as the product of two factors of size about $\sqrt{d}$ that span the full hidden dimension and applies them as two chained matrix products, never forming T. In LLMs, rotations can be absorbed offline into adjacent weights through RMSNorm; in DiTs they cannot, as discussed in Section 1, so per-step cost becomes load-bearing. KroQuant takes the other end of the Kronecker construction: five 2×2 factors on each 32×32 MX block, the maximal depth at which the factorization terminates. At that depth each block is a butterfly with one learned factor per stage, that is, a learnable relaxation of the block Hadamard; it costs fewer parameters per layer than SmoothQuant's diagonal scales, and its inverse is closed-form.

Low-rank weight splitting and the role of the activation transform. A complementary line of work splits the linear-layer weight W into a low-rank branch and a 4-bit residual. SVDQuant [Li et al., 2025] smooths outliers and SVD-decomposes the smoothed weight with data-dependent low-rank calibration; LoRaQ [Bouquet et al., 2026] instead fits the low-rank branch data-free to the residual quantization error, allowing the low-rank branch itself to be quantized below 16 bits. Both methods split the weight but quantize the activation as-is, so they require an online activation transform with no normalization to fuse against, and both default to a SmoothQuant-style diagonal T. KroQuant is a drop-in replacement for that transform: we adopt LoRaQ's weight correction unchanged, in its W4A4 configuration (LoRaQ-A in Bouquet et al. [2026]), and substitute its SmoothQuant front-end with our learned block-diagonal T, yielding the headline LoRaQ-with-KroQuant comparison against LoRaQ-with-SmoothQuant, with SVDQuant as an additional reference.

3 Method

We work within the formal setup of Section 1 and instantiate the blockwise quantizer Q_b left abstract there as follows.

3.1 Quantization setup

We denote by Q_b (Section 1) the blockwise quantize-then-dequantize operator at bit width b, taking a real matrix and returning a matrix of the same shape: each row of the input is partitioned into non-overlapping blocks of size n; each block is encoded as a shared scale s (the set of representable scales, e.g., FP8) and a n vector $\mathbf{q}$ in the chosen b-bit format (FP4 when $b = 4$); the dequantized blocks are reassembled into the output.

Goal. We want to construct a transform $T \in \mathrm{GL}(d)$ that is block-diagonal,

$$T = \mathrm{diag}(K^{(1)}, K^{(2)}, \ldots, K^{(\lceil d/n \rceil)}), \tag{4}$$

where each diagonal block $K^{(j)} \in \mathrm{GL}(32)$ is learnable (unlike a fixed Hadamard) and parametrized compactly enough to be rebuilt inside the quantizer kernel. The construction of Section 3.2 uses 15 parameters per block, fewer than the 32 of per-channel scaling and $68\times$ fewer than a dense $\mathrm{GL}(32)$ block. We aim to leave the linear map invariant ($AT \cdot (WT^{-\top})^\top = AW^\top$) while reducing $\|\mathcal{E}\|_F$.

3.2 Kronecker-structured invertible block transforms

This structure of T couples features within each group of n channels while leaving groups independent, matching the block structure of the MXFP4e2 quantizer Q_4. We set $n = 32$, the MX block size, so that the transform couples exactly the channels that share a quantization scale; this is a design choice tied to the format rather than a limit of the construction.

Kronecker product construction. We parametrize each $n \times n$ block as a Kronecker product of $\log_2 n$ small matrices:

$$K_n = G_1 \otimes G_2 \otimes \cdots \otimes G_{\log_2 n}, \qquad G_k \in \mathbb{R}^{2\times 2},\ k = 1, \ldots, \log_2 n. \tag{5}$$

For $n = 32$ this gives $\log_2 32 = 5$ factors. Since the Kronecker product of invertible matrices is invertible, and since $(G_1 \otimes \cdots \otimes G_5)^{-1} = G_1^{-1} \otimes \cdots \otimes G_5^{-1}$, inversion reduces to five independent 2×2 inversions.

Butterfly structure. By the mixed-product property, $G_1 \otimes \cdots \otimes G_5 = \prod_{k=1}^{5} \big(I_{2^{k-1}} \otimes G_k \otimes I_{2^{5-k}}\big)$, where the k-th factor applies the same G_k to the 16 pairs of channels whose indices differ only in bit $5 - k$. Each block is therefore a five-stage butterfly, the structure of the fast Walsh–Hadamard transform, with one 2×2 factor shared by all pairs of a stage. Applied stage by stage, it costs $2 \log_2 n = 10$ multiply-accumulates per element instead of $n = 32$ for the dense block, and K_n is never formed. The family is generated by five stages of free 2×2 mixing, one per binary axis of the channel index: it captures couplings aligned with this binary decomposition and cannot express couplings that require permuting channels first; Appendix E explains why we do not use a free orthogonal 32×32 block instead.

Unit-determinant LU parametrization. To prevent degenerate scaling solutions, where the transform collapses one operand toward zero while inflating the other, we enforce $|\det G_k| = 1$ for every factor. We parametrize each 2×2 matrix via an LU decomposition with fixed unit determinant:

$$G_k = \underbrace{\begin{pmatrix} 1 & 0 \\ c_k & 1 \end{pmatrix}}_{L_k} \underbrace{\begin{pmatrix} a_k & b_k \\ 0 & -1/a_k \end{pmatrix}}_{U_k}, \qquad a_k \neq 0. \tag{6}$$

One can verify that $\det G_k = a_k \cdot (-1/a_k) = -1$, so $|\det G_k| = 1$ is satisfied for any $a_k \neq 0$. The upper-triangular factor U_k uses a negative reciprocal on the $(2, 2)$ entry so that the parametrization includes, as special cases, the orthogonal reflections with a nonzero $(1, 1)$ entry, such as the normalized 2×2 Hadamard factor; since $\det G_k = -1$, no factor is a rotation. Each G_k is governed by three real parameters (a_k, b_k, c_k); with five factors per block this gives **15 parameters per 32×32 block**, compared to $32^2 = 1024$ for an unconstrained invertible matrix and 1 for a scalar channel-wise scale.

Hadamard initialization. The normalized Hadamard matrix H_n ($n = 2^k$) is a Kronecker product of k copies of the 2×2 factor $\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$. Each 2×2 factor has determinant -1 and $|\det H_n| = 1$, so H_n belongs to our family. We initialize every G_k as $G_k^{(0)} = \begin{pmatrix}\cos\left(\frac{\pi}{4}\right) & \sin\left(\frac{\pi}{4}\right)\\ \sin\left(\frac{\pi}{4}\right) & -\cos\left(\frac{\pi}{4}\right)\end{pmatrix}$ which is equivalent to initializing T to a block-diagonal Hadamard rotation. This is known empirically as a good initialization strategy to reduce activation outliers and lower quantization error before any gradient-based optimization [Ashkboos et al., 2025, Liu et al., 2025].

3.3 Layer-wise transform optimization

Objective. Given a small calibration dataset providing N activation samples $\{A^{(s)}\}_{s=1}^{N}$, we optimize T independently for each linear layer by minimizing the output mean squared error:

$$\mathcal{L}_{\text{out}}(T) = \frac{1}{Nmo}\left\|Q_b(AT)\,Q_b(WT^{-T})^\top - AW^\top\right\|_F^2. \tag{7}$$

Regularization. Minimizing $\mathcal{L}_{\text{out}}$ alone can converge to solutions where one operand is well-quantized at the expense of the other — for example, a transform that whitens the activations perfectly while leaving the weights in a highly non-uniform representation. To encourage both operands to be individually well-quantized, we add two auxiliary terms:

$$\mathcal{L}_A(T) = \frac{1}{Nmd}\left\|Q_b(AT) - AT\right\|_F^2, \qquad \mathcal{L}_W(T) = \frac{1}{od}\left\|Q_b(WT^{-\top}) - WT^{-\top}\right\|_F^2. \tag{8}$$

The total loss is

$$\mathcal{L}(T) = \mathcal{L}_{\text{out}}(T) + \lambda_A\,\mathcal{L}_A(T) + \lambda_W\,\mathcal{L}_W(T). \tag{9}$$

Automatic scaling of λ. The regularization weights λ_A and λ_W are set so that each auxiliary term matches the magnitude of $\mathcal{L}_{\text{out}}$ at initialization:

$$\lambda_A = \frac{\mathcal{L}_{\text{out}}(T_0)}{\mathcal{L}_A(T_0)}, \qquad \lambda_W = \frac{\mathcal{L}_{\text{out}}(T_0)}{\mathcal{L}_W(T_0)}, \tag{10}$$

where T_0 is the Hadamard initialization. This normalization ensures that none of the three terms dominates early in training, preventing the regularizers from overriding the primary output-error signal during the initial phase of optimization, while still providing sufficient gradient signal to avoid degenerate solutions.

Optimization. We optimize $\mathcal{L}(T)$ with respect to the $15\cdot\lceil d/n\rceil$ free parameters of T using Adam. The quantizer Q_b is non-differentiable; we use a straight-through estimator (STE) to propagate gradients through the quantization operations. Since the transforms of the different layers are independent, they can be optimized in parallel or sequentially with negligible overhead relative to the calibration forward passes. We keep the optimization layer-wise. The LU parametrization has a pole at $a_k = 0$: the $(2,2)$ entry of U_k diverges as $1/a_k$ and its gradient as $1/a_k^2$. In our layer-wise runs the optimization stays away from the pole. Optimizing several layers jointly composes the five factors within each layer and the layers across depth, all through the STE; in our experiments this produced loss oscillations between epochs, no sustained descent, and divergence at larger step sizes, even with a bounded reparametrization that removes the pole (Appendix H).

3.4 Final correction: offline LoRaQ weight calibration

After the per-layer Kronecker transforms $\{T^{(\ell)}\}$ to optimize our objective, we apply the offline weight calibration procedure from LoRaQ [Bouquet et al., 2026] to the transformed weights $\widetilde{W}^{(\ell)} = W^{(\ell)}(T^{(\ell)})^{-\top}$.

Bouquet et al. [2026] decomposes each transformed weight $\widetilde{W}^{(\ell)}$ into a b-bit quantized residual branch and an additive low-rank correction branch $L^{(\ell)}R^{(\ell)}$, with $L^{(\ell)} \in \mathbb{R}^{o\times r}$ and $R^{(\ell)} \in \mathbb{R}^{r\times d}$, and optimizes both branches data-free to carry most information of the 16-bit activation to the low-rank branch, reducing the linear layer's overall quantization error further. The low-rank branch can itself be quantized below 16 bits without breaking the decomposition. At inference, the layer evaluates the

quantized residual matrix multiplication and the mixed-precision low-rank correction in parallel and sums their outputs.

At inference time the computation for layer ℓ is

$$\hat{O}^{(\ell)} = Q_b\Big(A^{(\ell)}T^{(\ell)}\Big) \times Q_b\Big(\widetilde{W}^{(\ell)} - L^{(\ell)}R^{(\ell)}\Big) + A^{(\ell)}T^{(\ell)}L^{(\ell)}R^{(\ell)}, \tag{11}$$

with $\hat{W}^{(\ell)}$ precomputed offline.

3.5 Hardware-efficient transform application

Applying a 32×32 block costs 32 multiply-accumulates per element as a dense matrix and $2\log_2 32 = 10$ as the butterfly of Section 3.2. Matrix cores offer far higher throughput than the vector units but consume dense operands, so they can absorb the larger dense cost only if the block is materialized; the butterfly never materializes the block but runs on the vector units. We implement the dense strategy in Triton and the butterfly in its Gluon dialect [Tillet et al., 2019]; all kernels fuse the MXFP4e2 output quantizer and apply each block $K^{(j)}$ to the corresponding 32-element slice of every token. The dense kernels tile the activation matrix $A \in \mathbb{R}^{m\times d}$ on a 2D grid of shape $(\lceil m/B_R\rceil,\ d/32)$, where $B_R \in \{16, 32, 64, 128\}$ is autotuned, and each program processes a tile of shape $(B_R, 32)$.

3.5.1 Kernel variants

K1 (precomputed dense block). Each 32×32 block $K^{(j)}$ is materialized once when the model is loaded and stored in HBM, so K1 pays parameter traffic but no construction. The kernel loads the activation tile $A_{\text{tile}} \in \mathbb{R}^{B_R\times32}$ and the corresponding $K^{(j)} \in \mathbb{R}^{32\times32}$, computes a single `tl.dot(`$A_{\texttt{tile}}$`,` $K_{\texttt{block}}$`)` accumulating in fp32, and stores the result after casting to the input dtype. Storage cost: 1024 values (2 KB) per column block; 256 KB total for $d = 4096$.

K2 (on-the-fly Kronecker construction). Instead of loading a dense matrix, K2 reconstructs $K^{(j)}$ from the 15 scalar parameters of the five 2×2 factors $G_1, \ldots, G_5$ (Section 3.2). Since $32 = 2^5$, every row and column index of $K^{(j)}$ is a 5-bit integer, and the Kronecker product entry-wise formula reads

$$K[r, c] = \prod_{k=1}^{5} G_k\big[r_{5-k},\, c_{5-k}\big], \tag{12}$$

where r_p and c_p denote the p-th bit of r and c respectively. We measure two ways of paying for this construction. The *in-register* implementation applies the formula as a streaming product over the five factors inside the kernel that consumes the block, materializing K factor by factor in registers via bit-indexed scalar lookups (pseudocode in Appendix B), so construction is repeated in every program. The *split* implementation builds each block once in a separate kernel, writes it to HBM, and lets the consuming kernel load it, at the cost of one extra launch and an HBM round trip; both kernels are timed. We report the faster of the two per shape. Storage cost: 15 scalar parameters per column block, $68\times$ fewer than K1.

K3 (butterfly). K3 applies the five factors as butterfly stages and never forms $K^{(j)}$. Each thread holds one 32-channel block of a token, so all stages and the shared-scale reduction of the quantizer stay in its registers, without communication between threads or matrix cores. Gluon, Triton's lower-level dialect, lets us fix this register layout, which Triton chooses itself. Each G_k is split into a diagonal and two shears, the diagonals of $G_1, \ldots, G_4$ merge into one per-channel scaling, and G_5 is applied directly: 112 packed FP16 operations per block, against 16 for per-channel scaling.

K1 and K2 satisfy the Triton `tl.dot` constraint $M, N, K \geq 16$ for all $B_R \geq 16$, since the GEMM shape is $(B_R, 32) \times (32, 32) \to (B_R, 32)$.

K1 and K2 thus form a compute-versus-memory study of the same dense operand: K1 pays parameter traffic, K2 pays construction, and the latency gap between them (Section 4.4) is the cost of computing each block at inference instead of loading it precomputed. K3 avoids the dense operand altogether.

4 Experiments

4.1 Experimental setup

Models. We evaluate on three diffusion transformers of increasing scale and architectural diversity:

- **PixArt-Σ** [Chen et al., 2024a] with 0.6 billion parameters, a single-stream architecture for text-to-image generation, sampled with 20 steps.
- **SANA** [Xie et al., 2025] with 1.6 billion parameters, a single-stream architecture for text-to-image generation with 1x1 convolution layers in the FFN module, sampled with 20 steps.
- **FLUX.1-schnell** [Black-Forest-Labs, 2024] with 12 billion parameters, a double-stream architecture for text-to-image generation, sampled with 4 steps.

Calibration dataset. Following the protocol of SVDQuant [Li et al., 2025] and LoRaQ [Bouquet et al., 2026], we use 128 text prompts drawn from the COCO 2017 validation captions [Chen et al., 2015] as calibration inputs. For each prompt a single forward pass is executed per linear layer to collect the activations used to optimize the transforms and, for the baselines, to calibrate the smoothing; the LoRaQ weight calibration itself uses no data.

Evaluation datasets. We report image quality on two different datasets, MJHQ-30K [Li et al., 2024] and sDCI [Urbanek et al., 2024], each with 5,000 prompts randomly sampled following the evaluation split used in prior work. The initial noise of each image is seeded from its prompt, so all methods denoise from identical noise and the comparisons are paired; Appendix G reports the variance across redrawn noise assignments.

Metrics. Following existing benchmarks, we evaluate performance on two criteria. To measure similarity to the 16-bit baseline, we use Learned Perceptual Image Patch Similarity (LPIPS) [Zhang et al., 2018] and Peak Signal-to-Noise Ratio (PSNR). To assess overall visual quality, we use Frechet Inception Distance (FID) [Heusel et al., 2018] and Image Reward (IR) [Xu et al., 2023].

Data format. We use the MXFP4 E2M1 format following the OCP Microscaling (MX) standard [Microsoft et al., 2023], which is motivated by the fact that current hardware and software stacks, including Advanced Micro Devices [2025] and emerging ML frameworks, natively support MX formats at this granularity, as also adopted by Bouquet et al. [2026]. The blockwise quantizer is defined in Section 3.1 and its implementation is described in Appendix F.

We provide a full description of the experiments for reproducibility in Appendix F.

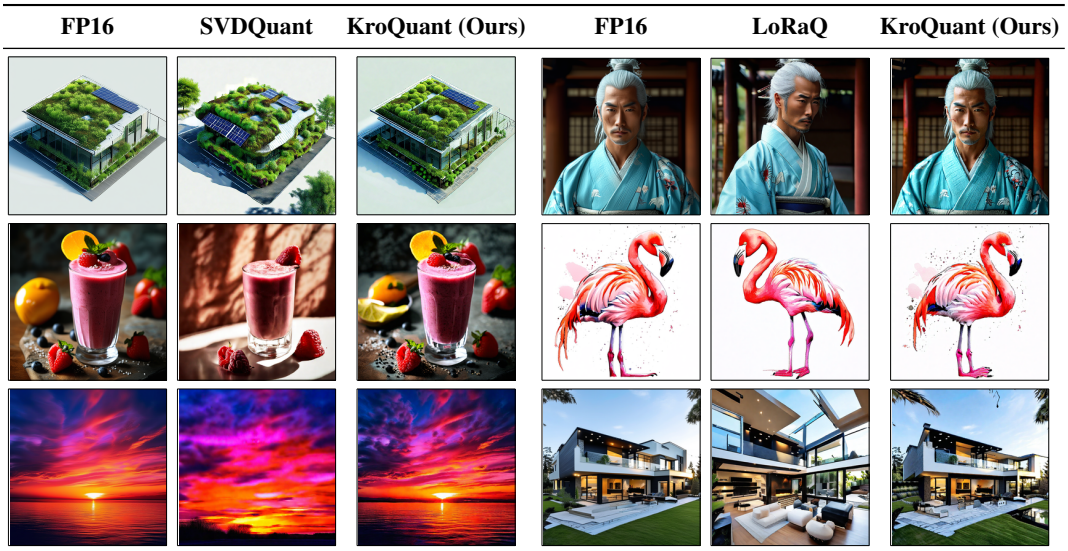
4.2 Main quantitative results

Table 1 compares KroQuant against prior PTQ methods on PixArt-Σ, SANA, and FLUX.1-schnell across two benchmarks. SVDQuant and LoRaQ use the smoothing calibration described in Li et al. [2025], Bouquet et al. [2026], and LoRaQ is used in its W4A4 configuration (LoRaQ-A in Bouquet et al. [2026]). KroQuant matches or improves upon both baselines on every metric, model, and benchmark. On MJHQ-30K, KroQuant reduces LPIPS over LoRaQ by 0.034 (relative 8.9%) on

Table 1: Image quality at W4A4 (MXFP4 E2M1) on MJHQ-30K and sDCI (5k samples each). LoRaQ is used in its W4A4 configuration (LoRaQ-A in Bouquet et al. [2026]): rank-128 MXFP4e2 low-rank branch with 16-bit low-rank activations. SVDQuant uses a rank-32, 16-bit low-rank branch. Best quantized result per model and dataset in **bold**; ties are all bold.

			MJHQ-30K				sDCI			
Model	Quantizer	LR Decomposition	FID ↓	IR ↑	LPIPS ↓	PSNR ↑	FID ↓	IR ↑	LPIPS ↓	PSNR ↑
PixArt-Σ	FP16 (reference)		16.6	0.948	—	—	24.8	0.966	—	—
	Smoothing	SVDQuant	19.3	0.876	0.433	15.2	23.3	0.904	0.476	13.9
	Smoothing	LoRaQ	17.5	0.952	0.381	16.4	23.7	0.970	0.399	15.0
	KroQuant (Ours)	LoRaQ	**16.9**	**0.989**	**0.347**	**16.5**	**21.9**	**1.01**	**0.378**	**15.2**
SANA	BF16 (reference)		16.2	1.09	—	—	22.4	1.07	—	—
	Smoothing	SVDQuant	**16.6**	1.07	0.256	17.1	23.8	1.02	0.285	15.4
	Smoothing	LoRaQ	17.4	**1.09**	0.229	17.7	24.1	1.03	0.259	15.9
	KroQuant (Ours)	LoRaQ	**16.6**	**1.09**	**0.206**	**18.4**	**23.6**	**1.04**	**0.241**	**16.4**
FLUX.1-schnell	BF16 (reference)		19.2	0.938	—	—	20.8	0.932	—	—
	Smoothing	SVDQuant	20.1	0.952	0.378	15.7	22.4	0.869	0.589	13.3
	Smoothing	LoRaQ	19.9	0.921	0.331	16.7	22.0	0.961	0.343	15.8
	KroQuant (Ours)	LoRaQ	**19.2**	**0.954**	**0.271**	**17.8**	**21.2**	**0.982**	**0.283**	**16.6**

Figure 2: Visual comparison on PixArt-Σ at MXFP4e2 W4A4. **Left three columns**: FP16 (reference), SVDQuant, KroQuant; KroQuant uses the LoRaQ backbone for offline weight calibration while SVDQuant uses its own SVD-based low-rank decomposition. **Right three columns**: FP16 (reference), LoRaQ leveraging the SmoothQuant quantizer, and KroQuant relying on the LoRaQ backbone.



PixArt-Σ, 0.023 (10.0%) on SANA, and 0.060 (18.1%) on FLUX.1-schnell, and it brings FID close to the 16-bit reference.

The sDCI benchmark confirms these trends, with the best FID, IR, LPIPS, and PSNR on all three models and an LPIPS reduction of 0.060 (17.5%) over LoRaQ on FLUX.1-schnell. On PixArt-Σ / MJHQ-30K, redrawing the sampling noise changes every metric by less than 0.5% (Appendix G).

FLUX.1-schnell interleaves text and image tokens in its double-stream blocks, but each stream goes through its own linear layers. Since KroQuant optimizes one transform per linear layer (Section 3.3), the two streams receive separate Kronecker factors without any architectural change.

Visual Results. Figure 2 shows a qualitative comparison of KroQuant against LoRaQ and SVDQuant on PixArt-Σ / MJHQ-30K. This figure shows that KroQuant produces sharper images with fewer artifacts than the baselines, consistent with the improved quantitative metrics. We provide additional visual comparisons in Appendix J.

4.3 Ablation study

Table 2 ablates the transform family on PixArt-Σ / MJHQ-30K. The upper group shows standalone transforms without LoRaQ; the lower group holds LoRaQ calibration fixed across all rows.

The standalone rows (without LoRaQ) isolate the transform contribution against the scaling method Smoothing: KroQuant alone reduces FID from 130.6 to 41.8, a large improvement but still far from full-precision quality. This confirms that LoRaQ weight calibration is essential for closing the

Table 2: Ablation: transform design choices on PixArt-Σ / MJHQ-30K (5k samples). All quantized rows use MXFP4 E2M1 with block size $n = 32$, W4A4. "–" in the LR Decomposition column indicates no offline weight-correction stage.

Quantizer	LR Decomposition	FID ↓	LPIPS ↓	PSNR ↑
Smoothing	–	130.6	0.712	13.0
KroQuant (Ours)	–	41.8	0.516	14.8
Smoothing	LoRaQ	17.5	0.381	16.4
Hadamard (32×32)	LoRaQ	17.5	0.382	15.7
KroQuant (Ours)	LoRaQ	**16.9**	**0.347**	**16.5**

remaining gap, while the transform provides a better starting point for weight calibration to operate on.

With LoRaQ, the block Hadamard-32 baseline matches per-channel smoothing on FID (both 17.5) but slightly degrades PSNR (15.7 vs. 16.4), indicating that a fixed Hadamard rotation does not uniformly reduce quantization error and can be mildly harmful for PSNR-sensitive layers. The gap between the fixed block Hadamard-32 and KroQuant, which starts from the same Hadamard and learns its factors, $\Delta\text{LPIPS} = 0.035$, $\Delta\text{PSNR} = 0.8$ dB, demonstrates that the 15 learnable parameters per block genuinely improve upon the fixed rotation, capturing per-layer numerical structure that a fixed orthogonal transform cannot adapt to.

4.4 Inference latency

We measure the wall-clock cost of the KroQuant block-diagonal transform (Section 3.5) against per-channel scaling (**Scale**, the cost floor of an online transform) and the Fast Walsh–Hadamard Transform (**FWHT** [Ashkboos et al., 2025, Liu et al., 2025, Tseng et al., 2024]).

Setup. All kernels run on an AMD Instinct MI350X in FP16, fuse a block-wise MXFP4e2 quantizer at their output, and are timed as medians over 500 repetitions after 100 warmup calls; reported latencies therefore reflect the full per-layer quantizer rather than the bare transform. The CDNA4 architecture of the MI350X supports the OCP MX formats natively [Advanced Micro Devices, 2025], the setting an MXFP4e2 quantizer targets. K3 and its matched scaling kernel are written in Gluon (Section 3.5), all other kernels in Triton [Tillet et al., 2019], which compiles a single source for GPUs of several vendors. **Q4** is the bare quantizer kernel and serves as a lower bound. FWHT requires power-of-two d, which forces zero-padding or truncation on production shapes, whereas KroQuant's kernels only require d divisible by 32; we therefore report production shapes here and compare against FWHT at power-of-two d in Appendix A.

Materialized blocks. Feeding the matrix cores requires the dense 32×32 block, and K1 and K2 measure what materializing it costs. K1 runs at parity with per-channel scaling on production shapes: its median speedup over Scale is $1.11\times$ (range 0.99–$1.31\times$), so one 32×32 matrix-core product per block adds no measurable cost over Scale's elementwise multiplies on these shapes. K2 reaches a median of $0.94\times$ (range 0.86–$1.02\times$) and is 5–37% slower than K1; this gap is the cost of rebuilding each block from its 15 parameters at inference instead of loading it precomputed. The faster K2 implementation depends on the shape: the split variant wins on seven of the eight shapes and the in-register variant only at $(m{=}512, d{=}3072)$ (Appendix A). At $(m{=}4096, d{=}12288)$ the bare quantizer alone takes $59.64\,\mu$s, leaving little headroom between any transform and the lower bound. Figure 3 shows the same behavior across token counts at $d = 4096$, where K1 and K2 are also 1.3–$3.2\times$ faster than FWHT.

Butterfly execution. K3 is compared with a Gluon per-channel scaling kernel that shares its code, tuning and quantizer epilogue, so that only the transform differs. Its median speedup over this kernel

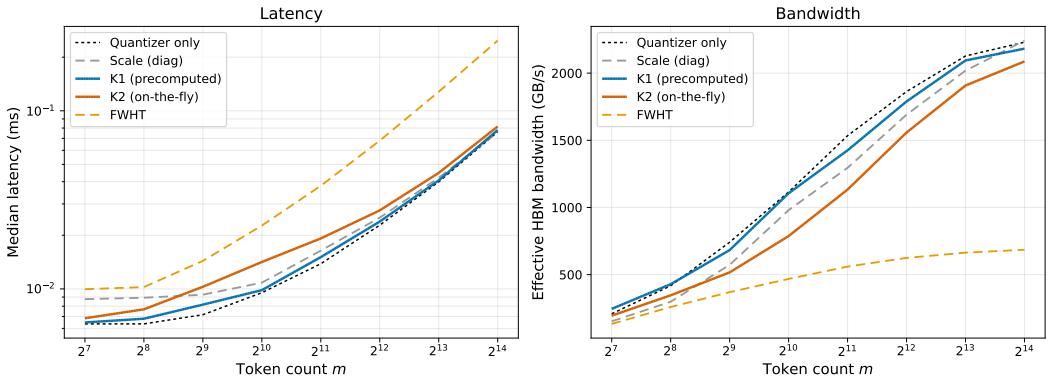


Figure 3: Median kernel latency and effective HBM bandwidth vs. token count m at $d = 4096$, MI350X, FP16, all kernels fusing the MXFP4e2 quantizer. Across m, K1 and K2 run at 0.97–$1.35\times$ and 0.77–$1.28\times$ the speed of Scale and are 1.5–$3.2\times$ and 1.3–$3.0\times$ faster than FWHT.

Table 3: Kernel latency (μs) on production layer shapes from FLUX.1 and PixArt-Σ, MI350X, FP16. All methods fuse the MXFP4e2 output quantizer; **Q4** is the bare quantizer kernel. K2 reports the faster of its in-register and split implementations per shape (Appendix A). Speedup ratios are reported as "Method A / Method B", meaning "Method B is $X\times$ faster than Method A" ($X < 1$ means slower). Medians are computed from the latencies.

m	d	Q4 (μs)	Scale (μs)	K1 (μs)	K2 (μs)	Scale/K1 Speedup	Scale/K2 Speedup
512	3072	6.72	9.04	7.64	8.84	$1.18\times$	$1.02\times$
512	12288	11.80	14.68	13.08	17.00	$1.12\times$	$0.86\times$
4096	1152	10.64	14.00	10.68	14.64	$1.31\times$	$0.96\times$
4096	3072	18.56	21.12	19.28	23.32	$1.10\times$	$0.91\times$
4096	3456	20.60	24.20	21.32	25.36	$1.14\times$	$0.95\times$
4096	4096	22.08	25.80	23.36	27.28	$1.10\times$	$0.95\times$
4096	4608	25.72	28.40	26.40	30.36	$1.08\times$	$0.94\times$
4096	12288	59.64	59.12	59.92	62.80	$0.99\times$	$0.94\times$
Median over the eight shapes						$1.11\times$	$0.94\times$

is $1.00\times$ on the eight shapes of Table 3 (range 0.94–$1.01\times$) and $0.99\times$ over 37 shapes (Appendix A): the learned butterfly runs at parity with a per-channel diagonal, without matrix cores or a dense block. The Gluon quantizer uses the native FP4 conversion of CDNA4 and the Triton one a software encoder, so we do not compare latencies across the two. The current limitation is portability: K3 depends on a fixed register layout and a CDNA4-specific instruction.

5 Conclusion and Limitations

We presented KroQuant, a post-training quantization method for diffusion transformers that introduces learned Kronecker-structured block transforms as an online activation preprocessing step. The key observation is that AdaLN prevents the offline weight-absorption used by LLM rotation methods, and that a full-dimensional online transform is too costly. A block-diagonal learned transform parameterized via Kronecker products of 2×2 factors fills this gap: the five factors of each block form a learnable butterfly with $O(d)$ parameters per layer, applied either as a materialized block on matrix cores or stage by stage.

The ablation confirms that learning the factors matters: starting from the block Hadamard, which matches per-channel smoothing on FID, optimizing 15 parameters per block reduces LPIPS by 0.035 and improves PSNR by 0.8 dB on PixArt-Σ. On an MI350X GPU, kernels that materialize the block for the matrix cores run close to parity with per-channel scaling (median $1.11\times$ precomputed and $0.94\times$ rebuilt on the fly over eight production shapes) and are 1.3–$3.2\times$ faster than FWHT at $d = 4096$; a kernel that executes the five-stage butterfly without forming the block also runs close to parity with a matched per-channel scaling kernel (median $1.00\times$).

KroQuant is complementary to weight-side corrections: the pipeline applies LoRaQ offline after the activation transforms are fixed, and the combination outperforms either component alone. On all three evaluated DiT architectures (PixArt-Σ, SANA, FLUX.1-schnell), KroQuant matches or improves on SVDQuant and on the W4A4 configuration of LoRaQ on every metric at MXFP4 E2M1 W4A4.

Limitations. The block size is tied to the MX format, and the $3 \log_2 n$ parameters per block cannot span $\mathrm{GL}(n)$; composing the butterfly with other transforms such as permutations is a natural extension. Joint multi-layer optimization of the transforms does not converge in our experiments (Section 3.3) and remains open, and timestep-aware calibration in the spirit of PTQ4DiT [Wu et al., 2024] is a natural direction. The butterfly kernel relies on an explicit register layout and a CDNA4-specific instruction, so unlike the Triton kernels it is not portable as written. Finally, KroQuant outputs tend to have higher contrast than the FP16 reference, an empirical effect whose theoretical interpretation lies beyond the scope of this paper.

Broader impacts. A broader-impacts discussion is provided in Appendix I.

Acknowledgments and Disclosure of Funding

This work was supported in part by the Swiss National Science Foundation via the grant 200020_214878. The main part of this work was done during a part-time contract of Yann Bouquet at AMD.

Competing interests. Alireza Khodamoradi and Kristof Denolf are affiliated with AMD, as listed on the first page. The authors declare no other competing interests.

References

Advanced Micro Devices. Amd instinct cdna 4 instruction set architecture. Technical report, Advanced Micro Devices, August 2025.

S. Ashkboos, A. Mohtashami, M. L. Croci, B. Li, P. Cameron, M. Jaggi, D. Alistarh, T. Hoefler, and J. Hensman. Quarot: outlier-free 4-bit inference in rotated llms. In *Proceedings of the 38th International Conference on Neural Information Processing Systems*, NIPS '24, Red Hook, NY, USA, 2025. Curran Associates Inc. ISBN 9798331314385.

Black-Forest-Labs. Flux.1, 2024. URL https://blackforestlabs.ai/.

Y. Bouquet, A. Khodamoradi, S. Y. Shen, K. Denolf, and M. Salzmann. Loraq: Optimized low rank approximation for 4-bit quantization. In *Advances in Neural Information Processing Systems*, 2026.

J. Chen, C. Ge, E. Xie, Y. Wu, L. Yao, X. Ren, Z. Wang, P. Luo, H. Lu, and Z. Li. Pixart-sigma: Weak-to-strong training of diffusion transformer for 4k text-to-image generation. In *Computer Vision – ECCV 2024: 18th European Conference, Milan, Italy, September 29–October 4, 2024, Proceedings, Part XXXII*, page 74–91, Berlin, Heidelberg, 2024a. Springer-Verlag. ISBN 978-3-031-73410-6. doi: 10.1007/978-3-031-73411-3_5. URL https://doi.org/10.1007/978-3-031-73411-3_5.

L. Chen, Y. Meng, C. Tang, X. Ma, J. Jiang, X. Wang, Z. Wang, and W. Zhu. Q-dit: Accurate post-training quantization for diffusion transformers, 2024b. URL https://arxiv.org/abs/2406.17343.

X. Chen, H. Fang, T.-Y. Lin, R. Vedantam, S. Gupta, P. Dollar, and C. L. Zitnick. Microsoft coco captions: Data collection and evaluation server, 2015. URL https://arxiv.org/abs/1504.00325.

E. Dellinger and A. Khodamoradi. Tensorcast: A casting/quantization library for pytorch. https://github.com/ROCm/tensorcast, 2025.

M. Heusel, H. Ramsauer, T. Unterthiner, B. Nessler, and S. Hochreiter. Gans trained by a two time-scale update rule converge to a local nash equilibrium, 2018. URL https://arxiv.org/abs/1706.08500.

D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

D. Li, A. Kamko, E. Akhgari, A. Sabet, L. Xu, and S. Doshi. Playground v2.5: Three insights towards enhancing aesthetic quality in text-to-image generation, 2024.

M. Li, Y. Lin, Z. Zhang, T. Cai, J. Guo, X. Li, E. Xie, C. Meng, J.-Y. Zhu, and S. Han. SVDQuant: Absorbing outliers by low-rank component for 4-bit diffusion models. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=vWR3KuiQur.

Z. Liu, C. Zhao, I. Fedorov, B. Soran, D. Choudhary, R. Krishnamoorthi, V. Chandra, Y. Tian, and T. Blankevoort. Spinquant: LLM quantization with learned rotations. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=ogO6DGE6FZ.

Microsoft, AMD, Arm, Intel, Meta, NVIDIA, and Qualcomm. Open compute project • ocp microscaling formats (mx) specification, 2023. URL https://www.opencompute.org/documents/ocp-microscaling-formats-mx-v1-0-spec-final-pdf.

Y. Shang, Z. Yuan, B. Xie, B. Wu, and Y. Yan. Post-training quantization on diffusion models. In *CVPR*, 2023.

Y. Sun, R. Liu, H. Bai, H. Bao, K. Zhao, Y. Li, J. Hu, X. Yu, L. Hou, C. Yuan, et al. Flatquant: Flatness matters for llm quantization, 2024.

P. Tillet, H. T. Kung, and D. Cox. Triton: An intermediate language and compiler for tiled neural network computations. In *Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages*, pages 10–19, 2019. doi: 10.1145/3315508.3329973.

A. Tseng, J. Chee, Q. Sun, V. Kuleshov, and C. De Sa. Quip#: even better llm quantization with hadamard incoherence and lattice codebooks. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

J. Urbanek, F. Bordes, P. Astolfi, M. Williamson, V. Sharma, and A. Romero-Soriano. A picture is worth more than 77 text tokens: Evaluating clip-style models on dense captions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 26700–26709, 2024.

J. Wu, H. Wang, Y. Shang, M. Shah, and Y. Yan. Ptq4dit: Post-training quantization for diffusion transformers. In *NeurIPS*, 2024.

G. Xiao, J. Lin, M. Seznec, H. Wu, J. Demouth, and S. Han. SmoothQuant: Accurate and efficient post-training quantization for large language models. In *Proceedings of the 40th International Conference on Machine Learning*, 2023.

E. Xie, J. Chen, J. Chen, H. Cai, H. Tang, Y. Lin, Z. Zhang, M. Li, L. Zhu, Y. Lu, and S. Han. SANA: Efficient high-resolution text-to-image synthesis with linear diffusion transformers. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=N8Oj1XhtYZ.

J. Xu, X. Liu, Y. Wu, Y. Tong, Q. Li, M. Ding, J. Tang, and Y. Dong. Imagereward: learning and evaluating human preferences for text-to-image generation. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 15903–15935, 2023.

R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. The unreasonable effectiveness of deep features as a perceptual metric. In *CVPR*, 2018.

T. Zhao, X. Ning, T. Fang, E. Liu, G. Huang, Z. Lin, S. Yan, G. Dai, and Y. Wang. Mixdq: Memory-efficient few-step text-to-image diffusion models with metric-decoupled mixed precision quantization, 2024. URL https://arxiv.org/abs/2405.17873.

T. Zhao, T. Fang, H. Huang, R. Wan, W. Soedarmadji, E. Liu, S. Li, Z. Lin, G. Dai, S. Yan, H. Yang, X. Ning, and Y. Wang. Vidit-q: Efficient and accurate quantization of diffusion transformers for image and video generation. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=E1N1oxd63b.

A Additional kernel measurements

Power-of-two dimensions and FWHT. Table 4 compares the kernels at power-of-two d representative of FLUX.1 and PixArt-Σ shapes, where FWHT applies without padding. As in Table 3, every kernel fuses the MXFP4e2 output quantizer and K2 reports the faster of its two implementations per shape.

K1 is at least as fast as Scale on every shape (1.00–$1.14\times$) and K2 runs at 0.75–$0.96\times$ the speed of Scale. K2 is 1.30–$2.45\times$ faster than FWHT and K1 1.77–$2.85\times$: FWHT's $\log_2 d$ butterfly stages couple the whole hidden dimension, whereas the block-diagonal transform confines each tile to one coalesced read and one coalesced write. At larger d the fused quantizer dominates kernel time (Q4 takes $74.44\,\mu$s of Scale's $75.44\,\mu$s at $d = 16384$), leaving little headroom for any transform.

Table 4: Kernel latency (μs) at power-of-two d, MI350X, FP16. All methods fuse the MXFP4e2 output quantizer; **Q4** is the bare quantizer kernel and a lower bound. K2 reports the faster of its in-register and split implementations per shape. Speedup ratios are reported as "Method A / Method B", meaning "Method B is $X\times$ faster than Method A" ($X < 1$ means slower).

m	d	Q4 (μs)	Scale (μs)	K1 (μs)	K2 (μs)	FWHT (μs)	Scale/K1 Speedup	Scale/K2 Speedup	FWHT/K2 Speedup
4096	1024	9.40	10.56	9.96	14.00	18.16	$1.06\times$	$0.75\times$	$1.30\times$
4096	4096	22.64	25.52	23.88	27.80	68.04	$1.07\times$	$0.92\times$	$2.45\times$
4096	16384	74.44	75.44	75.32	78.52	186.04	$1.00\times$	$0.96\times$	$2.37\times$
512	4096	7.20	9.24	8.12	10.28	14.40	$1.14\times$	$0.90\times$	$1.40\times$
512	16384	13.68	16.36	15.36	19.32	33.96	$1.06\times$	$0.85\times$	$1.76\times$

K2 implementations. K2 rebuilds each block from its 15 parameters (Section 3.5). The in-register implementation repeats this construction in every program that consumes the block; the split implementation builds each block once in a separate kernel and pays an extra launch and an HBM round trip. Table 5 reports both on the production shapes of Table 3.

Table 5: The two K2 implementations on production layer shapes, MI350X, FP16, fused MXFP4e2 quantizer. Latencies in μs; the margin is the slower latency divided by the faster one.

m	d	In-register	Split	Faster	Margin
4096	12288	116.56	62.80	split	$1.86\times$
4096	4608	47.32	30.36	split	$1.56\times$
4096	4096	42.40	27.28	split	$1.55\times$
4096	3456	37.08	25.36	split	$1.46\times$
4096	3072	33.56	23.32	split	$1.44\times$
4096	1152	16.24	14.64	split	$1.11\times$
512	12288	19.88	17.00	split	$1.17\times$
512	3072	8.84	11.60	in-register	$1.31\times$

The split implementation is faster on seven of the eight shapes, by 1.11 to $1.86\times$, and its margin grows with d at $m = 4096$. The in-register implementation is faster only at $(m{=}512, d{=}3072)$. Since the crossover depends on both m and d, we select the implementation per shape.

Butterfly kernel. Table 6 reports the speedup of K3 over the matched Gluon scaling kernel of Section 4.4 on 37 shapes, measured in one run with the timing protocol of Table 3. Per-shape ratios vary by up to 0.05 between runs; the median over the 30 power-of-two shapes is $0.99\times$ in both of our runs. Every output passes an SQNR check against the exact FP64 transform (threshold 17.5 dB).

Table 6: Speedup Scale/K3 of the butterfly kernel over the matched Gluon per-channel scaling kernel ($X < 1$ means K3 is slower), MI350X, FP16, fused MXFP4e2 quantizer. Top: production shapes of Table 3. Bottom: power-of-two shapes. Median over the 37 shapes: $0.99\times$ (range 0.92–$1.02\times$).

m	512	512	4096	4096	4096	4096	4096	4096
d	3072	12288	1152	3072	3456	4096	4608	12288
Scale/K3	1.00	0.95	0.96	1.00	0.94	1.01	1.00	0.99

$d\backslash m$	512	1024	2048	4096	8192	16384
1024	1.00	1.00	0.98	0.93	0.95	1.00
2048	1.00	0.98	0.94	0.96	1.00	0.99
4096	0.99	0.94	0.96	1.01	0.99	0.99
8192	0.95	0.96	1.02	0.98	0.99	1.00
16384	0.92	0.98	0.98	0.99	1.01	1.01

B K2 streaming-product kernel

The in-register implementation of K2 evaluates the Kronecker formula of Section 3.5 as a running product over the five factors $G_1, \ldots, G_5$; the split implementation runs the same construction in a separate kernel. Each step extracts one bit of the row index and one bit of the column index, looks up the corresponding entry of the current G_k via a vectorized scalar broadcast, and multiplies the running tile by the result element-wise.

```
i = tl.arange(0, 32)[:, None]   # row indices, shape (32, 1)
j = tl.arange(0, 32)[None, :]   # col indices, shape (1, 32)

# Step 1: K = G1[bit_4(r), bit_4(c)]
ir = (i >> 4) & 1; jc = (j >> 4) & 1
K = tl.where(ir == 0,
             tl.where(jc == 0, g1_00, g1_01),
             tl.where(jc == 0, g1_10, g1_11))

# Steps 2-5: K *= G_k[bit_{5-k}(r), bit_{5-k}(c)], k = 2, 3, 4, 5
for shift, (g00, g01, g10, g11) in zip(
        (3, 2, 1, 0),
        ((g2_00, g2_01, g2_10, g2_11),
         (g3_00, g3_01, g3_10, g3_11),
         (g4_00, g4_01, g4_10, g4_11),
         (g5_00, g5_01, g5_10, g5_11))):
    ir = (i >> shift) & 1; jc = (j >> shift) & 1
    K = K * tl.where(ir == 0,
                     tl.where(jc == 0, g00, g01),
                     tl.where(jc == 0, g10, g11))
```

Each `tl.where(ir==0, tl.where(jc==0, g_00, g_01), tl.where(jc==0, g_10, g_11))` broadcasts the four scalar entries of G_k across the $(32, 32)$ tile, selecting the correct one at every position from a single bit of the row and a single bit of the column. The element-wise multiplication accumulates the product $\prod_k G_k[r_{5-k}, c_{5-k}]$ that defines $K[r, c]$.

At each step only two (32×32) register tiles are live: the running product K and the temporary factor tile produced by `tl.where`. The temporary is consumed immediately and freed before the next step, so the construction never holds more than two 32×32 tiles, spread over the threads of the program. This small footprint leaves headroom to fuse the downstream MXFP4e2 quantizer (which requires its own bit-manipulation intermediates) into the same kernel.

C Normalization–rotation commutativity

For any invertible linear transform $T \in \mathrm{GL}(d)$ applied to a normalization's input to be absorbed offline into adjacent weight matrices, the normalization must satisfy $\mathrm{Norm}(xT)\,T^{-1} = \mathrm{Norm}(x)$. We test the orthogonal case $T = R \in O(d)$ as the easiest instance: any failure here generalizes to non-orthogonal but invertible T. Table 7 reports the maximum absolute deviation $\left\|\mathrm{Norm}(XR)R^\top - \mathrm{Norm}(X)\right\|_\infty$ for a random orthogonal $R \in \mathbb{R}^{64\times64}$ on a random batch $X \in \mathbb{R}^{32\times64}$ (seed 42). RMSNorm matches floating-point precision because its denominator depends only on $\|x\|_2$, which orthogonal rotations preserve; it would not commute with a non-orthogonal T. LayerNorm, BatchNorm, GroupNorm, AdaLN, and AdaRMSNorm all incur $O(1)$ deviation: the normalization either subtracts an axis-aligned mean, or applies a learned per-channel scale/shift, or both (AdaLN). Because both failure modes are axis-aligned, they cannot invert any cross-channel transform applied to the normalization's input; only per-channel input scaling can fold into the affine.

D Design space of activation transforms

Table 8 places the activation transforms discussed in Section 2 on common axes at $d = 4096$: number and size of the factors, support, parameters per layer, and online cost per token.

Table 7: Maximum absolute deviation between $\mathrm{Norm}(XR)R^\top$ and $\mathrm{Norm}(X)$ for common normalizations under a random orthogonal rotation R. Smaller is better; only RMSNorm commutes, and only with orthogonal R.

Normalization	Max abs. error	Commutes with $R \in O(d)$?
RMSNorm	1.7×10^{-6}	✓
LayerNorm	9.2×10^{-1}	✗
BatchNorm	1.3	✗
GroupNorm	2.1	✗
AdaLN	4.2	✗
AdaRMSNorm	4.4	✗

Table 8: Design space of activation transforms at $d = 4096$. Costs count multiply-accumulates (MAC) per token; fixed Hadamard transforms need additions and subtractions only. SpinQuant's cost assumes the transform runs online, as it must in DiTs. For KroQuant, the cost is given for the materialized block and for the butterfly.

Method	Factors	Factor size	Support	Params / layer	Cost / token
SmoothQuant [Xiao et al., 2023]	–	–	diagonal	4096 (1/ch)	4K MAC
QuaRot, QuIP# [Ashkboos et al., 2025, Tseng et al., 2024]	12	2	full d	0	49K add
SpinQuant [Liu et al., 2025]	1	d	full d	16.8M	16.8M MAC
FlatQuant [Sun et al., 2024]	2	$\sqrt{d} = 64$	full d	8192 (2/ch)	524K MAC
Hadamard-32 (fixed)	5	2	32×32 blocks	0	20K add
KroQuant	5	2	32×32 blocks	1920 (0.47/ch)	131K / 41K MAC

E A free orthogonal block instead of the Kronecker structure

A natural alternative to our parametrization is a free rotation in $\mathrm{SO}(32)$ for each block, for instance parametrized with the Cayley transform. It is a different constraint rather than a richer one. Our factors are volume-preserving: each fixes the product of its singular values but leaves the values themselves free, and a block has 15 degrees of freedom. $\mathrm{SO}(32)$ fixes every singular value to one and has 496 degrees of freedom. Neither family contains the other.

The deciding factor in DiTs is the online cost. Because AdaLN modulates the activations between the normalization and the linear layer (Appendix C), the transform cannot be folded offline as in SpinQuant [Liu et al., 2025] and runs at every denoising step. A free block must then either be stored dense, loading 1024 values per block at every step as K1 does, or be rebuilt from its 496 parameters, which for the Cayley map requires solving a 32×32 linear system per block, per layer, and per step, far beyond the five-factor construction of K2. The fixed Hadamard-32 row of Table 2 is the matched-cost control for the structure we keep: learning the 15 parameters improves LPIPS by 0.035 and PSNR by 0.8 dB over it.

F Implementation Details

Quantization. The activations forwarded to the low-rank branch are kept at 16-bit (full-precision). FP16 is used for PixArt-Σ while SANA and FLUX.1-schnell use BF16 by default. The activations forwarded to the residual branch and the residual weights are quantized to MXFP4e2 and a W4A4 matrix multiplication is applied. SVDQuant keeps the low-rank branch at full-precision for a rank of 32. LoRaQ, and KroQuant by extension, quantize the low-rank branch in MXFP4e2 with a rank of 128, for the same memory overhead as SVDQuant. All quantization operators (MXFP4e2, the FP8 E8M0 block scale, and the underlying MX block formatting) are implemented via the TensorCast library [Dellinger and Khodamoradi, 2025].

F.1 KroQuant

Calibration. For each linear layer we optimize the Kronecker-factored block transform T by minimizing the objective Equation 9 using Adam for 100 steps on the 128 calibration activations with a learning rate of 10^{-2}. The calibration samples are the same prompts as in Li et al. [2025], Bouquet et al. [2026]. 128 activations are randomly selected for each layer following the same seed

configuration as Li et al. [2025]. Those activations are kept and used across all experiments for fair comparison.

F.2 Smoothing

Smoothing Calibration Our method replaces the smoothing calibration used by our baselines. As a first calibration step, Li et al. [2025], Bouquet et al. [2026] apply per-channel smoothing [Xiao et al., 2023] with the layer-wise tuning of Li et al. [2025]: for each linear layer, a vector $\gamma \in \mathbb{R}^d$ is built component-wise as

$$\gamma_i = \frac{(\max_j |\boldsymbol{X}_{j,i}|)^\alpha}{(\max_j |\boldsymbol{W}_{i,j}|)^\beta}, \qquad i \in \{0, \dots, d-1\},$$

with the migration strengths α, β selected layer by layer to minimize the output mean-squared error of a downstream module that consumes the quantized layer's output. During this search, the weights are quantized through a rank-32 FP16 SVD truncation, and the search uses the same calibration dataset as KroQuant (Section 4.1).

F.3 LoRaQ

After KroQuant or Smoothing is fixed, offline LoRaQ weight calibration (Section 3.4) is applied with rank $r = 128$ in MXFP4e2 low rank branches for every layer.

Optimization. We optimize LoRaQ's objective for any experiment implying it with Adam [Kingma and Ba, 2014] using a learning rate of 10^{-3} and 500 steps per weight. The quantization-aware fine tuning of the low-rank components using a rotation transformation is done with a Cayley SGD algorithm at a learning rate of $5 \cdot 10^{-1}$ and 500 steps per weight, following Bouquet et al. [2026].

F.4 SVDQuant

Calibration. After the smoothing calibration, Li et al. [2025] calibrate their low-rank components using an iterative algorithm which we follow as well to reproduce the method in MXFP4e2: 100 candidates are recursively computed. An MSE loss is computed at the output level of the linear layer or transformer block, depending on the linear layer to calibrate. Each candidate is evaluated in the recursive order. A stopping algorithm returns the first encountered minimum local in the objective space with the associated low rank candidate which is then used as the low rank components for the auxiliary branch.

Compute. Calibration of all linear layers of one DiT (PixArt-Σ, SANA) runs on a single NVIDIA H100 within a few hours of wall-clock per model. Due to its size, FLUX.1-schnell required an NVIDIA H200 for calibration. In practice we used 4 GPUs at a time for smoothing, KroQuant calibration, and image generation. Kernel-latency benchmarks (Section 4.4) run on an AMD Instinct MI350X in FP16 mode, with kernels implemented in Triton 3.6.0 and its Gluon dialect against ROCm 7.2.2.

Asset licenses. We use the following external assets under their respective licenses: **PixArt-Σ** [Chen et al., 2024a] (CreativeML Open RAIL++-M); **SANA** [Xie et al., 2025] (Apache 2.0); **FLUX.1-schnell** [Black-Forest-Labs, 2024] (Apache 2.0); **MJHQ-30K** [Li et al., 2024]; **sDCI** [Urbanek et al., 2024]; **COCO 2017** [Chen et al., 2015] (CC BY 4.0 for annotations, with underlying images subject to their original Flickr licenses); and **TensorCast** [Dellinger and Khodamoradi, 2025] (MIT). All assets are used in accordance with their applicable license terms and restrictions.

G Variance across noise draws

The initial noise of each image is drawn from a seed derived from its prompt. A run is therefore exactly reproducible, and all methods denoise from the same noise for a given prompt, so the comparisons of Table 1 are paired by construction. A fixed assignment, however, yields a single realization of each aggregate metric. To measure how much the metrics move when the sampling noise changes, we add a constant offset k to every prompt seed, which redraws the initial noise of every image while

keeping the assignment shared across methods. We regenerate and re-score the KroQuant pipeline on PixArt-Σ / MJHQ-30K (MXFP4e2, W4A4, 5,000 images per draw) under four assignments: offset 0 is the default assignment used in Table 1, and offsets 1 to 3 redraw it.

Table 9: KroQuant on PixArt-Σ / MJHQ-30K under four noise assignments (seed offsets 0 to 3). Std is the sample standard deviation over the four draws; CoV is the coefficient of variation.

Metric	Offset 0	Offset 1	Offset 2	Offset 3	Mean	Std	CoV
FID ↓	16.841	16.771	16.829	16.669	16.7775	0.0785	0.47%
IR ↑	0.9870	0.9868	0.9897	0.9908	0.9886	0.0020	0.20%
LPIPS ↓	0.3490	0.3528	0.3499	0.3500	0.3504	0.0016	0.47%
PSNR ↑	16.518	16.491	16.520	16.545	16.5185	0.0221	0.13%

Across the four draws, the coefficient of variation stays below 0.5% on all four metrics, and every draw improves on the Smoothing + LoRaQ and SVDQuant rows of Table 1 on all four metrics.

H Joint multi-layer optimization

Layer-wise optimization (Section 3.3) keeps each objective well conditioned. When several layers are optimized jointly, the five factors compose within each layer and the layers compose across depth, all through the straight-through estimator, and a single a_k approaching the pole at zero corrupts its layer and every block downstream. We tested whether removing the pole suffices, using the bounded reparametrization $a_k = s_0 + s\tanh(\theta_k)$ with $s_0 = 1$ and $s = 0.9$, which confines a_k to $(0.1, 1.9)$, away from zero, and preserves the Hadamard initialization exactly. At the pathological configuration, it lowers the gradient norm from 2.98×10^{14} to 1.90×10^{2} and the block condition number from 9.97×10^{29} to 7.48×10^{9}. We then optimize PixArt-Σ jointly for 2000 epochs per block, twenty times the layer-wise budget, at four learning rates (Table 10).

Table 10: Joint optimization with the bounded reparametrization on PixArt-Σ: loss of transformer block 3 at fixed fractions of training (2000 epochs per block). Net is the relative decrease of the loss from start to end (positive is better). Block k's objective depends on the outputs of blocks 0 to $k-1$, so a diverged block ends the run.

Learning rate	0%	25%	50%	75%	100%	Net
10^{-2}	not reached: block 2 diverges (NaN) at epoch 468					
5×10^{-3}	block 3 diverges (NaN) at epoch 779					
10^{-3}	2.7547	2.7278	2.7537	2.7523	2.7421	+0.46%
10^{-4}	2.4054	2.3796	2.4090	2.4228	2.4192	−0.57%

The two larger learning rates diverge, and the two that complete block 3 do not optimize it: the loss oscillates and ends within 0.6% of its starting value, slightly lower at 10^{-3} and slightly higher at 10^{-4}. The response to the step size is not monotone either: block 0 improves by 67.5% at 10^{-2} but degrades by 89.6% at the smaller 5×10^{-3}, so a working learning rate cannot be found by bisection. A well-conditioned parametrization is therefore not sufficient for joint optimization through the straight-through estimator, which we leave as an open problem.

I Broader Impacts

KroQuant accelerates inference of three publicly released diffusion transformers (PixArt-Σ, SANA, FLUX.1-schnell) without altering their generative capabilities, training data, or safety tuning. We discuss both directions of its societal impact and the boundary of what changes relative to a baseline FP16 deployment of the same upstream checkpoints.

Positive consequences. A method that delivers near-FP16 image quality at W4A4 with online transforms at the cost of per-channel scaling reduces the per-inference energy and memory cost of

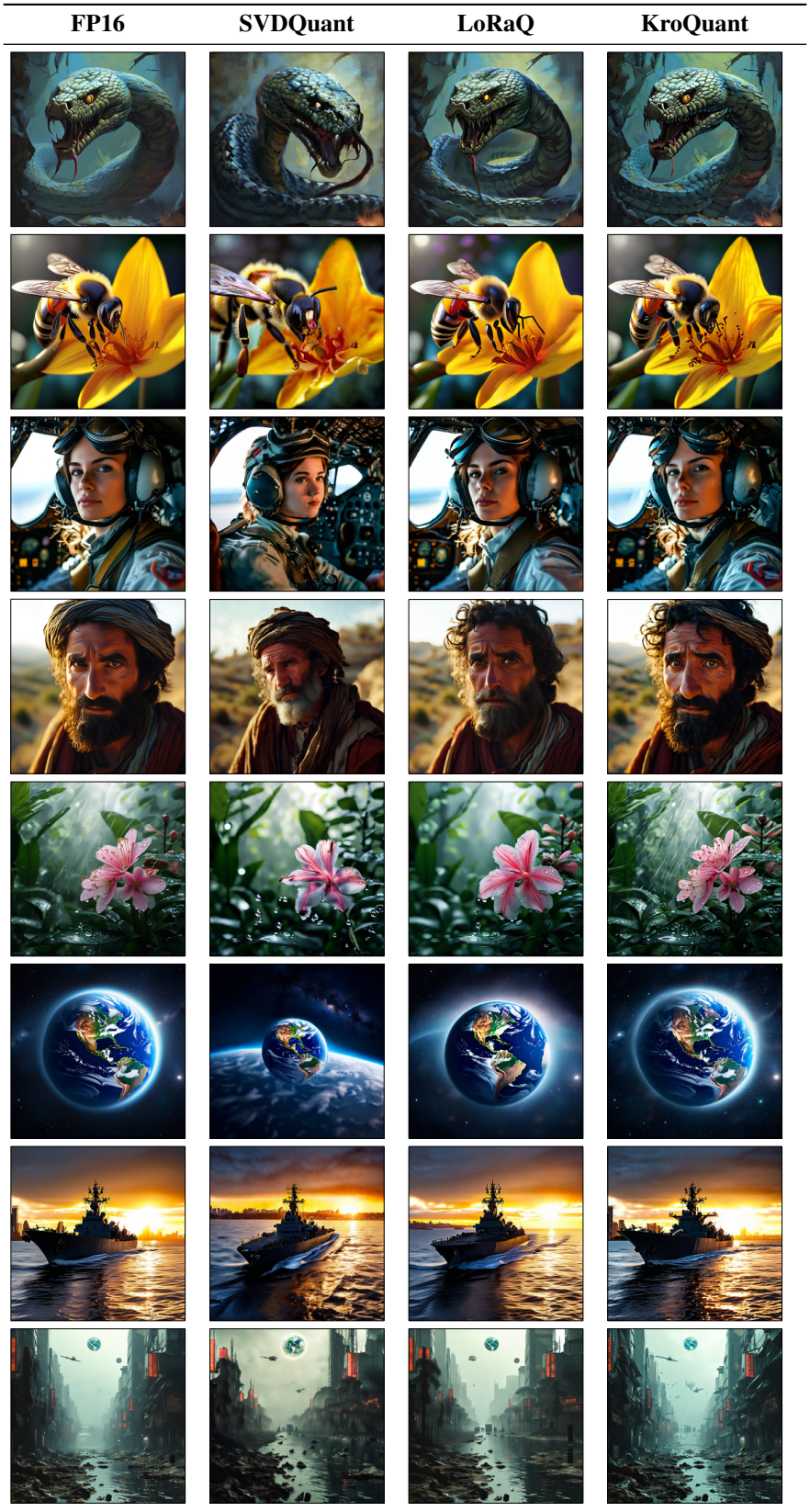
large text-to-image models. The resulting deployment cost reduction broadens access to current state-of-the-art image generation: lower hardware tiers become viable for academic research, classroom use, and applied work in resource-constrained settings, and large-scale services see a smaller carbon footprint per generated sample. By keeping the rotation parameter count at $O(d)$ and avoiding any retraining, KroQuant also lowers the engineering and compute barrier to deploying public DiTs on commodity accelerators.

Negative consequences. The same efficiency improvement makes the dual-use risks of upstream diffusion transformers cheaper to operate at scale. Non-consensual deepfakes, targeted disinformation, and copyright-infringing synthesis become accessible to actors that previously could not afford the inference cost, and a sub-16-bit pipeline that runs on a single commodity accelerator widens the population of potential misuse operators relative to an FP16 deployment.

Boundary of incremental impact. KroQuant introduces no new generative capability, no new modality, no relaxation of safety tuning, and no new dataset or pretrained-model release at submission. The qualitative threat model of the upstream checkpoints (PixArt-Σ under OpenRAIL-M, SANA and FLUX.1-schnell under Apache 2.0) is therefore unchanged; only the cost curve of operating those models shifts. Responsibility for misuse mitigation remains with the original upstream license terms, which already bind the released weights and are not relaxed by quantization. We treat KroQuant's incremental societal impact as bounded by that of the upstream models it accelerates, while acknowledging that any inference-cost reduction is amplified at deployment scale.

J Additional Visual Results

Figure 4: Comparison of images generated by PixArt-Σ across different quantization configurations following Table 1 settings.



NeurIPS Paper Checklist

1. **Claims**

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

   Answer: [Yes]

   Justification: The abstract and Section 1 state KroQuant's three contributions (a learned non-diagonal online activation transform compatible with AdaLN, a Kronecker-LU parametrization at $0.47\,d$ scalars per layer, and fused quantizer kernels that run close to parity with per-channel scaling on MI350X, with median speedups of $1.11\times$ and $0.94\times$ over eight production shapes) and explicitly bound the scope to MXFP4e2 W4A4 PTQ on three DiT architectures. Each claim is supported by a corresponding section in the body (Section 3.2 for the transform, 1 for quality, 3 for kernel speed).

   Guidelines:

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

2. **Limitations**

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

   Answer: [Yes]

   Justification: Section 5 contains a dedicated `\paragraph{Limitations.}` block: the block size is tied to the MX format and the Kronecker family does not span $\mathrm{GL}(n)$, joint multi-layer optimization of the transforms does not converge in our experiments and remains open, calibration is not timestep-aware, the butterfly kernel relies on an explicit register layout and a CDNA4-specific instruction, and KroQuant outputs show a higher contrast than the FP16 reference.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.

- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

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

   Answer: [N/A]

   Justification: The paper states no formal theorem, proposition, or lemma. The normalization-vs-rotation commutativity claim used in Section 1 is supported by an empirical maximum-deviation table in Appendix C, not a proof.

   Guidelines:

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

4. **Experimental result reproducibility**

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

   Answer: [Yes]

   Justification: Section 4.1 and Appendix F disclose the optimizer (Adam) and per-stage step counts (100 for KroQuant, 500 for LoRaQ, 500 for the Cayley-SGD rotation tuning), all learning rates (10^{-2}, 10^{-3}, $5 \cdot 10^{-1}$), the calibration set (128 prompts from COCO 2017 validation under a fixed-seed protocol), the data format (MXFP4e2 with block size $n = 32$ following the OCP MX standard), the rank settings (LoRaQ $r = 128$, SVDQuant baseline $r = 32$), the sampling steps (20 for PixArt-Σ and SANA, 4 for FLUX.1-schnell), and the per-method calibration recipes for KroQuant, Smoothing-with-LoRaQ, and SVDQuant. The K2 kernel implementation is described with pseudocode in Appendix B.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.

- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

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

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

   Answer: [Yes]

   Justification: The code is available at `https://github.com/AMDResearch/KroQuant`. The evaluation datasets (MJHQ-30K, sDCI) and the calibration prompts (COCO 2017) are public, and the method, hyperparameters, and per-method recipes are described in Section 3 and Appendix F.

   Guidelines:

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

6. **Experimental setting/details**

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

   Answer: [Yes]

   Justification: Section 4.1 and Appendix F fully specify the calibration protocol (Adam, step counts, learning rates), the data format (MXFP4e2, block size 32, FP8 E8M0 scale), the rank/bitwidth pairs per method, and the fixed-seed evaluation protocol on 5,000 samples each from MJHQ-30K and sDCI.

   Guidelines:

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

7. **Experiment statistical significance**

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

   Answer: [Yes]

   Justification: The main tables report one run per configuration under a per-prompt seeding that is identical across methods, so the comparisons are paired. Appendix G reports the mean, sample standard deviation, and coefficient of variation over four noise assignments for KroQuant on PixArt-Σ / MJHQ-30K (coefficient of variation below 0.5% on all four metrics). Kernel latencies (Table 3) are medians over 500 timed repetitions after 100 warmup calls.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
   - If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

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

   Answer: [Yes]

   Justification: KroQuant transform optimization and LoRaQ weight calibration run on a single NVIDIA H100 GPU for PixArt-Σ and SANA and on an NVIDIA H200 for FLUX.1-schnell, within a few hours of wall-clock per model; in practice four GPUs ran smoothing, calibration, and image generation in parallel (Appendix F). Kernel-latency benchmarks (Section 4.4) run on an AMD Instinct MI350X in FP16.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.

- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

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

   Answer: [Yes]

   Justification: We have read the NeurIPS Code of Ethics and confirm that the research conducted in this paper conforms with it.

   Guidelines:

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

10. **Broader impacts**

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

    Answer: [Yes]

    Justification: Section 5 flags a broader-impacts discussion which is developed in Appendix I. The appendix discusses both directions, lower per-inference energy and broader access on the positive side, cheaper dual-use misuse (deepfakes, disinformation, copyright-infringing synthesis) on the negative, and bounds the incremental societal impact: KroQuant introduces no new generative capability, no new modality, and no new model release, so the qualitative threat model of the upstream public DiTs (PixArt-Σ, SANA, FLUX.1-schnell) is unchanged and only their cost curve shifts.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

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

Answer: [N/A]

Justification: The paper releases no new generative model weights or scraped datasets. KroQuant is a quantization method applied to existing publicly released checkpoints (PixArt-Σ, SANA, FLUX.1-schnell); the inherent misuse risk of those models is governed by their original licenses and is not amplified by the proposed method.

Guidelines:

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

12. **Licenses for existing assets**

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

Answer: [Yes]

Justification: All external assets are cited at first use and their licenses are listed in Appendix F, including the model checkpoints PixArt-Σ [Chen et al., 2024a], SANA [Xie et al., 2025], and FLUX.1-schnell [Black-Forest-Labs, 2024]; the evaluation datasets MJHQ-30K [Li et al., 2024] and sDCI [Urbanek et al., 2024]; the calibration captions from COCO 2017 [Chen et al., 2015]; and the TensorCast quantization library [Dellinger and Khodamoradi, 2025].

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

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

Answer: [Yes]

Justification: The only new asset is the code, released at `https://github.com/AMDResearch/KroQuant` with its documentation and license in the repository. The paper releases no new dataset or pretrained model.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.

- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

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

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

    Answer: [N/A]

    Justification: The paper involves no crowdsourcing or human-subjects research. All evaluation uses automated metrics (FID, IR, LPIPS, PSNR) on synthetic image samples.

    Guidelines:

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

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

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

    Answer: [N/A]

    Justification: The paper involves no human-subjects research, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

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

    Answer: [N/A]

    Justification: LLMs were used only for writing and editing assistance. They are not part of the proposed method, the calibration pipeline, the kernel implementations, or the experimental analysis; per the checklist guideline, declaration is therefore not required.

Guidelines:

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